Enabling ECN in Multi-Service Multi-Queue Data Centers

Wei Bai, Li Chen, and Kai Chen, ***Hong Kong University of Science and Technology;***
Haitao Wu, ***Microsoft***

https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/bai

This paper is included in the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16).

March 16–18, 2016 • Santa Clara, CA, USA

ISBN 978-1-931971-29-4

Open access to the Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16) is sponsored by USENIX.

Enabling ECN in Multi-Service Multi-Queue Data Centers

Wei Bai[1], Li Chen[1], Kai Chen[1], Haitao Wu[2]
[1]*SING Group @ HKUST* [2]*Microsoft*

Abstract

Recent proposals have leveraged Explicit Congestion Notification (ECN) to achieve high throughput low latency data center network (DCN) transport. However, most of them implicitly assume each switch port has one queue, making the ECN schemes they designed inapplicable to production DCNs where multiple service queues per port are employed to isolate different traffic classes through weighted fair sharing.

In this paper, we reveal this problem by leveraging extensive testbed experiments to explore the intrinsic tradeoffs between throughput, latency, and weighted fair sharing in multi-queue scenarios. Using the guideline learned from the exploration, we design MQ-ECN, a simple yet effective solution to enable ECN for multi-service multi-queue production DCNs. Through a series of testbed experiments and large-scale simulations, we show that MQ-ECN breaks the tradeoffs by delivering both high throughput and low latency simultaneously, while still preserving weighted fair sharing.

1 Introduction

Data centers host a variety of applications and services with diverse network requirements: some services (*e.g.*, monitoring services) demand low latency for sporadic short messages; some (*e.g.*, data-parallel computation [13]) require high throughput for large flows; while others (*e.g.*, online data-intensive applications) desire both high throughput and low latency for a large number of concurrent flows [18].

To meet these requirements, ECN has been employed as a powerful tool by recent DCN transport proposals such as [6, 8, 29, 31, 32], and they show that a properly tuned ECN marking scheme can deliver high throughput and low latency simultaneously [31]. Due to their simplicity and effectiveness, ECN-based transports such as DCTCP [6] and DCQCN [32] are widely used in industry—DCTCP has been integrated into various OS kernels [3, 4] and deployed in DCNs of Google [27] and Morgan Stanley [19]; while DCQCN has been deployed in DCNs of Microsoft [32] to enable RDMA.

A further look at these proposals reveals that their ECN marking schemes are mostly designed based on the implicit assumption that each switch port only has one queue. However, the industry trend in production DCNs is going beyond such one queue per port paradigm [8, 9]. Today's commodity switches already support 4–8 classes of service queues per port [9, 10, 20]. Current operation practice is to leverage queues to segregate traffic from different services and enforcing weighted fair sharing among different queues [8, 9, 19]. For example, operators assign a higher weight to all traffic belonging to a more important *real-time* search application over a *background* backup application, thus providing differentiated network performance. A key question in such single-queue to multi-queue transition is the applicability of ECN, which remains unexplored.

We point out, via extensive testbed experiments, that the prior ECN schemes developed for the single queue model fall short when directly migrated to the multi-queue scenarios (§2). There exist fundamental tradeoffs between high throughput, low latency, and weighted fair sharing. Our experiments demonstrate: 1) applying per-queue ECN with the standard marking threshold derived before on each queue independently ensures high throughput, but can incur high latency when many queues are active; while apportioning this threshold among all the queues statically according to their weights guarantee low latency, but can degrade throughput when few queues are live; 2) applying per-port ECN with such standard threshold can maintain both high throughput and low latency, but violating weighted fair sharing across different queues.

Motivated by above problem, we seek a solution that can break the tradeoffs and enable ECN for multi-service multi-queue DCNs. To this end, we present MQ-ECN, a simple yet effective solution that achieves our goal (§3). First, MQ-ECN takes the per-queue ECN approach to preserve weighted fair sharing. Then, at its heart, MQ-ECN adjusts the ECN marking threshold for each queue based on its dynamic weighted fair share rate, rather than sticking to its static fair share weight, which enables MQ-ECN to well adapt to traffic variations while maintaining both high throughput and low latency in a highly dynamic DCN environment.

We explain that MQ-ECN is feasible to implement on

existing commodity switch hardware as MQ-ECN just requires one additional moving average register per port compared to the standard ECN/RED switch implementation (§4.1). We also present a MQ-ECN software implementation for testbed evaluation (§4.2). In our software prototype, MQ-ECN is implemented as a Linux `qdisc` kernel module running on a multi-NIC server to emulate switch behaviors.

We build a small-scale testbed with 9 Dell servers connected to a 9-port server-emulated MQ-ECN-enabled switch. We evaluate the basic properties of MQ-ECN on the testbed using realistic workloads [6, 16, 25]. Our experiments demonstrate that MQ-ECN achieves both high throughput and low latency simultaneously, while strictly preserving weighted fair sharing. For example, compared to per-queue ECN with the standard threshold, MQ-ECN achieves up to 72.8% lower 99th percentile FCT for small flows while delivering similar performance (*e.g.*, within 2.7%) for large flows (§5.1).

To complement small-scale testbed experiments, we conduct large-scale ns-2 [5] simulations to deep-dive into MQ-ECN. Our simulation results further confirm the superior performance of MQ-ECN. For example, compared to per-queue ECN with the standard threshold, MQ-ECN reduces the 99th percentile FCT for small flows by up to 43.7%. In addition, MQ-ECN achieves up to 13.2% lower average FCT for large flows compared to per-queue ECN with the minimum threshold (§5.2). Finally, we show, through a series of targeted simulations, that MQ-ECN is robust to different network environments and parameter settings, such as the number of queues, queue weights, transport protocols, and so on (§5.3).

To make our work easy to reproduce, we make our codes available online at: `http://sing.cse.ust.hk/projects/MQ-ECN`.

2 Problem Exploration

In this section, we begin by introducing the ECN mechanisms supported by existing commodity switching chips. Then, we explore the problems and tradeoffs of applying ECN in multi-service multi-queue DCNs. Finally, we summarize our design goals.

2.1 ECN on Commodity Switching Chips

Today's commodity switching chips provide multiple ECN/RED configuration options. For example, in our testbed, the Broadcom BCM-56538 chip supports per-queue, per-port, and per service pool ECN markings. For all schemes, the marking decision is made when a packet is enqueued (required by RED [15]). The main difference among them is that they use buffer occupancy in different egress entities to make marking decisions.

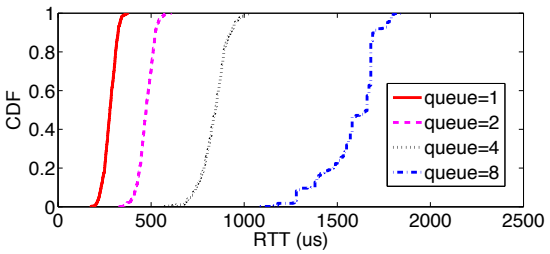


Figure 1: [Testbed] RTT: under per-queue ECN with the standard threshold, more queues lead to worse latency.

Briefly, in per-queue ECN marking, each queue has its own threshold and performs ECN marking independently to other queues. In per-port ECN marking, each port is assigned a single marking threshold. When the sum of queue buffer occupancy belonging to the same port is larger than the marking threshold, packets will get ECN marking. In per service pool ECN marking, packets are marked when total buffer occupancy in a shared buffer pool exceeds the marking threshold.

2.2 Problems and Tradeoffs

Before exploring the problems, we first introduce the *standard* ECN marking threshold derived by prior works [7, 31] based on the single queue model. Consider synchronized flows with identical round-trip times sharing the only queue of a bottleneck link, according to [7, 31], to fully utilize the link bandwidth while achieving low latency, the ECN marking threshold K[1] should be set as follows:

$$K = C \times RTT \times \lambda \quad (1)$$

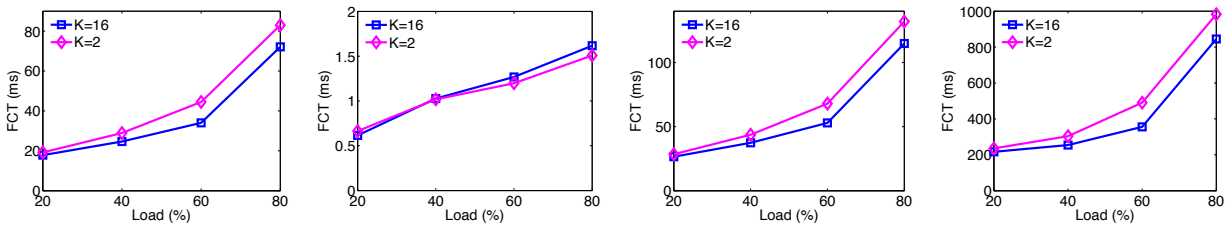
where RTT is average round-trip time, C is link capacity, and λ is a tunable parameter closely related to congestion control algorithms[2]. In production DCNs, round-trip times are relatively stable and operators can estimate RTT through large-scale measurements to compute the standard ECN marking threshold [17, 31].

2.2.1 Per-queue ECN with the standard threshold

In multi-queue environment, per-queue ECN marking is widely employed by operators for its good isolation among different queues. However, how to set the ECN marking threshold for each queue is a challenge. DCN traffic is well known for its volatility and burstiness [11, 16]. Thus, to achieve high utilization in any

[1] We are aware that ECN/RED has two (low and high) thresholds. Many ECN-based transports [6, 29, 31] set them to the same value. Without loss of generality, we also assume that the low and high thresholds are set to the same value to simplify analysis.

[2] For example, $\lambda = 1$ for regular ECN-enabled TCP which simply cuts window by half in the presence of ECN [31].



(a) Overall (b) (0,100KB] (c) (100KB,10MB] (d) (10MB,∞)

Figure 2: [Testbed] Average FCT statistics: under per-queue ECN with minimum threshold (*e.g.*, K=2), it suffers from degraded throughput as only one queue is active, thus leading to higher FCT overall.

condition, some network operators have configured the standard marking threshold, *e.g.*, $C \times RTT \times \lambda$, for each queue. Under this configuration, any queue can fully utilize link capacity independently. However, the problem is that when N queues are busy simultaneously, the total buffer occupancy can easily reach N times the standard threshold, thus introducing high queueing delay and huge buffer pressure[3].

To quantify such impairment, we build a small testbed consisting of 15 servers connected to a Pica8 P-3295 GbE switch. DCTCP is enabled on all the servers[4]. We configure Deficit Weighted Round Robin [26] with equal quantum per queue on the switch. We set the per-queue ECN marking threshold to 16 packets. We generate 8 long-lived flows using `iperf` from 8 servers to the same receiver. We vary the number of queues from 1 to 8 and evenly classify all the flows into these queues by setting different Differentiated Services Code Point (DSCP) values. Given all the queues should have similar queueing delay, we run `ping` in an active queue to measure RTT. Figure 1 shows RTT distributions. Clearly, more queues lead to worse latency. Compared to the single queue, the average and 99th percentile RTTs achieved by 8 queues degrade by 5.7X (279μs to 1582μs) and 4.9X (375μs to 1850μs), respectively.

Observation 1: *Per-queue ECN with the standard threshold suffers from poor latency when many queues are concurrently active.*

2.2.2 Per-queue ECN with the minimum threshold

To address the defect above, a natural way is to apportion the standard threshold among all the queues according to their fair share weights. Assume each port has N queues in total and the weight of queue i is W_i, then the *minimum* threshold for queue i, K_i, can be set as:

$$K_i = \frac{W_i}{\sum_{j=1}^{N} W_j} \times C \times RTT \times \lambda \quad (2)$$

Note that $\frac{W_i}{\sum_{j=1}^{N} W_j}$ is the normalized weight for queue i and $\frac{W_i}{\sum_{j=1}^{N} W_j} \times C$ is the minimum guaranteed rate for this queue. Hence, the minimum threshold K_i ensures that each queue can receive its minimum guaranteed bandwidth. Since K_i is proportional to W_i, it can also preserve weighted fair sharing among different queues. Furthermore, given that $\sum_{j=1}^{N} K_j = C \times RTT \times \lambda$, such configuration can achieve good latency and burst tolerance regardless of the total number of queues.

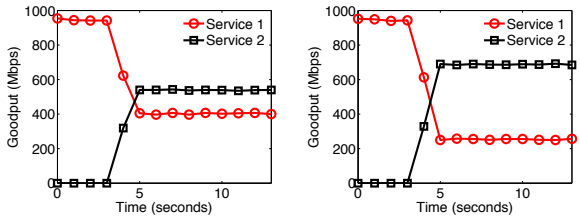
However, the problem of this method is that it can seriously degrade link utilization, especially when only few queues are active. The reason is that the bandwidth for inactive queues cannot be fully utilized by active queues as they are throttled by the statically-configured minimum ECN marking thresholds. The low throughput directly degrades the flow completion times (FCT).

To quantify this impact, we develop a client/server application to generate traffic according to the web search workload [6]. The client instance, running on one server, periodically generates requests to server instances, running on the other 14 machines, to fetch data. All the traffic is classified into the same switch queue. Since only one queue is active, to fully utilize link capacity, we should assign the standard threshold, *e.g.*, 16 packets, for this queue. Given we have 8 queues with equal weights, the corresponding minimum threshold for one queue is 2 packets. Thus, we evaluate the performance of both 16 packets and 2 packets in the experiment. Figure 2 shows the FCT results across different flow size regions. It turns out that the scheme with the threshold of 16 packets achieves $7.2 - 23.5\%$ lower overall average FCT (due to higher throughput) compared to that with the minimum threshold of 2 packets. This performance improvement stems mainly from the flows larger than 100KB.

Observation 2: *Per-queue ECN with the minimum threshold cannot maintain high throughput especially when few queues are concurrently active.*

[3]Taking Pica8 P-3922 10GbE switch [1] as an example, it has 9MB buffer shared by 384 queues (48 ports$\times$8 queues/port). DCTCP [6] recommends using at least 65 packets as the threshold for 10G networks. Hence, when 97 queues are busy simultaneously, the buffer is likely to be overfilled. Frequent packet drops can also severely degrade latency.

[4]By default, we choose DCTCP as the transport protocol for all experiments/simulations in this paper except special declaration.



(a) 1 flow : 4 flows (b) 1 flow : 8 flows

Figure 3: [Testbed] Aggregate goodput statistics: in per-port ECN, a queue with more flows grabs more bandwidth.

2.2.3 Per-port and per service pool ECN

Unlike above two approaches, per-port ECN can achieve both high throughput and low latency with the standard threshold. However, the problem is that it cannot ensure isolation among different queues of the same port [10]. This is because packets from one queue may get marked due to buffer occupancy of the other queues belonging to the same port. This undesirable interaction can severely violate weighted fair sharing among queues. We believe the problem will deteriorate under per service pool ECN marking because in such case queues belonging to even different ports may interfere with each other.

To understand this impairment, we start several long-lived TCP flows from two senders to the same receiver. We classify traffic into two services based on their senders. On the switch, both services have a equal-quantum dedicated queue. The per-port marking threshold is 16 packets. We vary the numbers of flows and measure aggregate goodputs for two services. Ideally, both services should always equally share the link capacity. Figure 3 shows the actual share results. When service 1 has 1 flows and service 2 has 4 flows, their aggregate goodputs are 403Mbps and 539Mbps, respectively. When the number of flows in service 2 is increased to 8, its aggregate goodput further reaches 688Mbps. This suggests that under per-port ECN marking, packets of service queue 1 get over-marked due to the aggressiveness of packets in service queue 2, thus making service 1 fail to achieve its weighted fair share rate.

Observation 3: *Per port and per service pool ECN can violate weighted fair sharing among different queues.*

2.3 Design Goals

Motivated by the above problems, we aim to design an ECN marking scheme for multi-service multi-queue production DCNs with the following properties:

- **High throughput:** Our scheme must be work-conserving. Active services should be able to fully utilize the network bandwidth as long as they have enough demands.
- **Low latency:** Our scheme should maintain low buffer occupancy in order to provide low queueing delay and good burst tolerance.
- **Weighted fair sharing:** Our solution should strictly preserve the weighted fair sharing policy among different service queues at any time.
- **Compatible with legacy ECN/RED implementation:** Although there are a few ECN improvements that leverage dequeue marking [31], to the best of our knowledge, few chip providers have offered the support. Therefore, we choose to design a scheme that can benefit from most ECN features that are available on existing switching commodity chips.

We show how MQ-ECN achieves the first three goals in the next section. To achieve the last goal, we require MQ-ECN to perform RED-like enqueue marking, *e.g.*, comparing the average queue length against a threshold at the enqueue side. And we discuss our implementation requirements in §4.1.

3 The MQ-ECN Design

3.1 Design Guideline

The above problem exploration has guided our design of MQ-ECN. The lesson we learned is two-fold:

- To avoid interference among different queues and preserve weighted fair sharing, the ECN marking should be performed on a per queue basis while complying with the weights across different queues.
- To achieve both high throughput and low latency simultaneously, we should not set static ECN marking thresholds for queues—applying the standard threshold on each queue independently can cause high latency (observation 1), while apportioning this threshold among all queues statically according to their weights can lead to low throughput (observation 2). Instead, the ECN marking threshold for each queue should adapt to traffic dynamics, and it should be set in a way that can *barely* maintain its weighted fair share rate while not introducing extra queueing delay. More specifically, for each queue, if its input rate is larger than its weighted fair share rate, we should use ECN to properly throttle it for latency; otherwise, we should not impose any constraint in order not to affect its throughput. As a result, the core of MQ-ECN is to derive such a proper ECN marking threshold for each queue according to its weighted fair share rate.

In our implementation, we find that the ECN threshold setting is closely related to the underlying packet scheduler that enforce the weighted fair sharing. Thus, in the following, we first describe the base design of MQ-ECN with ideal Generalized Processor Sharing (GPS) [23].

Then we discuss how to extend the base design to practical packet scheduling algorithms that are widely implemented in existing commodity switching chips.

3.2 MQ-ECN with Ideal GPS

3.2.1 Base Model

We consider a switch output link with the capacity C. The switch uses GPS as the underlying scheduler. There are N flows in total and the demand of flow i is r_i. Flow i is mapped to queue i with the weight of w_i[5]. The total demand is A, and $A = \sum_{i=1}^{N} r_i$. Let α denote the fair share rate and $w_i\alpha$ is the corresponding weighted fair share rate for flow i. According to [28], if $A > C$, the link is congested and α is the unique solution for equation $C = \sum_{i=1}^{N} min(r_i, w_i\alpha)$; if $A \leq C$, then there is no congestion and $\alpha = max\{\frac{r_i}{w_i}\}$. The output rate of queue i is given by $min(r_i, w_i\alpha)$.

According to design guideline in §3.1, we classify the queues into two categories based on the relations between their input rates and weighted fair share rates. For queues whose $r_i > w_i\alpha$, we use the following Equation (3) to throttle their input rates to keep queue occupancy and maintain low latency. For queues whose $r_i \leq w_i\alpha$, they should not be constrained.

$$K_i = w_i\alpha \times RTT \times \lambda \quad (3)$$

However, in order to enforce the scheme, the premise is to identify the queues whose $r_i > w_i\alpha$ and estimate their weighted fair share rates (*e.g.*, output rates) $w_i\alpha$, which was a challenge. Some previous works [22, 28] first estimate the input rates r_i and then calculate the weighted fair share rates $w_i\alpha$ using various complicated heuristics in the context of FIFO scheduling. Accurate rate estimation is challenging in data centers as traffic is volatile and bursty. Unlike previous approaches, we use GPS as the underlying scheduler. Thus, we can take advantage of the special properties of GPS to address this challenge in a much simpler way.

Note that GPS serves all backlogged queues in a bit-by-bit round-robin fashion. Assume $quantum_i = w_i$ bits is the quantum for queue i in a round, and T_{round} is the total time to serve all queues once. In case $r_i > w_i\alpha$ for queue i, the data in the queue will keep growing and eventually use up its $quantum_i$ in a round, then we can use $\frac{quantum_i}{T_{round}}$ to calculate the output rate of queue i, which is $w_i\alpha$. Thus, Equation (3) can be translated to:

$$K_i = \frac{quantum_i}{T_{round}} \times RTT \times \lambda \quad (4)$$

where $quantum_i$, RTT and λ are known while T_{round} can be well estimated through continuous sampling as we show later in §3.3.

[5]Given each queue only has one flow, we use 'flow' and 'queue' interchangeably in §3.

Interestingly, though intended for queues whose $r_i > w_i\alpha$, we find that Equation (4) can also be applied to queues whose $r_i \leq w_i\alpha$ with no harm. Here is the reason. For a queue i whose $r_i \leq w_i\alpha$, the data drained in a round is no more than $quantum_i$, this means that we can use $\frac{quantum_i}{T_{round}}$ to safely cap the output rate of queue i, which is r_i. Thus applying Equation (4) to queue i does not throttle its input rate, but still allows it to grow beyond its weighted fare rate before taking effect. This greatly simplifies our design because, we can apply Equation (4) to every queue with no differentiation, without the need of explicitly identifying the relations between their input rates and weighted fair share rates. As a result, Equation (4) establishes the ECN marking scheme of MQ-ECN with the ideal GPS scheduler.

3.2.2 Why it works?

We find Equation (4) well achieves our design goals in §2.3. First, $quantum_i$ ensures that different queues have thresholds in proportion to their weights, thus preserving the weighted fair sharing. Second, T_{round} reflects traffic dynamics of the link and automatically balances its throughput and latency. When there are more queues whose input rates exceed their weighted fair share rates, T_{round} tends to become larger, then K_i automatically becomes smaller to maintain low latency. When there are fewer queues reach their weighted fair share rates, T_{round} becomes smaller, then K_i automatically becomes larger to maintain high throughput.

Furthermore, in practice, T_{round} may change drastically because data center traffic is volatile and bursty. Accurately estimating T_{round} is challenging and deviation is unavoidable. However, we find that MQ-ECN can be self-healing:

- Assume that T_{round} is over-estimated initially, we get a smaller K_i which degrades throughput. Then more and more queues will be over-throttled by MQ-ECN and cannot achieve their weighted fair share rates. As a consequence, T_{round} becomes smaller.
- Assume that T_{round} is under-estimated initially, we get a larger K_i which increases latency. Then more and more queues will ramp up and exceed their weighted fair share rates. As a consequence, T_{round} becomes larger.

Therefore, an inaccurate initial estimation for T_{round} can be cured by itself eventually in the later stages. Furthermore, in our implementation, to prevent any temporary impact of under-estimation, we use Equation (5) below to bound it, considering that the weighted fair share rate should never be larger than the link capacity. Our evaluation results in §5 further confirm that MQ-ECN works well in practice.

$$K_i = min(\frac{quantum_i}{T_{round}}, C) \times RTT \times \lambda \quad (5)$$

3.3 MQ-ECN with Practical Packet Schedulers

In this section, we show how to extend the solution derived from ideal GPS to practical packet scheduling algorithms that try to approximate GPS. These schemes can be generally divided into two classes: fair queueing and round robin. Fair queueing schemes, such as Weighted Fair Queueing [14], achieve good fairness in general, but they are expensive to implement due to high $O(log(n))$ time complexity (n is the number of queues). Round robin schemes [21, 26] suffer from short-term burstiness and unfairness, but they are widely implemented in commodity switching chips for their $O(1)$ time complexity. For example, some dominant chipsets such as Broadcom Trident-I&II [2] adopted in many production data centers only support several round-robin schemes, such as Weighted Round Robin (WRR) and Deficit Weighted Round Robin (DWRR). Hence, we mainly focus on round robin schemes in this paper.

To apply Equation (5) for a packet scheduling algorithm, we need to obtain average round time T_{round} and per-queue quantum $quantum_i$ in the new context of this algorithm. Here, we show how to get T_{round} and $quantum_i$ for two popular round-robin schemes: WRR and DWRR. We envision that similar approaches can be extended to other round-robin schemes.

3.3.1 Estimate T_{round}

Round-robin packet scheduling algorithms usually serve queues in a circular order. Intuitively, one can obtain a sample of T_{round} whenever the scheduler finishes serving all the queues in a round. However, the sample frequency of this approach is directly affected by the total number of active queues. When many queues are concurrently active, it cannot track traffic dynamics efficiently. Hence, we propose to sample T_{round} whenever a queue just finishes its service in a round. A benefit is that such sampling frequency is independent to the number of queues. We assume that each queue i maintains a variable T_{pre} to store the time stamp when queue i finishes the service in previous round. Every time queue i finishes its service, it records the current time stamp T_{now} and calculates a round time sample T_{sample} as: $T_{sample} = T_{now} - T_{pre}$. Then we reset T_{pre} with T_{now}. Every time we get a sample, we smooth T_{round} using exponential filter as follows:

$$T_{round} = \beta \times T_{round} + (1 - \beta) \times T_{sample} \quad (6)$$

where β is a parameter in $(0, 1)$. We note that the above approach may have two potential limitations. First, T_{round} will be updated too frequently when there are many empty queues. Second, sampling stalls when the link is idle. To address the first limitation, we only sample T_{round} on active queues. If queue i is empty, we just reset T_{pre} with T_{now} and move forward to next queue. To address the second limitation, we simply set T_{round} as $\beta \times T_{round}$ (as if we get a T_{sample} of 0) when the switch port is idle for a pre-defined T_{idle} time.

3.3.2 Derive $quantum_i$

Deriving $quantum_i$ for WRR and DWRR is relatively simple. Recall that $quantum_i$ defines the maximum amount of data a queue can send in a round.

- **WRR**: In the latest implementation of WRR on chips, each queue is configured with a quantum Q_i worth of bits, and in each round queue i can at most transmit Q_i (rather than a fixed number of packets in earlier proposals [21]). Thus, $quantum_i = Q_i$ for WRR.
- **DWRR**: In the implementation of DWRR, each queue is also configured with a quantum Q_i worth of bits. Typically, Q_i should be no smaller than maximum transmission unit (MTU) to provide $O(1)$ time complexity [26]. Instead of Q_i, the DWRR scheduler maintains a *deficit counter* for each queue to bound the maximum amount of data to send in each round. This deficit counter maintains the unused quota left in previous round, and is incremented by Q_i in current round (or reset to 0 if the queue is empty). Considering queue i keeps backlogged for M rounds, let $sent_i$ denote the total amount of bits sent by queue i in this period. We can bound $sent_i$ as follows [26]:

$$M \times Q_i - MTU \leq sent_i \leq M \times Q_i + MTU \quad (7)$$

  On average, the amount of data queue i can send in each round is: $\frac{sent_i}{M} = [Q_i - \frac{MTU}{M}, Q_i + \frac{MTU}{M}] \approx Q_i$. Thus, we set $quantum_i = Q_i$ for DWRR.

3.4 Discussion

Weighted Fair Queueing: The reader may wonder how to extend MQ-ECN to weighted fair queueing (WFQ) or other fair queueing schemes. Unlike round robin schemes, WFQ does not have the explicit round concept. So it is difficult to directly apply Equation (5) to WFQ.

A straightforward approach is to divide the standard ECN marking threshold to all backlogged queues. For example, we can define a queue is backlogged if it is not empty. We use w_{sum} to denote the sum of weights of all backlogged queues, and w_{sum} can also be updated using exponential filter like T_{round}. The ECN marking threshold for queue i can be set as $\frac{w_i}{w_{sum}} \times C \times RTT \times \lambda$. However, this formula is built on an implicit assumption that a non-empty queue is able to use up its weighted fair share rate, which may not always hold in all cases. In fact, $\frac{w_i}{w_{sum}} \times C$ is the lower bound for $w_i\alpha$ when link

is congested. Hence, the above approach may underestimate weighted fair share rates and derive lower ECN marking thresholds. We believe a key factor for this approach is to accurately define the backlogged queues and quickly identify them, which may not be so easy. However, a more general MQ-ECN scheme that better supports WFQ is our future work.

Strict Priority Queueing: In production DCNs, network operators may reserve few (typically one) extra queues with strict higher priority to deliver a small amount of important control messages. The remaining queues are typically scheduled using DWRR/WRR in the lowest priority. MQ-ECN does not directly apply to such scenario as well. As an approximation, for strict higher priority queues without round concept, we statically set their marking thresholds to the standard marking threshold $C \times RTT \times \lambda$. For DWRR/WRR queues in the lowest priority, we still apply MQ-ECN (Equation (5)) to calculate dynamic marking thresholds.

Probabilistic Marking: ECN/RED actually has two thresholds and a maximum marking probability to perform a probabilistic marking. When two thresholds are set to the same value, the maximum marking probability no longer takes effect. Some transports, *e.g.*, DCQCN [32], require such probabilistic marking by setting different values for two thresholds. MQ-ECN can be easily extended to perform such probabilistic marking. Let K_{min}, K_{max} and P_{max} denote the low standard threshold, the high standard threshold and the maximum marking probability derived under the single queue model. The low/high thresholds and the maximum probability for queue i of MQ-ECN are $K_{i,min}$, $K_{i,max}$ and $P_{i,max}$, respectively. They can be given as: $K_{i,min} = K_{min} \times min(\frac{quantum_i}{C \times T_{round}}, 1)$, $K_{i,max} = K_{max} \times min(\frac{quantum_i}{C \times T_{round}}, 1)$, and $P_{i,max} = P_{max}$.

4 Implementation

In this section, we first analyze the feasibility of MQ-ECN implementation on switching chips, and then describe a software prototype of MQ-ECN in detail. An implementation of MQ-ECN on switching chip hardware is under negotiation but beyond the scope of this work.

4.1 Switch Implementation

In typical switch implementation for ECN/RED, there is a comparison for an averaged queue length and a static threshold, which is setup using registers. In MQ-ECN's implementation, the comparison is between the same average queue length and a dynamic threshold. In this section, we discuss the implementation complexity for the dynamic threshold. To calculate K_i for a queue, we need to calculate T_{round}. The calculation of T_{round} can be implemented by the moving average of round robin time taken on scheduling. Compared to the per queue average queue length, T_{round} is per port. Therefore, MQ-ECN keeps the same scale implementation complexity as ECN/RED, as we just need one additional register per port to store T_{round}.

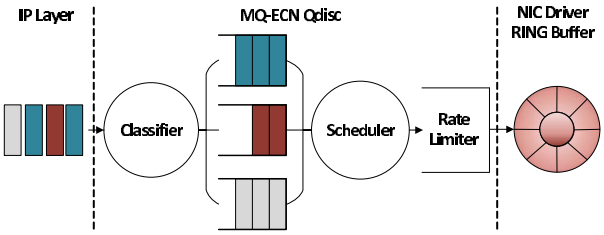


Figure 4: MQ-ECN software stack.

However, one potential challenge is that too frequent moving average calculation cannot be easily achieved by switching chips. This problem becomes more and more serious as the link capacity of production DCNs keeps increasing in recent years. To the best of our knowledge, some chip vendors take a time based moving average calculation to address this challenge. For example, for average queue length calculation of ECN/RED, the moving average is taken for a static interval at microseconds granularity rather than each packet arrival/departure. Similarly, our T_{round} moving average update can also be implemented using a time based version. In our discussion, we prefer a time related to the transmission time of an MTU. Taking 10G link capacity and MTU=9KB as an example, the transmission time is 7.2μs. The moving average calculation at this time granularity can be achieved by most switching chip vendors as we know.

In summary, MQ-ECN maintains the same scale implementation complexity as ECN/RED as it just requires one addition moving average register per port.

4.2 Software Prototype

Since we cannot program our switching chips, we use a server with multiple Network Interface Cards (NICs) to emulate the switch and implement MQ-ECN on top of that. MQ-ECN is implemented as a new Linux queueing discipline (`qdisc`) kernel module. Hence, we can avoid the overhead of data copy and context switch between user and kernel space. Figure 4 shows the software stack of MQ-ECN. MQ-ECN prototype has three components: a packet classifier, a packet scheduler, and a rate limiter. Instead of modifying the Linux `tc`, we expose new `sysctl` interfaces for users to configure the new `qdisc` module in user space.

Packet Classifier: MQ-ECN kernel module maintains multiple FIFO transmit queues. Packets are classified into different queues based on the IP DSCP field. When

MQ-ECN kernel module receives a packet from IP layer, it: 1) classifies the packet based on DSCP value, 2) calculates the ECN marking threshold of the corresponding queue, 3) performs ECN marking if needed, and 4) enqueues the packet.

Packet Scheduler: The packet scheduler of MQ-ECN kernel module is built on the top of the DWRR scheduler available in Linux. Our implementation can also be easily extended to WRR by resetting deficit counter to 0 whenever a queue finishes its service in a round.

The DWRR scheduler maintains a linked list for all active queues. When an empty queue receives a packet, it is inserted to the tail of the linked list. The scheduler always serves the head node of the linked list. If a queue just finishes its service and but still has packets, it is inserted to the tail of the linked list again. Each queue has a variable T_{start} to store the time stamp when this queue is inserted to the linked list last time. Whenever a queue finishes its service, we use current time minus T_{start} to get a sample of T_{round}. In this way, we only sample T_{round} on active queues, just as §3.3.1 described.

Rate Limiter: One implementation challenge is to make the buffer occupancy in `qdisc` reflect the real buffer occupancy of the emulated switch port. A packet dequeued by `qdisc` further goes through NIC driver and NIC hardware before it is delivered to the wire. If we dequeue packets from `qdisc` as fast as possible, many packets can still get queued on NIC driver and hardware. Consequently, the buffer occupancy in `qdisc` is likely to be smaller than the actual buffer occupancy of the emulated switch port. To avoid such impact, we implement a Token Bucket rate limiter to rate-limit the outgoing traffic from `qdisc` at 99.5% of the line rate. The bucket size is ∼1.67 MTU (2.5KB) in our experiment, which is large enough to saturate more than 99% of link capacity while introducing little burstiness. In this way, we can eliminate undesirable buffering in other places and make the buffer occupancy in `qdisc` accurately reflect the buffer occupancy of the emulated switch port.

To confirm the effectiveness of the rate limiter, we install MQ-ECN kernel module on a server with 10 GbE NICs to emulate switch. Two other servers are connected to this software switch. The shaping rate is 995Mbps. The bucket size is 2.5KB. We start a long-lived TCP flow to measure goodput. The goodputs with and without kernel module are 937Mbps and 942Mbps, respectively. MQ-ECN module introduces ∼0.53% goodput degradation, exactly enforcing the desired rate (995Mbps).

5 Evaluation

In this section, we use testbed experiments and ns-2 [5] simulations to answer following three key questions:

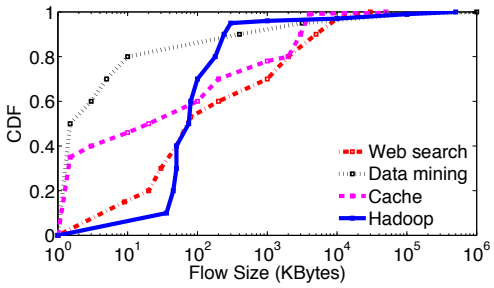
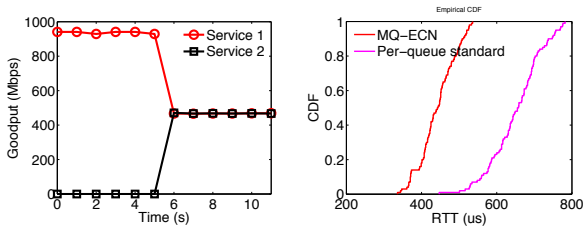


Figure 5: Flow size distributions used for evaluation.

- **How does MQ-ECN perform in practice?** In a static flow experiment (§5.1.1), we show that MQ-ECN strictly preserves weighted fair sharing while maintaining high throughput and low latency. Using realistic workloads in our testbed experiments (§5.1.2), we show that MQ-ECN outperforms the other schemes. For example, it achieves up to 72.8% lower 99th percentile FCT for small flows compared to the per-queue ECN with standard threshold.
- **Does MQ-ECN scale to large data center topologies?** Using large-scale ns-2 simulations (§5.2), we show that MQ-ECN scales to multi-hop topologies and delivers the best overall performance. For example, it reduces the 99th percentile FCT for small flows by up to 43.7% compared to the standard threshold, while achieving up to 13.2% lower average FCT for large flows compared to the minimum threshold.
- **How robust is MQ-ECN to network environments and parameter settings?** Using a series of targeted simulations (§5.3), we show that MQ-ECN is robust to 1) the number of queues (§5.3.1), 2) transport protocol (§5.3.2), and 3) parameter settings (§5.3.3).

Benchmark traffic: We use four traffic distributions based on measurements from production DCNs (Figure 5): a web search workload [6], a data mining workload [16], a cache workload [25], and a Hadoop workload [25]. In general, all the workloads are heavy-tailed. Among them, the web search workload and the cache workload are more challenging since they are less skewed. For example, ∼60% of all bytes in the web search workload are from flows smaller than 10MB. Consequently, in the web search workload, it is likely that several flows are concurrently active in the same link, thus increasing network contention. Ideally, different services have different traffic distributions. However, to create more challenges, we hypothetically use the most challenging web search workload for all services in the testbed experiments. We use all the four workloads in the large-scale simulations.

Schemes compared: We evaluate the performance of three schemes, MQ-ECN, per-queue ECN with the standard threshold and per-queue ECN with the minimum



(a) Aggregate goodput (b) RTT

Figure 6: [Testbed] (a) Aggregate goodput of two services achieved by MQ-ECN. (b) RTT distributions.

threshold. We exclude per-port ECN as it can violate weighted fair sharing. For MQ-ECN, there are only two parameters to configure (§3.3.1): β and T_{idle}. In both testbed experiments and simulations, we set β to 0.75 and T_{idle} to the transmission time of a MTU (12μs in 1G testbed and 1.2μs in 10G simulation). We further analyze the sensitivity to above parameters in §5.3.

Performance metric: We use the flow completion time (FCT) as the performance metric. We consider the overall average FCT of all flows and average FCT across different flow sizes (small, medium and large). To evaluate tail latency, we also show the 99th percentile FCT of small flows. For clear comparison, we normalize the final FCT results (not per flow FCT) to the values achieved by per-queue ECN with the standard threshold.

5.1 Testbed Experiments

Testbed setup: We build a small-scale testbed with 9 servers connected to a 9-port server-emulated MQ-ECN-enabled switch. Each server is a Dell PowerEdge R320 with a 4-core Intel E5-1410 2.8GHz CPU, 8G memory, a 500GB hard disk, and the server-emulated switch has 10 Broadcom BCM5719 NetXtreme Gigabit Ethernet NICs. We reserve one NIC on the server-emulated switch for control access. All the servers run Linux kernel 3.18.11 and DCTCP is enabled. We set TCP RTOmin to 10ms as many proposals suggest [6, 19, 30]. On the server-emulated switch, we deploy a MQ-ECN `qdisc` kernel module with 4 queues (per port) scheduled by DWRR. The quantum of each queue is 1MTU. We disable offloading techniques on the switch to avoid large segments. Each switch port has 96KB buffer which is completely shared by all the queues in a first-in-first-serve bias. The base RTT is ~250μs. Given that, we set the standard ECN marking threshold to 32KB.

5.1.1 Static Flow Experiment

We begin with a basic static flow experiment to show that MQ-ECN can achieve high throughput, low latency and weighted fair sharing simultaneously. We start 5 TCP flows from two senders to the same receiver and classify them into two services based on their senders. Service 1 has 1 flow and service 2 has 4 flows. Both services have a equal-quantum dedicated queue on the switch. We evaluate the performance of MQ-ECN and per-queue ECN with the standard threshold.

Figure 6(a) shows the sharing results achieved by MQ-ECN. The sharing result achieved by the standard threshold is quite similar to Figure 6(a). We omit it due to space limitation. In contrast to Figure 3(a), both services roughly achieve the same goodput. We also use ns-2 simulation to reproduce the experiment and find that MQ-ECN achieves similar convergence time as the standard threshold. This suggests that MQ-ECN can strictly preserve weighted fair sharing. Furthermore, the sum of aggregate goodputs of two services achieved by MQ-ECN is ~936Mbps. This suggests that MQ-ECN can fully utilize the link capacity.

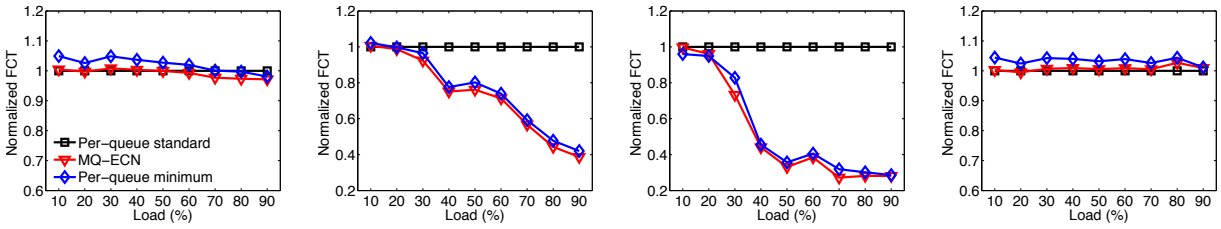
We also measure RTT of the dedicated queue of service 2 using `ping`. Figure 6(b) gives the RTT distributions achieved by MQ-ECN and the standard threshold. Compared to the standard threshold, MQ-ECN achieves 32.3% (651μs to 441μs) and 31.5% (782μs to 536μs) lower RTT in average and the 99th percentile. Recall that the base RTT is ~250μs. Hence, MQ-ECN reduces queueing delay by ~50%. This suggests that MQ-ECN can achieve low latency.

5.1.2 Realistic Workloads

For this experiment, we develop a client/server application to generate dynamic traffic according to the web search workload [6]. The client application, running on 1 server, generates requests through persistent TCP connections to the other 8 servers to fetch based on a Poisson process. The server applications, running on the other 8 servers, respond with requested data. To map a flow to a service queue, the server application uses `setsockopt` to set DSCP for outgoing packets. We create two traffic patterns: *balanced traffic* and *unbalanced traffic*. In balanced traffic, each flow is randomly mapped to a service queue. In unbalanced traffic, each flow is mapped to 4 service queues with probabilities of 10%, 20%, 30% and 40%. We vary the network load from 10% to 90%.

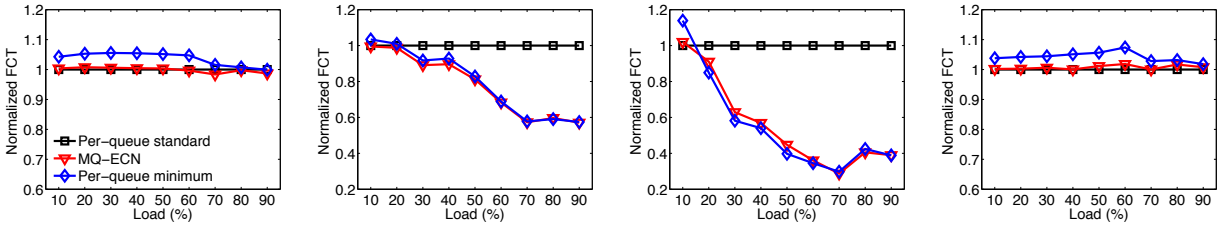
Figure 7 and 8 show the overall average FCT (a), FCT across small (0,100KB] (b,c) and large (10MB,∞) flows, respectively. Due to space limitation, we omit the results for the medium (100KB,10MB] flows whose performance is quite similar to that of overall average FCT. We make the following three observations.

Overall: MQ-ECN generally achieves the best overall average FCT. Compared to the standard threshold, MQ-ECN delivers similar performance at low loads ($\leq 50\%$) and achieves up to ~2.85% (balanced) and ~1.65% (unbalanced) lower FCT at high loads. When the load is



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

Figure 7: [Testbed] Balanced traffic pattern: FCT statistics across different flow sizes.



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

Figure 8: [Testbed] Unbalanced traffic pattern: FCT statistics across different flow sizes.

low, it is not likely that multiple queues are concurrently active. Hence, both MQ-ECN and the standard threshold can achieve good FCT due to their high throughput. When the load is high, MQ-ECN achieves better performance by efficiently reducing packet latency. Compared to the minimum threshold, MQ-ECN outperforms it at all loads for both traffic patterns.

Small flows: MQ-ECN performs similar as the minimum threshold for small flows while significantly outperforming the standard threshold. Compared to the standard threshold, MQ-ECN reduces the average FCT for small flows by up to ∼61.3% (balanced) and ∼42.9% (unbalanced). The performance gap on 99th percentile FCT is even larger: MQ-ECN achieves up to ∼72.8% (balanced) and ∼71.3% (unbalanced) lower 99th percentile FCT for small flows. We attribute the large tail FCT of the standard threshold to its poor burst tolerance. When all the 4 queues are concurrently active, the total buffer occupancy achieved by the standard threshold can easily reach 128KB (4×32KB), thus overfilling shallow switch buffer (96KB).

Large flows: MQ-ECN also achieves good performance for large flows. Compared to the standard threshold, the average FCT for large flows of MQ-ECN is within ∼2.7% for the balanced traffic pattern and ∼1.8% for the unbalanced traffic pattern. This is expected because MQ-ECN adjusts the ECN marking threshold for each queue based on its dynamic weighted fair share rate, thus not adversely affecting its throughput. By contrast, the minimum threshold, due to its throttle on rates, delivers the worst performance for large flows: it achieves ∼1.2–4.4% (balanced) and ∼1.8–7.4% (unbalanced) larger FCT compared to the standard threshold.

5.2 Large-scale NS-2 Simulations

In this section, we use ns-2 [5] simulations to evaluate MQ-ECN's performance in large-scale DCNs.

Topology: We use a 144-host leaf-spine topology with 12 leaf (ToR) switches and 12 spine (Core) switches. Each leaf switch has 12 10Gbps downlinks to hosts and 12 10Gbps uplinks to spines, forming a non-blocking network. The base RTT across the spine (4 hops) is 85.2μs. We employ ECMP for load balancing.

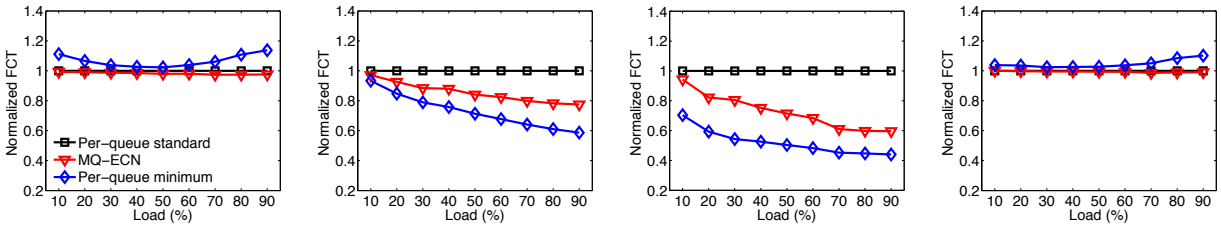
Workloads: We use all the 4 flow size distributions in Figure 5. Since there are 144 hosts, we have 144×143 communication pair in total. We evenly map these pairs to 8 services. Every two services share a flow size distribution. All simulations last for 50000 flows.

Transport: We use DCTCP by default. The initial window is 16 packets. We set both initial and minimum value of TCP RTO to 5ms.

Switch: Each switch port has 300KB buffer shared by all the 8 queues in a first-in-first-serve bias. We set the standard marking threshold to 65 packets. We use both DWRR and WRR in our simulations. All the queues have the same quantum of 1.5KB.

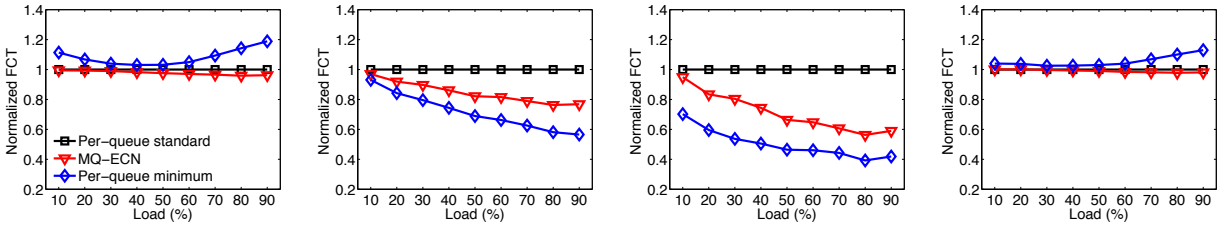
Figure 9 and 10 give the FCT results across different flow sizes. In the interest of space, we omit the results for the medium flows (100KB,10MB] whose performance trend is very similar to that of overall average FCT. We have the following three observations.

Overall: MQ-ECN generally achieves the best overall performance, consistent with our testbed experiments in §5.1. Compared to the standard threshold, MQ-ECN achieves up to ∼4.1% lower average FCT. The performance of the minimum threshold is volatile. Compared



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

Figure 9: [Simulation] DWRR: FCT statistics across different flow sizes



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

Figure 10: [Simulation] WRR: FCT statistics across different flow sizes

to the other schemes, it shows obvious performance gap at low loads. For example, it achieves ∼11% larger FCT at 10% load compared to the standard threshold. This is because, at low loads, it is likely that only few queues are active. In such scenario, the minimum threshold severely degrades throughput. When the load increases, the minimum threshold shows better performance as expected. However, at extremely high loads, the performance of the minimum threshold degrades again, which is counter-intuitive. We suspect the reason is because ECMP is agnostic to both flow sizes and service classes, and it does not guarantee to spread large flows from the same service class across different paths. When the load is unbalanced or large flows from the same class are concentrated (even overall traffic is balanced), the problem may arise.

Small flows: MQ-ECN greatly outperforms the standard threshold for small flows. Compared to the standard threshold, MQ-ECN reduces the average and 99th percentile FCT for small flows by up to 23.7% and 43.7%, respectively. Compared to the minimum threshold, the average FCT for small flows with MQ-ECN is within 24.3% for DWRR and 26.4% for WRR. The performance gap is because that the minimal threshold trades throughput for better latency. In our simulation, 65.1% of small flows are smaller than 24KB (16 packets), which are small enough to complete within one RTT. The minimum threshold can provide ideal performance for such mice flows since their FCTs are only determined by latency.

Large flows: For large flows, MQ-ECN achieves comparable performance as the standard threshold while significantly outperforming the minimum threshold. This suggests MQ-ECN achieves high throughput. MQ-ECN even slightly outperforms the standard threshold at extremely high loads. For example, compared to the standard threshold, MQ-ECN with WRR achieves 2.1% lower average FCT for large flows at 80% load. This is because MQ-ECN can provide better burst tolerance, thus greatly reducing packets drops and retransmissions. As we check, at 80% load, the standard threshold with WRR causes 720 TCP timeouts while MQ-ECN only has 45. As expected, the minimum threshold performs the worst. For example, it is 13.2% worse than MQ-ECN with WRR at 90% load for large flows.

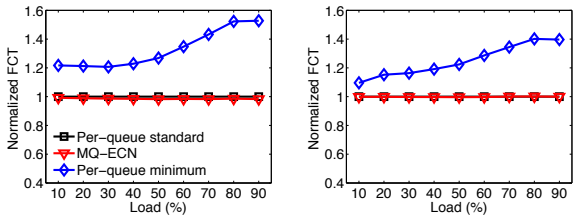
5.3 MQ-ECN deep dive

In this section, we conduct a series of targeted simulations to evaluate MQ-ECN's robustness to network environments and parameters. By default, we use DCTCP as the transport protocol and DWRR (8 queues) as the packet scheduler. The other settings are same as §5.2

5.3.1 Impact of the Number of Queues

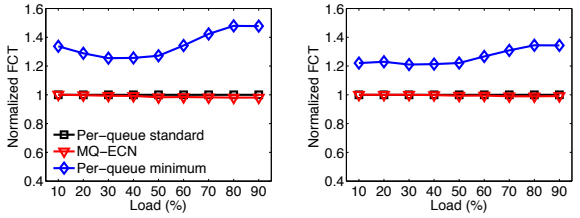
In the future, switching chips may support more and more queues. To verify whether MQ-ECN can scale to a larger number of queues, we increase the number of queues per switch port to 32. In the interest of space, we only show overall performance and average FCT for large flows in Figure 11.

We find that MQ-ECN still maintains the best overall performance. However, the performance of the minimum threshold degrades significantly, particularly at high loads. It is 35.7% worse than MQ-ECN at 90% load for overall average FCT. The reason behind this is: at a given load, the more queues we use, the less likely that the majority of queues are concurrently active. Thus,



(a) Overall (b) (10MB,∞): Avg

Figure 11: [Simulation] FCT with 32 queues.



(a) Overall (b) (10MB,∞): Avg

Figure 12: [Simulation] FCT with ECN*.

the throughput of the minimum threshold is affected, enlarging FCT for large flows. By contrast, MQ-ECN can effectively adjust ECN marking thresholds based on traffic dynamics to maintain high throughput. This indicates that MQ-ECN is robust to the number of queues.

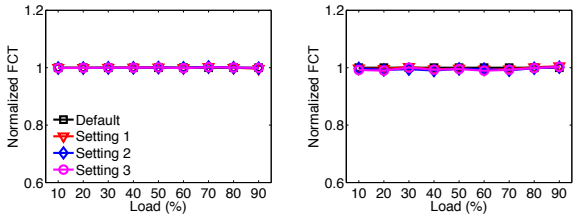
5.3.2 Impact of Transport Protocol

In addition to DCTCP, there are many other ECN-based DCN transport protocols, such as ECN* [31]. Unlike DCTCP, ECN* simply reduces the window by half in the presence of ECN. Hence, ECN* is more sensitive than DCTCP. A lower ECN marking threshold can greatly affect the throughput of ECN* [31]. For example, with zero buffering, DCTCP can maintain 94% throughput in theory [7] while ECN* only achieves 75% throughput.

We evaluate the performance of all the three schemes with ECN*. We set the standard ECN marking threshold to 84 packets. As Figure 12 shows, MQ-ECN still outperforms the other schemes under ECN*. This indicates that MQ-ECN can efficiently maintain high throughput by adjusting ECN marking thresholds based on dynamic weighted fair share rates. As expected, the throughput of the minimum threshold degrades severely. Compared to MQ-ECN, it increases FCT for large flows by ∼22–36%.

5.3.3 Sensitivity to Parameters

We now try to explore MQ-ECN's sensitivity to parameters. Recall that MQ-ECN has two parameters to configure: β and T_{idle}. In our simulation, β is 0.75 and T_{idle} is 1.2μs (1.5KB/10Gbps) by default. Here, we compare the default setting with the other 3 settings: 1) β=0.875, T_{idle}=7.2μs, 2) β=0.5, T_{idle}=1.2μs, and 3) β=0.75, T_{idle}=7.2μs.



(a) Overall (b) (0,100KB]: Avg

Figure 13: [Simulation] FCT with different parameters. FCT is normalized to the value achieved by default setting.

Figure 13 gives the FCT results achieved by above 4 settings. Note that FCT is normalized to the value achieved by the default setting. In general, compared to the default setting, the other 3 settings achieve very close performance. Their performance is within 0.37% for overall average FCT and 1.05% for average FCT of small flows. The results suggest that MQ-ECN is robust to different parameter settings.

6 Related Work

Tons of literatures working along the general ECN/RED, *e.g.*, [6, 12, 15, 24, 29, 31, 32], are related to MQ-ECN. For space limitation, we do not introduce these literatures one by one. However, the key difference is that MQ-ECN is perhaps the first effort that investigates the problem of applying ECN in multi-service multi-queues production DCNs. MQ-ECN does not challenge the fundamental principle of prior work on ECN; instead it builds on the theory (*e.g.*, the standard ECN marking threshold) developed by prior work especially [6, 7, 31], and extends its applicability to a new production environment.

7 Conclusion

In this paper, we have presented MQ-ECN for multi-service multi-queue DCN that is capable of delivering both high throughput and low latency simultaneously, while maintaining weighted fair sharing. We have shown that MQ-ECN achieves all its properties without requiring advanced features and is readily implementable with existing commodity chips. At last, we performed a series of testbed experiments and large-scale simulations to validate its performance as well as robustness to different network environments and parameter settings.

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 under Grant No.2014CB340303. We would like to thank the anonymous NSDI reviewers and our shepherd Changhoon Kim for their constructive feedback and suggestions.

References

[1] http://www.pica8.com/documents/pica8-datasheet-64x10gbe-p3922-p3930.pdf.

[2] "Broadcom BCM56850," https://www.broadcom.com/collateral/pb/56850-PB03-R.pdf.

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

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

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

[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] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *IMC 2010*.

[12] L. Chen, S. Hu, K. Chen, H. Wu, and D. Tsang, "Towards Minimal-Delay Deadline-Driven Data Center TCP," in *HotNets 2013*.

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

[14] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 4. ACM, 1989, pp. 1–12.

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

[16] 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*.

[17] 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*.

[18] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable Message Latency in the Cloud," in *SIGCOMM 2015*.

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

[20] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. Liu, and F. Dogar, "Friends, not Foes - Synthesizing Existing Transport Strategies for Data Center Networks," in *SIGCOMM 2014*.

[21] J. Nagle, "On Packet Switches with Infinite Storage," *IEEE Transactions on Communications*, vol. 35, no. 4, pp. 435–438, Apr 1987.

[22] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate fairness through differential dropping," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 23–39, 2003.

[23] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Transactions on Networking (ToN)*, vol. 1, no. 3, pp. 344–357, 1993.

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

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

[26] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *Networking, IEEE/ACM Transactions on*, vol. 4, no. 3, pp. 375–385, 1996.

[27] 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 Googles Datacenter Network," in *SIGCOMM 2015*.

[28] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high-speed networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 1, pp. 33–46, 2003.

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

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

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

[32] 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*.