Support ECN in Multi-Queue Datacenter Networks via per-Port Marking with Selective Blindness

Yawen Pan[†], Chen Tian[†], Jiaqi Zheng[†], Gong Zhang[‡], Hengky Susanto[‡], Bo Bai[‡], Guihai Chen[†]
[†]*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China*
[‡]*Future Network Theory Lab, Huawei, Hong Kong*

***Abstract*—ECN is a powerful tool that can achieve low latency and high throughput simultaneously. Support ECN for multi-queue scenarios is an industry trend in datacenter networks. However, ECN schemes developed for per-port marking cannot be applied directly to the multi-queue scenarios. It hurts at least one metric among latency, throughput, and the scheduling policy. State-of-the-art multi-queue ECN marking schemes each has its own limitations. In this paper, we present *per-Port Marking with Selective Blindness* (PMSB). The intuition is that: if a flow is found to be a victim of per-port marking, we can either revoke the marking or cancel the flow back-off even if its packets qualify the per-port threshold (*i.e.*, selective blindness). By breaking the fixed causal relationship between ECN marking and flow back-off, flows from un-congested queues can be protected. We evaluate PMSB with large-scale NS-3 simulations. Our results demonstrate that PMSB can preserve a given scheduling policy. Compared with the current practice, PMSB can reduce the average/99% completion time for small flows by 64.49%/72.89% respectively while delivering a slightly better performance for large flows.**

I. Introduction

In nowadays data centers, ECN is a powerful tool that can achieve low latency and high throughput simultaneously [1], [12], [16], [17], [13], [14], [15], [9]. It requires both ECN-enabled switches in the network and ECN-based transport protocols at end hosts. A switch puts an ECN mark on a passing packet if the average/instantaneous buffer length exceeds a marking threshold. Meanwhile, a transport protocol increases/decreases the congestion window (*e.g.*, DCTCP [1] and D2TCP [16]) or transmission rate (*e.g.*, DCQCN [18]) according to the occurance/ratio of ECN-marked packets. Usually, the buffer length can be kept around the marking threshold without sacrificing throughput.

Support ECN for multi-queue scenarios is an industry trend in data centers. Commodity switches usually support 4-8 service queues per port. The current practice is to isolate different services to different queues so that differentiated performance can be provisioned [8], [4], [11], [7].

However, ECN schemes developed for per-port marking cannot be applied directly to the multi-queue scenarios. It hurts at least one metric among latency, throughput, and the scheduling policy [5]. Per-queue marking with the standard ECN threshold ensures high throughput but incurs high latency when most queues are active. Dividing the threshold among queues according to their weights ensures low latency but incurs throughput loss when there are few active queues. Directly using per-port ECN marking maintains the performance of both latency and throughput, but may violate the scheduling policy (*e.g.*, weighted fair sharing).

State-of-the-art multi-queue ECN marking schemes each has its own limitations (Section II). MQ-ECN [5] dynamically calculates ECN marking threshold for each queue. It only works for round-based schedulers (*e.g.*, WRR and DWRR) and does not support others (*e.g.*, WFQ and SP) that do not possess the concept of "round". Instead of buffer length, TCN [3] uses each packet's sojourn time as the threshold metric. Due to its duration-based nature, it cannot accelerate the delivery of congestion information via dequeue ECN marking [17]. As a result, its performance is also sub-optimal.

We take one step back and ask a fundamental question: *can we enable ECN support in multi-queue datacenter networks via per-port marking?* Let's revisit the hurt to the scheduling policy (*e.g.*, weighted fair sharing) brought by per-port ECN marking. It is claimed that "...packets from one queue may get marked due to buffer occupancy of the other queues belonging to the same port..." [5].

In this paper, we present *per-Port Marking with Selective Blindness* (PMSB). The intuition of PMSB is that: if a flow is found to be a victim of per-port marking, we can either revoke the marking or cancel the flow back-off even if its packets qualify the per-port threshold (*i.e.*, selective blindness). Here, a victim means a flow that does not experience congestion, but the packets of this flow are marked due to buffer occupancy from other queues of the same port. By breaking the fixed causal relationship between port marking and back-off, flows from un-congested queues can be protected. With PMSB, a switch marks ECN only if both conditions hold: (1) the port buffer length is larger than a per-port threshold (*i.e.*, port marking), and (2) the queue buffer length is also no less than a per-queue filter threshold (*i.e.*, selective blindness).

The challenge is how to achieve the best trade-off point of a selective blindness algorithm by determining the per-queue threshold. If the algorithm is too aggressive to accept ECN marking, there could be *false positive* decisions (*i.e.*, a flow accepts marking while its queue is actually not congested) which would hurt the scheduling policy. If the algorithm is too conservative to accept ECN marking, there could be *false negative* decisions (*i.e.*, a flow refuses marking while its queue is actually congested) which could hurt latency.

Through extensive experiments, we find that the *necessary condition* for the hurt (to happen) is that victim flows should be *extensively* marked. Otherwise, victim flows with a low

ECN marking ratio would not back-off too much, and the scheduling policy may still get respected. Thus, a selective blindness decision can be relatively aggressive to trade a small probability of false positive for the elimination of false negative.

Based on this observation, we derive bounds for per-queue filter threshold. We then perform steady state analysis for the algorithm. As PMSB requires switch modification, we also develop an heuristic end-host version PMSB(e) as an immediately-deployable alternative. With PMSB(e), a flow compares its current round-trip-time (RTT) with the base RTT and decides whether it should accept a marking ECN signal.

We evaluate PMSB with large-scale NS-3 simulations. Our results demonstrate that PMSB can preserve a given scheduling policy, at the same time achieve both low latency and high throughput. Compared with TCN with generic packet scheduling, PMSB can reduce the average/99% completion time for small flows by 64.49%/72.89% respectively while delivering a slightly better performance for large flows. When compared with MQ-ECN, the performance gain is smaller but still significant. The average/99% completion time for small flows can be reduced by 40.00%/41.21% respectively. Although the performance of PMSB(e) is lower than that of PMSB, its improvement of average/99% completion time for small flows can be 25.03%/25.81% respectively when compared with MQ-ECN.

II. Motivation

We begin by introducing the background of ECN marking and scheduling policies of multiple queues supported by existing commodity switching chips. Then, we briefly introduce the challenges to multi-queue ECN marking. At last, we present existing solutions and discuss their limitations.

A. Background

ECN marking: In current data centers, DCTCP [1] is widely used for port-level ECN marking [15], [9]. DCTCP uses a special parameter setting of RED [6] ECN marking. Specifically, an ECN marking threshold K (*i.e.*, standard ECN marking threshold) is given in a switch. A switch puts an ECN mark on a passing packet if the instantaneous occupied buffer length of this port exceeds K. At end hosts, DCTCP increases/decreases the congestion window of each connection according to its ratio of ECN-marked packets. Usually, the buffer length can be kept around the marking threshold without sacrificing throughput.

To fully utilize the link bandwidth and keep low latency, prior works [17], [2] suggest:

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

The theoretical model considers synchronized flows with identical round-trip times sharing the only queue of a bottleneck link. Here C is the link bandwidth, RTT is the average round-trip time, and λ is a tunable parameter that decided by congestion algorithms. Note that in production datacenter networks, RTT is relatively stable and can be measured through large-scale measurements.

Multi-queue scheduling: Existing commodity switching chips support multiple (typically 4-8) physical queues per port. Datacenter operators usually use multiple queues to isolate different services to provide Quality-of-Service differentiation. Typical multi-queue scheduling policies include Weighted Fair Queuing (WFQ), Strict Priority (SP), Weighted Round Robin (WRR), and Deficit Weighted Round Robin (DWRR) *etc.*

Commodity switching chips provide per-queue, per-port, and per service pool ECN markings capability. The main difference among them is that they use different buffer entities to decide marking or not. In per-queue ECN marking, each queue is assigned a threshold *Queue-K* independently to other queues, and marking packet when the length of queue buffer occupancy exceeds *Queue-K*. In per-port ECN marking, each port is assigned a single threshold *Port-K*, and marking packet when the length of port buffer occupancy exceeds *Port-K*. Similarly, in per service pool ECN marking, packets are marked when total buffer occupancy in a shared buffer pool exceeds per service pool marking threshold.

B. Challenges to multi-queue ECN support

Given weight values to each queue, how to choose marking strategy is challenging. To confirm these observations, we perform experiments of weighted share among queues similar to that in MQ-ECN [5].

Per-queue ECN with standard threshold: We can configure each queue with a standard threshold to achieve high throughput, *e.g.*, $C \times RTT \times \lambda$. We run a small simulation. We set the per-queue standard threshold to 16 packets, and start 8 flows from 8 senders to the same receiver. We vary the number of switch queues from 1 to 8, and classify flows into these queues evenly by setting Differentiated Services Code Point (DSCP) IP field. The link capacity of the bottleneck link is 10 Gbps.

Figure 1 shows the distribution of RTT. With the increasement of the number of queues, the RTT also increases rapidly. This is because if there are many active queues with relatively large threshold, the queuing delay increases highly. This confirms the conclusion that per-queue ECN marking with standard threshold can cause high latency [5].

Per-queue ECN with fractional threshold: In contrast to per-queue ECN with standard threshold, we can apportion the standard threshold among all the queues according to their fair share weights to achieve low latency. For example, suppose each switch port has N queues, and the weight of $queue_i$ is W_i. So the fractional threshold K_i of $queue_i$ is set as follows:

$$K_i = \frac{W_i}{\sum_{j=1}^{N} W_j} \times C \times RTT \times \lambda. \tag{2}$$

Apparently, this configuration ensures low latency, because it maintains a small buffer occupancy in switch. In this experiment, we configure per-queue ECN threshold as 2 and 16 packets respectively. We start only one flow, and observe its throughput.

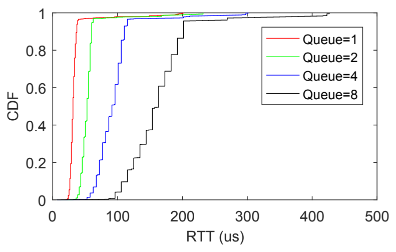

Fig. 1. Per-queue marking (standard threshold)

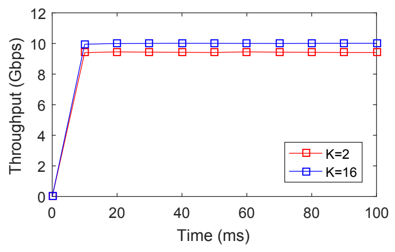

Fig. 2. Per-queue marking (fractional threshold)

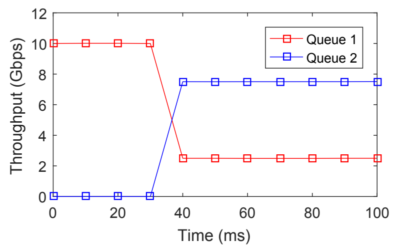

Fig. 3. Per-port marking (1 flow v.s. 8 flows)

Figure 2 shows the throughput. When we set per-queue ECN threshold as 16 packets, the throughput of this flow can reach about 10 Gbps. However, when we set per-queue ECN threshold as 2 packets, the throughput of this decrease about 6%. Thus, we can confirm that per-queue ECN marking with fractional threshold can degrade the utilization of link when there are few active queues.

Per-port and per service pool ECN: Per-port ECN with standard threshold can achieve both high throughput and low queuing latency. In this experiment, we set the per-port threshold to 16 packets, and we build 9 flows from 9 senders to the same receiver. We classify these flows into 2 switch queues with 1:1 weight. Here Queue 1 has 1 flow, and Queue 2 has 8 flows.

Figure 3 shows the throughput of queues. The throughput of Queue 1 and Queue 2 are about 2.49 Gbps and 7.51 Gbps respectively. Obviously, the flow of Queue 1 is the victim. Per-port ECN marking can violate weighted fair sharing among different queues belonging to the same port. It cannot ensure isolation between queues of the same port. Packets from one queue may be marked due to shared buffer occupied by other queues of the same port.

We believe per service pool will also violate weighted fair sharing, because queues belonging to different ports may interfere with each other.

C. *Related Work*

Cater to these challenges, MQ-ECN and TCN have been proposed. MQ-ECN [5] and TCN [3] can achieve high throughput, low latency simultaneously, but each has its own limitations.

MQ-ECN: Per-queue ECN with standard/fractional threshold keeps a static threshold. MQ-ECN instead keeps a dynamic threshold K_i for each queue respectively as follows:

$$K_i = min(\frac{quantum_i}{T_{round}}, C) \times RTT \times \lambda, \tag{3}$$

where RTT and λ are known, $quantum_i$ is the weight of queue i that is used to keep weighted fair sharing. T_{round} is the total time to serve all queues once. In one T_{round} time, the size of data that dequeued by $queue_i$ cannot exceed $quantum_i$. Thus, $\frac{quantum_i}{T_{round}}$ indicates the drain rate of $queue_i$, and the drain rate of $queue_i$ should not exceed link capacity C.

T_{round} balances throughput and latency automatically. When there are more queues whose input rates exceed their weighted fair share rates, T_{round} becomes larger, then K_i becomes smaller to keep low latency. When there are fewer queues whose drain rates reach their weighted share rates, T_{round} becomes smaller, then K_i becomes larger to achieve high throughput. Since the threshold of each queue is dynamically changing, it can achieve high throughput and low latency.

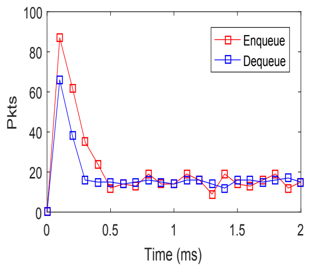

Fig. 4. DCTCP enqueue/dequeue marking

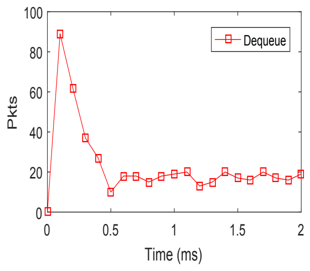

Fig. 5. TCN cannot accelerate

However, MQ-ECN only supports round-based schedulers like WRR and DWRR. It does not support other schedulers such as WFQ and SP that do not possess the concept of "round".

TCN: The goal of TCN is to enable ECN over generic packet schedulers, and maintain good network performance. It changes Equation (3) as follows:

$$\frac{K_i}{min(\frac{quantum_i}{T_{round}}, C)} = RTT \times \lambda. \tag{4}$$

The left side of Equation (4) means the largest sojourn time that packets can experience if they would not be marked. Thus, TCN use $RTT \times \lambda$ as threshold T_k. If the sojourn time of one packet exceeds T_k, it will be marked. In order to calculate sojourn time, TCN attaches a time stamp $t1$ to each packet at the enqueue time. At the dequeue time, TCN uses current time stamp $t2$ minus $t1$ to get its sojourn time. Obviously, TCN does not limit itself to round-based schedulers, and it can be used over generic schedulers.

However, TCN cannot accelerate the delivery of congestion information with dequeue marking. This is because TCN finds congestion only when packets experience congestion. While dequeue-based ECN/RED can deliver congestion information earlier.

TABLE I
COMPARE MQ-ECN, TCN AND PMSB.

	MQ-ECN	TCN	PMSB	PMSB(e)
Generic scheduler	×	√	√	√
Round-based scheduler	√	√	√	√
Early notification	√	×	√	√
No switch modification	×	×	×	√

To understand this drawback, we start 4 flows from 4 senders to the same receiver, and we set the threshold as 16 packets for DCTCP and 19.2 μs for TCN. As the bandwidths of all links are 1Gbps, draining 16 packets(1502 Bytes/packet) needs 19.2μs for switch. All traffic is classified into the same switch queue. Figure 4 shows the buffer occupancy of DCTCP. At the beginning, there are peak buffer occupancies. This is because TCP congestion window grows exponentially during the slow start phase before ECN takes effect. The peak value of DCTCP reaches 87 packets when marking CE codepoint at enqueue time. As a comaprison, when marking CE codepoint at dequeue time, the peak value of DCTCP decreases about 25%. This is because when marking CE codepoint at dequeue time, the sender of DCTCP can receive congestion feedback earlier. However, TCN has a duration-based nature. Figure 5 shows the buffer occupancy of TCN which confirms that TCN can not deliver congestion information earlier.

III. PMSB OVERVIEW

Intuition: Let's first revisit the hurt to the scheduling policy (*e.g.*, weighted fair sharing) brought by per-port ECN marking. It is claimed that "...packets from one queue may get marked due to buffer occupancy of the other queues belonging to the same port..." [5]. This gives us the intuition: *what if we can identify a victim flow and ignore the faulty ECN marking signals appear in its packets?*.

We present *per-Port Marking with Selective Blindness* (PMSB). If a flow is found to be a victim of per-port marking, we can either revoke the marking or cancel the flow back-off even if its packets qualify the per-port threshold (*i.e.*, selective blindness). By breaking the fixed causal relationship between port marking and back-off, flows from un-congested queues can be protected.

PMSB design: With the basic switch version of PMSB, a switch marks ECN only if two conditions both hold: (1) the port buffer length is larger than a per-port threshold (i.e., port marking), and (2) the queue buffer length is also no less than a per-queue filter threshold (i.e., selective blindness). The challenge is how to determine the per-queue threshold so that the best trade-off point of a selective blindness algorithm can be achieved. Following we present the key observations to this question, and leave design details to the next part (Section IV).

Per-port ECN deep dive: As mention above, per-port ECN marking violates the weighted fair sharing due to the interference between queues belonging to the same port. We change the port threshold to 65 packets, and repeat the simulation in Figure 3. Figure 6 shows that by increasing port threshold,

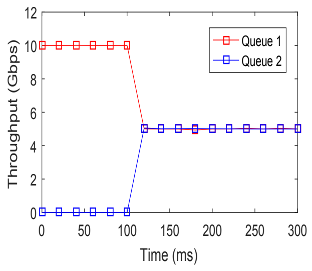

Fig. 6. 65Pkts & 1 flow : 8 flows

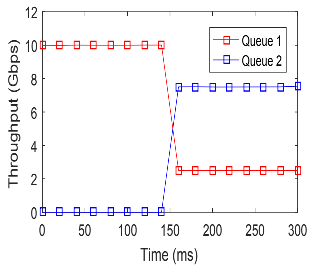

Fig. 7. 65Pkts & 1 flow : 40 flows

TABLE II
PARAMETERS USED IN OUR ALGORITHMS.

Parameter	Description
`port_length`	The buffer occupancy of port p.
`port_threshold`	The threshold of port p.
`queue_length_i`	The buffer occupancy of $queue_i$.
`queue_threshold_i`	The threshold of $queue_i$.
`weight_i`	The weight of $queue_i$.
`weight_sum`	The sum of all queues' weights in port p.
`is_mark`	A boolean variable indicates that if the packets will be marked or not.
`cur_rtt`	The current RTT.
`rtt_threshold`	The RTT threshold used to determine if the senders ignore ECN's congestion information or not.
`ignore_mark`	A boolean variable indicates that if the senders ignore ECN's congestion information or not.

even per-port marking can keep weighted fair sharing. This is because increasing port threshold can decrease the number of packets being marked. Without extensively marking signal, victim flows with a low ECN marking ratio would not back-off too much, and the scheduling policy may still get respected.

Note that increasing the port threshold cannot entirely solve this problem. If the crossing flow number increases, the stable buffer point would eventually exceed a fixed threshold, then flows get extensively marked. Figure 7 shows the result when the flow number reaches 40, where weighted sharing fair sharing is again violated. We cannot arbitrarily increase the port-level threshold, as increasing port threshold will also result in high latency. The observation is that *a selective blindness decision can be relatively aggressive to trade a small probability of false positive for the elimination of false negative*.

We also develop an heuristic end-host version PMSB(e) as an immediately-deployable alternative. With PMSB(e), a flow compares its current round-trip-time (RTT) with the base RTT and decides whether it should accept a marking ECN signal (Section V). Table I presents the comparison among existing approaches and PMSB.

IV. DETAILED DESIGN AND ANALYSES

In this section, we first give our design goals and introduce algorithm about PMSB, then we discuss the implementation, finally, we give the steady state analysis of PMSB.

Algorithm 1: PMSB

```
Input: $port\_length$, $port\_threshold$,
       $queue\_length\_i$, $weight\_i$, $weight\_sum$
Output: $is\_mark$
1 if ($port\_length < port\_threshold$) then
2     $is\_mark \leftarrow false$;
3     return $is\_mark$;
4 $queue\_threshold\_i \leftarrow \frac{weight\_i}{weight\_sum} \times port\_threshold$;
5 if ($queue\_length\_i \geq queue\_threshold\_i$) then
6     $is\_mark \leftarrow true$;
7     return $is\_mark$;
8 $is\_mark \leftarrow false$;
9 return $is\_mark$;
```

A. Design Goals

We highlight our design goals as following:

*1) **Weighted fair sharing:*** Obviously, in order to achieve weighted fair sharing, each queue requires to maintain an independent threshold that is proportional to the queue's weight.

*2) **High throughput and low latency:*** To achieve high throughput and low latency simultaneously, PMSB keeps a per-port threshold for each port and a per-queue filter threshold for each queue.

*3) **Deliver congestion information early:*** As TCN uses sojourn time to determine if the packets will be marked or not, it cannot indicate the congestion in advance. If the senders received congestion notification information earlier, they would reduce the degree of the network congestion. When marking ECN at dequeue time, PMSB can tell senders early about the network congestion.

*4) **Serviceable over generic packet schedulers:*** MQ-ECN only supports round-based schedulers such as WRR and D-WRR, which cannot be generalized to other schedulers. Unlike MQ-ECN, PMSB can support generic packet schedulers and does not rely on round-based schedulers.

B. Algorithm

Algorithm 1 shows pseudo-code for PMSB. And Table II describes the parameters used in algorithm 1. PMSB not only takes full advantage of per-port ECN, but also achieves weighted fair sharing that per-port ECN cannot achieve.

We use parameter $port\_threshold$ to capture the certain port's threshold shared by all queues, and use parameter $queue\_threshold\_i$ to represent the threshold of $queue_i$.

$$port\_threshold = C \times RTT \times \lambda \tag{5}$$

$$queue\_threshold\_i = \frac{weight\_i}{weight\_sum} \times port\_threshold \tag{6}$$

where the parameter $weight\_sum$ is the sum of all queues' weights in the same port, and $weight\_i$ indicates the weight of the ith queue that need to judge marking packets or not.

TABLE III
KEY NOTATIONS IN THIS PAPER.

n_i	The number of flows in ith queue.
V	The switch v.
c	The capacity of bottleneck link l.
p	The output port connected with bottleneck link l.
q	The number of queues in output port p.
w_i	The assigned weight of ith queue.
k_i	The marking threshold of ith queue.
$W(t)$	The window size of each flow at t.
$Q_i(t)$	The length of ith queue at t.
Q_i^{min}	The minimum queue length of ith queue.
Q_i^{max}	The maximum queue length of ith queue.
RTT	The round-trip time

When the size of packets buffered in one port is smaller than port's threshold, PMSB does not require to mark any packets, preserving high throughput and low latency. Once the buffered packets exceed port's threshold, PMSB begins to check that if the current queue's length beyond the queue's threshold or not. If these two conditions can be established at the same time (i.e., $port\_length$ is larger than $port\_threshold$ and $queue\_length\_i$ is larger than $queue\_threshold\_i$), the packets will be marked to indicate that congestion happens. Otherwise, they remain the same.

C. Discussion

PMSB is unaware of the concept of "round" and thus does not rely on the round-based schedulers. Besides, when marking operation at dequeuing time, PMSB can predict the congestion and deliver congestion notification in advance. Since PMSB will mark packets at dequeuing time once the length of port's buffer exceeds port's threshold and the length of queue's buffer exceeds queue's threshold. Hence, the senders can receive congestion information earlier. In the typical switch implementation for ECN/RED, there is a comparison operation between averaged queue length and a static threshold. PMSB can directly compare instantaneous or average queue length with threshold. MQ-ECN requires one additional moving average register to store T_{round} for each port. TCN is more complicate as it needs to perform unpacking and packing operations for each packet to get sojourn time. However, PMSB does not need additional registers, and keeps the same scale implementation complexity as ECN/RED.

D. Steady State Analysis

We analyze the lower bound of the threshold in order to avoid throughput loss. We first present our network model. We assume $\sum_{i=1}^{q} n_i$ long-lived flows with identical RTT, passing the bottleneck link of capacity c through port p, where the port p is associated with q queues and n_i synchronized flows share with ith queue. For convenience, we summarize important notations in Table III.

We first discuss the lower bound of threshold k_i. The length of ith queue at time t can be given by

$$Q_i(t) = n_i \cdot W(t) - \frac{w_i}{\sum_{j=1}^{q} w_j} \cdot C \cdot RTT \tag{7}$$

where $W(t)$ is the window size of each flow and w_i is the assigned weight for ith queue.

From the analysis in [1], we can conclude that the maximum length of ith queue is

$$Q_i^{max} = n_i(W^* + 1) - \gamma_i \cdot C \cdot RTT = k_i + n_i \tag{8}$$

where $W^* = \frac{\gamma_i \cdot C \cdot RTT + k_i}{n_i}$ and $\gamma_i = \frac{w_i}{\sum_{j=1}^{q} w_j}$. The notation W^* represents the window size for a single flow in the sender where the length of ith queue reaches k_i.

The amplitude of ith queue oscillations is

$$A_i = \frac{1}{2} \cdot \sqrt{2 \cdot n_i \cdot (\gamma_i \cdot C \cdot RTT + k_i)} \tag{9}$$

From equations (8) and (9), we can obtain

$$\begin{aligned} Q_i^{min} &= Q_i^{max} - A_i \\ &= k_i + n_i - \frac{1}{2} \cdot \sqrt{2 \cdot n_i \cdot (\gamma_i \cdot C \cdot RTT + k_i)} \end{aligned}$$

To determine the lower bound of Q_i^{min}, we calculate the derivative of the function $Q_i^{min}(n_i)$ as following.

$$\frac{\partial Q_i^{min}}{\partial n_i} = 1 - \frac{1}{2}\sqrt{\frac{\gamma_i \cdot C \cdot RTT + k_i}{2 \cdot n_i}} = 0$$

We can derive that the lower bound of Q_i^{min} can be established as following.

$$Q_i^{-} = \frac{7}{8} \cdot k_i - \frac{\gamma_i \cdot C \cdot RTT}{8} \tag{10}$$

when the variable n_i satisfies the below condition.

$$n_i = \frac{\gamma_i \cdot C \cdot RTT + k_i}{8} \tag{11}$$

To ensure that the queue i does not underflow and results in throughput loss, the lower bound Q_i^{-} must be greater than zero, i.e., $Q_i^{-} > 0$, which implies that

$$k_i > \frac{\gamma_i \cdot C \cdot RTT}{7} = \frac{w_i}{\sum_{j=1}^{q} w_j} \cdot \frac{C \cdot RTT}{7} \tag{12}$$

which means that the marking threshold of ith queue must satisfy the condition (12) such that the throughput loss can be avoided.

Based on the analysis above, we have the following theorem.

Theorem IV.1. *In order to avoid throughput loss, the threshold k_i of ith queue should satisfy the following condition.*

$$k_i > \frac{w_i}{\sum_{j=1}^{q} w_j} \cdot \frac{C \cdot RTT}{7}$$

V. An end-host heuristic version

In this section, we introduce another approach that we do not need to modify switches, and we name it PMSB(e). The goals of PMSB(e) include achieving high throughput, low latency, weighted fair sharing, deliver congestion notification early and support generic packet schedulers.

Algorithm 2: PMSB(e)

```
Input: $cur\_rtt$, $rtt\_threshold$, $is\_mark$
Output: $ignore\_mark$
1 if ($is\_mark == false$) then
2     $ignore\_mark \leftarrow true$;
3     return $ignore\_mark$;
4 if ($cur\_rtt < rtt\_threshold$) then
5     $ignore\_mark \leftarrow true$;
6     return $ignore\_mark$;
7 $ignore\_mark \leftarrow false$;
8 return $ignore\_mark$;
```

A. *Basic Idea*

As we mention that per-port ECN marking decision achieves high throughput and low latency, but it violates weighted fair sharing principle. As it cannot provide buffer isolation among the queues in a port, packets from one queue may be marked due to shared buffer occupied by other queues of the same port. To achieve weighted fair sharing, per-port ECN marking decision must avoid interference among different queues belonging to the same port.

PMSB(e) is based on per-port ECN, which combines ECN and RTT instead of only taking ECN as the congestion signal. Algorithm 2 shows the pseudo-code of PMSB(e), and the parameters used in PMSB(e) are shown in Table II. When the sender receives ACK with congestion information of ECN from the receiver, the sender may ignore this congestion signal if the RTT is small enough. Combining ECN and RTT, PMSB(e) can achieve weighted fair sharing.

B. *Discussion*

PMSB(e) has no concept of "round", and supports generic packet schedulers. According to the performance of per-port ECN marking, PMSB(e) can achieve high throughput, low latency. Due to the auxiliary judgment of RTT, PMSB(e) can provide isolation among queues belonging to the same port to achieve weighted fair sharing. In addition, through comparing the occupancy of port buffer with port threshold at dequeue time, PMSB(e) can also deliver congestion information early. The implement of PMSB(e) is uncomplicated, as it does not need to modify switches. PMSB(e) just needs to add a simple judge when the sender receives packets with congestion information of ECN. And it can coexist with other ECN-based transports like DCTCP. The main challenge is how to determine a time threshold that used to compare with the baseline RTT. However, the RTT in data center is stable, and recently work [10] has shown that RTT can be accurately measured with advance NIC hardware.

VI. Evaluation

In this section, we use ns-3 simulations to answer the following questions.

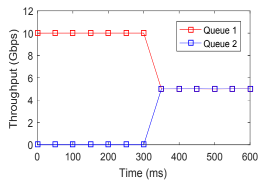

Fig. 8. DWRR &12Pkts & 1flow : 4flows

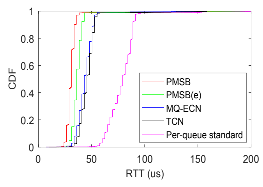

Fig. 9. RTT

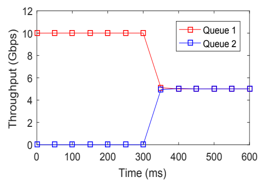

Fig. 10. DWRR & 12Pkts & 1flow : 100flows

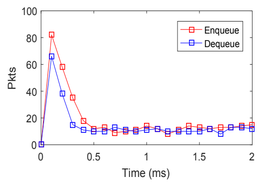

Fig. 11. PMSB

How does PMSB perform in large-scale datacenter? In the large-scale simulations, we measure the flow completion time to answer this question. Through extensive simulations, we found that PMSB can reduce 95th, 99th percentile and average completion time for small flows by up to 50.12%, 50.07% and 48.89% compared to TCN when using DWRR as scheduler. If we use WFQ as scheduler, the improvements are more significant: the results can reach up to 67.56%, 72.89% and 64.49% respectively.

Does PMSB rely on a specific scheduler? The answer is no. In static flow experiment, we test various schedulers to evaluate PMSB. The results show that PMSB can support Weighted Fair Queueing (WFQ), Deficit Round Robin (DRR) and Priority Queue(SP), and PMSB can achieve excellent performance.

Is it hard to determine the parameters for PMSB? Of course not, we only need to configure the port's threshold. Besides, Theorem IV.1 gives the lower bound of queue's threshold to avoid throughput loss. Accordingly, we can obtain the port's threshold by summing up the thresholds of all queues belonging to this port.

We use DCTCP to perform congestion control. Unless specifically mentioned, the bandwidth of all links are set to 10 Gbps. As MQ-ECN [5] suggests, we set parameter β to 0.75 and T_{idle} to the transmission time of a MTU. Note that the limitation of MQ-ECN and TCN: the former can only support round-based schedulers, and the later only marks packets at the dequeuing time.

A. Static Flow Experiment

We begin with some basic simulations to show that PMSB and PMSB(e) can achieve high throughput, low latency, weighted fair sharing and delivering congestion information early simultaneously.

1) Weighted Fair Sharing, High Throughput and Low Latency: At each switch, we configure DWRR including two queues with equal weights. The port thresholds for both PMSB and PMSB(e) are set to 12 packets, and the RTT threshold for PMSB(e) is set to 40μs. As for TCN, we set the sojourn time threshold to 39μs.

- **1:4** In this setting, queue 1 has one flow and queue 2 has four flows. Figure 8 shows the weighted fair sharing results achieved by PMSB. The throughput of queue 1 and queue 2 are both around 5 Gbps, which suggests that PMSB can strict preserve weighted fair sharing. Furthermore, the sum of throughput of these flows achieves 10Gbps that suggests that PMSB can fully utilize the link capacity. We omit the result of weighted fair sharing for PMSB(e) as its performance is quite similar to that of PMSB.

  We also measure the RTT of flows which classified into queue 2. Figure 9 shows the RTT distributions achieved by PMSB, PMSB(e), MQ-ECN, TCN and per-queue setting with standard threshold. Compared to the per-queue ECN with standard threshold, PMSB achieves 62.6% and 63.2% lower RTT in 99th percentile and average. And PMSB(e) achieves 55.5% and 55.8% lower RTT in 99th percentile and average, compared to the per-queue ECN with standard threshold. In summary, these results suggest that PMSB and PMSB(e) achieve low latency.
- **1:100** In this setting, we confirm that PMSB and PMSB(e) can also keep weighted fair sharing when the traffic becomes heavy. Queue 1 has one flow and queue 2 has one hundred flows. Figure 10 shows the result of PMSB, and PMSB(e) achieve the similar result with PMSB. Even though we change the ratio between two queues to 1:100, PMSB and PMSB(e) can also achieve weighted fair sharing and high throughput. Therefore, we can get the conclusion that PMSB and PMSB(e) achieve weighted fair sharing and high throughput even the traffic is heavy.

2) Deliver Congestion Information Early: We have confirm that TCN can not deliver congestion information early above, because packets must experience sojourn time before they are marked. However, PMSB and PMSB(e) can deliver congestion information early when marking at dequeue side. And we make another simulation to confirm that. We start 4 flows from 4 senders to the same receiver simultaneously and these flows are classified into the same queue. Then we measure the size of buffer occupancy of this queue. We set the port threshold for PMSB and PMSB(e) as 12 packets. And the RTT threshold for PMSB is set to 14.4μs.

Figure 11, 12 show the switch buffer occupancies versus time achieved by PMSB and PMSB(e) respectively. At the beginning, there are peak buffer occupancies. This is because TCP windows grow exponentially during the slow start phase before DCTCP takes effect. After these peaks, the size of buffer occupancies fluctuate near the port threshold due to the effect of DCTCP. When marking at enqueue side, the peak values of PMSB and PMSB(e) reach 82 packets, this is because congestion information need to experience a cor-

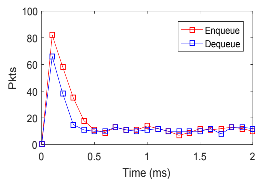

Fig. 12. PMSB(e)

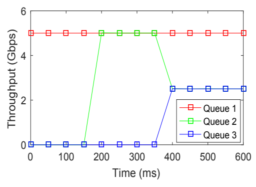

Fig. 13. SP+WFQ

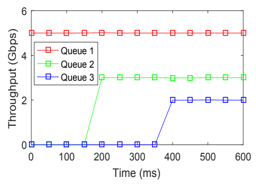

Fig. 14. SP

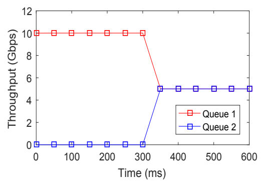

Fig. 15. WFQ

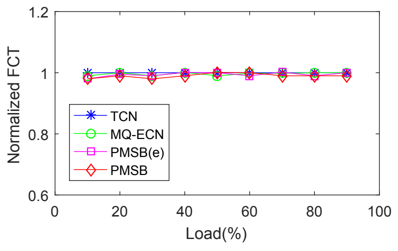

Fig. 16. Overall

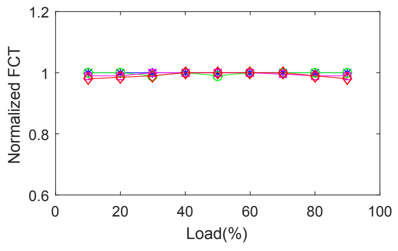

Fig. 17. (10MB,∞: 99th percentile)

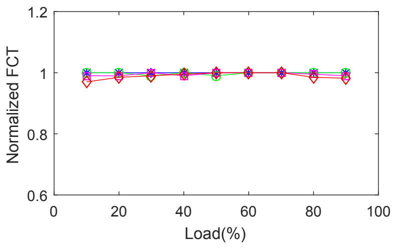

Fig. 18. (10MB,∞: Avg)

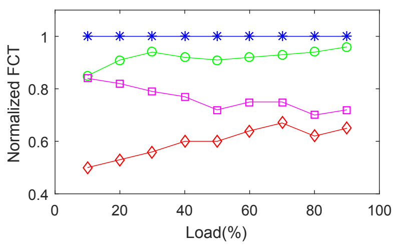

Fig. 19. (0,100KB]: 95th percentile)

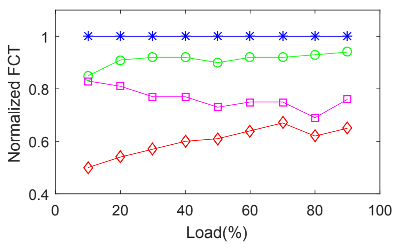

Fig. 20. (0,100KB]: 99th percentile)

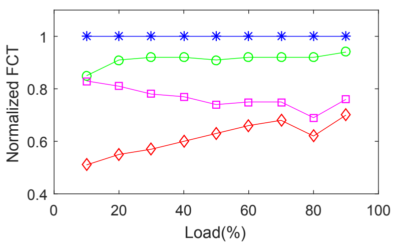

Fig. 21. (0,100KB]: Avg)

responding sojourn time before delivered to senders, thus the senders cannot react timely.

By contrast, when marking at dequeue side, the peak values of PMSB and PMSB(e) decrease about 20% compared to enqueue marking scheme. This is because senders can receive congestion information early, then decrease their rate to relieve the network congestion.

3) Support generic packet schedulers: We configure switch with SP+WFQ, SP, and WFQ respectively, then we evaluate the performance of PMSB and PMSB(e).

- **SP+WFQ:** At the switch, we configure SP+WFQ with three queues: queue 1 has a strict higher priority while queue 2 and queue 3 have equal weights in the lowest priority. First, we start a 5 Gbps TCP flow from sender 1, then start a flow from sender 2, finally, start 4 flows from sender 3. The flows from sender 1, 2 and 3 are classified to queue 1, 2, and 3 respectively. Based on the SP+WFQ scheduling policies, we can infer that the final throughput of queue 1, 2 and 3 should be 5Gbps, 2.5Gbps, 2.5 Gbps, respectively.

  Figure 13 shows the throughput of each queue versus time achieved by PMSB. During this experiment, queue 1 always keeps 5Gbps throughput, matching it's strict priority at the switch. When queue 3 is inactive, queue 2 also keeps 5Gbps throughput. After queue 3 becomes active, queue 2 and 3 keep the identical 2.5Gbps throughput. We omit the performance of PMSB(e) which is similar with that of PMSB. This suggests that PMSB and PMSB(e) can strictly preserve the scheduling policies when we combine SP and WFQ.
- **SP:** At the switch, we configure SP with three queues: queue 1 has the highest priority and queue 3 has the lowest priority, while the priority of queue 2 is between that of queue 1 and queue 3. First, we start a 5 Gbps TCP flow from sender 1, then start a 3 Gbps TCP flow from sender 2, start a 10 Gbps TCP flow from sender 3, the flows from sender 1, 2 and 3 are classified to queue 1, 2, and 3 respectively. Note that the bandwidth of links are 10Gbps. Based on the SP scheduling policies, we can infer that the final throughput of queue 1, 2 and 3 should be 5 Gbps, 3 Gbps, 2 Gbps, respectively.

  Figure 14 shows the throughput of each queue versus time achieved by PMSB. During this experiment, queue 1 always keeps 5 Gbps throughput, matching it's strict priority at the switch. When queue 2 is active, queue 1 also keeps 5 Gbps throughput, and queue 2 keeps 3 Gbps throughput. After queue 3 becomes active, queue 3 keeps 2 Gbps throughput, and queue 1, 2 remain unchanged. We omit the performance of PMSB(e) which is similar with that of PMSB. This suggests that PMSB and PMSB(e) can strict preserve the scheduling policies of SP while achieving good throughput.

- **WFQ:** At the switch, we configure WFQ with two queues: queue 1, 2 has the equal weights. First, we start one TCP flow from sender 1, then start four TCP flows from sender 2, the flows from sender 1, 2 are classified to queue 1, 2 respectively. Based on the WFQ scheduling policies, we can infer that the final throughput of queue 1, 2 should be equal to 5 Gbps.

  Figure 15 shows the throughput of each queue versus time achieved by PMSB. When queue 2 is inactive, queue 1 keeps 10 Gbps throughput. When queue 2 becomes active, queue 1 and 2 keep the identical 5 Gbps throughput. We omit the performance of PMSB(e) which is similar with that of PMSB. This suggests that PMSB and PMSB(e) can strictly preserve the scheduling policies of WFQ while achieving good throughput.

So we get the conclusion that PMSB and PMSB(e) can support generic packet schedulers.

B. *Large-scale NS-3 Simulations*

In this section, we use ns-3 simulations to evaluate the performance of PMSB and PMSB(e) in large-scale datacenter networks. We use Deficit Weighted Round Robin (DWRR) and Weighted Fair Queueing (WFQ) as the schedulers of switches respectively. Because MQ-ECN cannot support WFQ, we just compare PMSB, PMSB(e) and TCN when using WFQ to schedule queues.

1)DWRR Scheduler

Topology: We use 48-host leaf-spine topology with 4 leaf (ToR) switches and 4 spine (Core) switches. Each leaf switch has 12 10Gbps downlinks to hosts and 4 10Gbps uplinks to spines, forming a non-blocking network. We employ ECMP for load balancing.

Workloads: We use realistic workload for all simulations. Since there are 48 $\times$ 47 communications in total. And we classify those communications into 8 services evenly. We generate 48flows in the simulations and the traffic pattern is based on Poisson process. Among these flows, the small flows account for 60%, and the large flows account for 10%.

Transport: We use DCTCP as congestion control protocol and the initial window size is set to 16 packets.

Switch: According to the Theorem IV.1, we set the port threshold of PMSB and PMSB(e) as 12 packets. And the RTT threshold for PMSB(e) is $85.2\mu s$. We set standard threshold as 65 packets for MQ-ECN according to [5], and we set TCN threshold as 78.2 μs according to [3]. All queues have the equal weights. For PMSB, PMSB(e) and MQ-ECN, switches mark packets at enqueue time.

Next, we give the FCT results across different flow size. Figure 16∼21 show the results, and we omit the result of the medium flows (100KB, 10MB) whose performance trend is very similar to that of overall average FCT.

Overall: Figure 16 shows the results of overall average FCT. All of these schemes achieve the similar results. This is because most of the bytes are from a small number of large flows that are throughput-intensive. Therefore, these large flows determine the overall average FCT. Since the link can be utilized fully by all schemes, the overall average FCT are similar.

Although these schemes achieve similar overall average FCT generally, they also exist difference at low network load. PMSB, and PMSB(e) can reduce overall average FCT by up to 2% at low loads.

Large Flows: Figure 17, 18 show that PMSB and PMSB(e) can also keep good performance for large flows. At low and high load, PMSB and PMSB(e) can reduce about 1%∼2% FCT for large flows.

Small Flows: As for small flows, figure 19∼21 show that PMSB and PMSB(e) achieve better performance than TCN at any loads. Compared to TCN, PMSB can reduce 95th percentile FCT for small flows up to 52% at low load and reduce more than about 35% 95th percentile FCT at any other loads. PMSB(e) can reduce up to 30% 95th percentile FCT at high loads, and reduce more then 20% 95th percentile FCT at most loads. Besides, we can get the similar improvement for 99th percentile FCT and average FCT.

When compared to MQ-ECN, PMSB can reduce about 40% 95th percentile FCT at low loads, and reduce more than 30% 95th percentile FCT at most loads. PMSB(e) can reduce about 25% 95th percentile FCT at high loads compared to MQ-ECN. Besides PMSB, PMSB(e) can achieve similar good results for 99th percentile FCT and 100th percentile FCT.

2)WFQ Scheduler

Here we just replace DWRR with WFQ, and other settings remain unchanged. Figure 22∼27 show the results of FCT across different flow size, and we omit the result of the medium flows (100KB, 10MB) whose performance trend is very similar to that of overall average FCT. MQ-ECN is excluded, because it only support round-based schedulers. For PMSB, PMSB(e) and TCN, switches mark packets at enqueue time.

Overall: Figure 22 shows PMSB and PMSB(e) both achieve good performance. Compared to TCN, PMSB and PMSB(e) just increase overall average FCT within 2% .

Large Flows: About large flows, figure 23, 24 show that PMSB and PMSB(e) achieve similar performance with TCN. Compared to TCN, PMSB(e) reduce about 2% FCT for large flows at low loads. And the difference of FCT between PMSB and TCN within 2%.

Small Flows: For small flows, figure 25∼27 show that PMSB and PMSB(e) achieve excellent FCT. Compared to TCN, PMSB can reduce up to 68% 95th percentile FCT at low load and reduce exceed 40% 95th percentile FCT at high loads. PMSB(e) can reduce up to 23% 95th percentile FCT at most loads. In addition, PMSB and PMSB(e) can also achieve the similar improvement for 99th percentile FCT and overall average FCT.

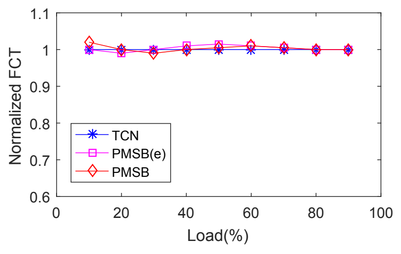

Fig. 22. Overall

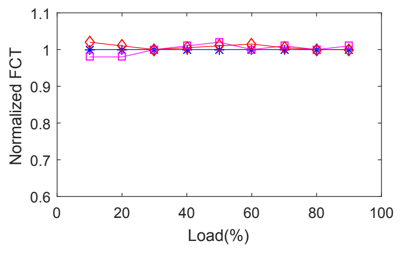

Fig. 23. (10MB,∞: 99th percentile)

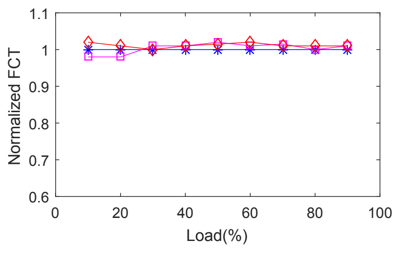

Fig. 24. (10MB,∞: Avg)

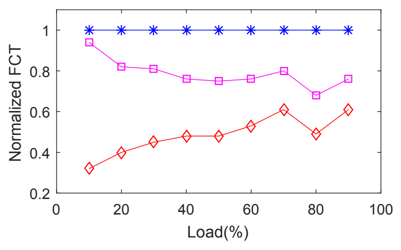

Fig. 25. (0,100KB]: 95th percentile)

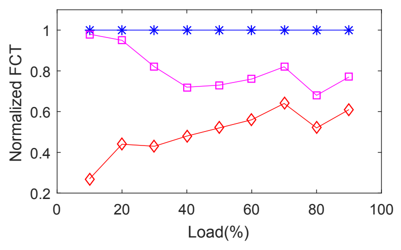

Fig. 26. (0,100KB]: 99th percentile)

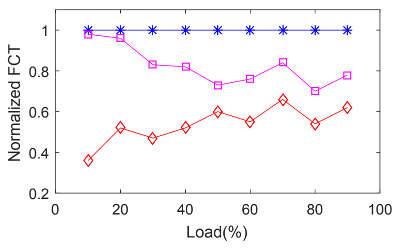

Fig. 27. (0,100KB]: Avg)

VII. Conclustion

In this paper, we present *per-Port Marking with Selective Blindness* (PMSB) that can achieve high throughput, low latency and weighted fair sharing simultaneously. Evaluation results show that PMSB can reduce the flow completion time for small flows while delivering a slightly better performance for large flows.

Acknowledgment

The authors would like to thank reviewers for their valuable comments. This work was supported in part by the National Science and Technology Major Project of China under Grant Number 2017ZX03001013-003, the Fundamental Research Funds for the Central Universities under Grant Number 0202-14380037, the National Natural Science Foundation of China under Grant Numbers 61772265, 61602194, 61502229, 61672276, and 61321491, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *in Proc. ACM SIGCOMM 2011*, volume 41, pages 63–74. ACM, 2011.

[2] M. Alizadeh, A. Javanmard, and B. Prabhakar. Analysis of dctcp: stability, convergence, and fairness. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 73–84. ACM, 2011.

[3] W. Bai, K. Chen, L. Chen, C. Kim, and H. Wu. Enabling ecn over generic packet scheduling. In *CoNEXT*, pages 191–204, 2016.

[4] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. Information-agnostic flow scheduling for commodity data centers. In *NSDI*. USENIX, 2015.

[5] W. Bai, L. Chen, K. Chen, and H. Wu. Enabling ecn in multi-service multi-queue data centers. In *NSDI*, pages 537–549, 2016.

[6] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking (ToN)*, 1(4):397–413, 1993.

[7] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can jump them! In *NSDI*. USENIX, 2015.

[8] K. Jang, J. Sherry, H. Ballani, and T. Moncaster. Silo: Predictable message latency in the cloud. *ACM SIGCOMM Computer Communication Review*, 45(4):435–448, 2015.

[9] G. Judd. Attaining the promise and avoiding the pitfalls of tcp in the datacenter. In *USENIX NSDI 2015*, pages 145–157, 2015.

[10] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats, et al. Timely: Rtt-based congestion control for the datacenter. In *Proc. ACM SIGDC 2015*, pages 537–550. ACM, 2015.

[11] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar. Friends, not foes: synthesizing existing transport strategies for data center networks. In *ACM SIGCOMM 2014*, pages 491–502. ACM, 2014.

[12] 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 *in Proc. IEEE INFOCOM, 2013*, pages 2157–2165. IEEE, 2013.

[13] D. Shan, W. Jiang, and F. Ren. Absorbing micro-burst traffic by enhancing dynamic threshold policy of data center switches. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 118–126. IEEE, 2015.

[14] D. Shan and F. Ren. Improving ecn marking scheme with micro-burst traffic in data center networks. In *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2017.

[15] 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 *Proc. ACM SIGDC 2015*, pages 183–197. ACM, 2015.

[16] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *ACM SIGCOMM*, pages 115–126. ACM, 2012.

[17] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang. Tuning ecn for data center networks. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 25–36. ACM, 2012.

[18] 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 *ACM SIGCOMM Computer Communication Review*, volume 45, pages 523–536. ACM, 2015.