Computer Networks 91 (2015) 593–613

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Alleviating flow interference in data center networks through fine-grained switch queue management☆

Guo Chen, Youjian Zhao, Dan Pei*

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 21 October 2014
Revised 5 August 2015
Accepted 27 August 2015
Available online 9 September 2015

Keywords:
Data center networks
Switch queue management scheme
Flow interference

ABSTRACT

Modern data centers need to satisfy stringent low-latency for real-time interactive applications (*e.g.* search, web retail). However, short delay-sensitive flows generated from these applications often have to wait a long time for memory and link resource occupied by a few of long bandwidth-greedy flows because they share the same switch output queue (OQ). To address the above flow interference problem, this paper advocates more fine-grained flow separation in the switches than traditional OQ. We propose CQRD, a simple queue management scheme for data center switches, without change to the transport layer and requiring no coordination among switches. Through simulations, we show that CQRD can reduce the flow completion time (FCT) of short flows by more than 25% in a single switch and up to 50% in a multi-stage data center network, only at the cost of a minor goodput decrease of large flows. Additionally, just a 50% deployment of CQRD in top-of-rack (ToR) switches can lead to a ∼10–24% FCT reduction of short flows. Moreover, CQRD can improve short flows' FCT by ∼30–40% from OQ switches, using DCTCP (Alizadeh et al., 2010) [2] transport in DCN. Furthermore, we validate the feasibility of CQRD approach by implementing an 8 × 8 logical CQRD switch through simply changing the configurations of existing commodity switches. Also, we use a small testbed experiment to verify the implementation and the effectiveness of CQRD to alleviate flow interference in real environment.

1. Introduction

As people and business increasingly rely on the Internet in their daily life and work, the performance requirement on the data center networks (DCN), where most of the Internet applications are hosted, has become more stringent. However, recent studies have shown that short delay-sensitive flows from the real-time interactive applications (*e.g.* search, web retail), although contributing to majority of flows in DCNs [3], often have to wait a long time at switches for buffer and bandwidth resources occupied by a few of long bandwidth-greedy flows (*e.g.* backup, replication etc.). This causes a dramatic increase to the flow completion time (FCT) of most short flows, which can be more than 10 times higher [2].

As analyzed in many recent studies [2,4–6], the fundamental reason for the above mentioned performance degradation is that the commodity DCN switches' traditional and coarse (output queue, or OQ) queue management schemes are not suited well for the DCN traffic characteristics, causing

☆ This paper was previously presented in part [1] at LCN'14, Edmonton Canada, September 2014. Extensions to the conference version include detailed description to the CQRD approach and analysis to its impact on TCP performance. Also, we discuss how CQRD co-works with transport methods with adaptive rate control schemes. Moreover, a small-scale implementation and testbed experiments are added. Additionally, new experiments about incremental deployment, the impact of different buffer sizes, and transport methods with adaptive rate control schemes to CQRD's performance, are added in this paper.

* Corresponding author. Tel.: +86 (10) 62792837.
E-mail addresses: chen-g11@mails.tsinghua.edu.com (G. Chen), zhaoyoujian@tsinghua.edu.com (Y. Zhao), peidan@tsinghua.edu.com (D. Pei).

http://dx.doi.org/10.1016/j.comnet.2015.08.033
1389-1286/

unnecessary **flow interference**. We define that two flows **are interfered** by each other, when they pass through a switch while overlapping in time, and contend for some shared resources at switches, such as queue memory, or link capacity, *etc*. Many transport layer solutions [2,4,7,8] have been proposed to get around the coarse queue management problem by optimizing flows' rate assignment to keep the switch queues near empty. However, precise rate control is a great challenge due to the bursty traffic in DCN. Thus, only using these transport methods, flow interference can still happen in the coarse OQ due to flow burstiness. Therefore, a more fine-grained queue management scheme could be a good complement to these transport methods (more analysis in Section 4.3.3 and shown by experiments in Section 6.6). Moreover, all these transport approaches need to modify end host's protocol stack, which makes them facing some deployment difficulties. Another direction to solve the above performance degradation problem is flow scheduling [5,6]. These methods try to implement optimal flow scheduling to minimize the FCT of short flows. However, these solutions require significant changes on existing software or hardware of end hosts or switches, which are challenging to deploy. Furthermore, using these flow scheduling methods, small flows may still be interfered by long flows in original OQ switches, because of non-optimal scheduling [5] and long scheduling latency [6]. A fine-grained queue management scheme could also be complementary to these methods. More detailed discussion about related works is in Section 2.

Different from these previous approaches, we address the DCN flow interference problem by directly tackling its root cause: coarse switch queue management schemes. Hence, we argue that the DCN flow interference, especially for the interference between large number of small delay-sensitive flows and a small number of giant flows, calls for a more fine-grained queue management than the current output queue (OQ) in the commodity DCN switches in order to alleviate the flow interference problem in DCNs. Toward this direction, in this paper we propose a simple[1] queue management scheme, crosspoint-queue with random-drop (CQRD). In CQRD, a separate queue is assigned to each pair of input and output port, and packets are randomly dropped upon the full of crosspoint-queue (or randomly marked with ECN [9] tag upon the queue length above the threshold). This paper presents the design, analysis, implementation and evaluation of CQRD, to alleviate flow interference in DCN. Our contributions can be summarized as follows:

- We revisit the mature crosspoint-queue and random-drop techniques, and combine them together into a simple fine-grained queue management scheme named CQRD, to solve flow interference problem in DCN. These two underlying techniques are widely used in current switching hardwares [10,11], which makes it simple to implement CQRD. Moreover, the proposed approach requires neither any coordination among switches, nor modification to end hosts, which makes it easy to deploy.
- For DCN environment which uses adaptive transport rate control based on ECN [9] (*e.g.* DCTCP [2]), we accordingly design CQRD with a random-mark scheme (see Section 4.3.3) besides random-drop for traditional TCP environment. A hybrid of CQRD and transport layer methods achieves even better performance.
- We implement an 8×8 logical CQRD switch through simply modifying the configurations of existing commodity switches, which validates the simplicity and feasibility of CQRD's approach.
- Through simulations, we show that CQRD significantly reduces the flow completion time of short flows by more than 25% in a single switch and up to ~50% in a multi-stage data center network, only potentially at the cost of a minor goodput decrease for large flows. Furthermore, CQRD can be incrementally deployed. Just a 50% deployment of CQRD in ToR switches leads to a ~10–24% FCT reduction of short flows. Moreover, we show that CQRD can further improve short flows' FCT by ~30–40% from OQ switches, using DCTCP transport in DCN. Also, we conduct a small testbed experiment to verify the CQRD implementation and the effectiveness of CQRD to alleviate flow interference in real environment.

The rest of the paper is organized as follows. We review the related works in Section 2. In Section 3, we use analysis and simulation to study flow interference and its performance impact in traditional OQ, HCF (state-of-the-art switch queue management for DCN) [12], and classic CQ [10]. In Section 4, we present the CQRD approach and theoretically analyze how it can alleviate flow interference in DCN. In Section 5, we introduce our implementation of a small scale CQRD switch and discuss the implementation of a large scale CQRD switch through application-specific integrated circuit chips (ASIC). In Section 6, we use simulation experiments to show that CQRD greatly improves the overall DCN performance, both at fully and partially deployment. Also, we study the impact of different buffer sizes to CQRD's performance. Additionally, we show that a hybrid of CQRD and transport layer methods could achieve even better performance. In Section 7, we evaluate CQRD in a small testbed. Finally we conclude in Section 8.

2. Related works

Long delay of short delay-sensitive flows due to flow interference is a well known problem in data center network. We describe several solutions below and illustrate the difference between CQRD and them.

2.1. Transport layer rate control

A major direction of prior work uses transport layer rate control to reduce flow completion time of short flows. DCTCP [2] and HULL [7] apply adaptive rate control schemes based on ECN [9] and packet pacing, to control the rate of giant flows. By keeping the queue size of switches near empty, they improve the overall FCT of short flows. D^2TCP [8] uses the deadline information for rate control, which allocates the rate of each flow according to their deadline information.

However, as also pointed out by their own authors [6], precise rate control is challenging due to the bursty traffic

[1] By "simplicity", in this paper, we mean that the CQRD design is easy to understand without much complexity, and the implementation is easy based on existing and mature underlying techniques.

in DCN. It is very hard to control the sending rate to exactly fully utilize the bandwidth, while keeping network buffer near empty. This is also shown in our later analysis (Section 4.3.3) and experiments (Section 6.6). Therefore, in traditional OQ switches, packets of short flows still may queue behind a considerable number of long flows' packets due to a burst of these long flows, which increases short flows' queuing delay. What is worse, because short flows and long flows compete the same output buffer resource, short flows' sending rate may also be throttled when the long flows are occupying the output queue. A more fine-grained queue management scheme, such as CQRD, is a good complement to these transport layer methods, which could offer further performance improvement.

Moreover, all these methods require a modification to end hosts' TCP stack to implement their rate control schemes, which leads to some challenges in deployment. To be more specific, on one hand, network operators often do not control the end-host stack (*e.g.* in a public cloud or with hosts using closed-source operating systems). On the other hand, even if they do, some high performance applications (such as low latency storage systems [13,14]) implement their own transport, and bypass the kernel. Furthermore, they are not well compatible with legacy TCP. In addition to TCP stack change, some of them even require a significant change of end hosts' NIC [7] and/or switch hardware [4,7], which further increases the difficulty of their deployment.

2.2. Flow scheduling

Another direction to solve this problem is the flow scheduling. Recent work such as PDQ [5] and pFabric [6] try to implement optimal flow scheduling to minimize the FCT of short flows. They preemptively schedule each flow based on certain priorities (*e.g.* sizes or deadlines), thus to reduce short flows' FCT.

However, there are several challenges for such flow scheduling approaches. (1) It is an NP-hard problem to compute global optimal flow scheduling [5] in multi-tier DCNs. (2) The latency of scheduling is too long for short flows [6]. Thus, only using these methods, small flows may still be interfered by long flows in original OQ switches, because of non-optimal scheduling or long scheduling latency. A fine-grained queue management scheme such as CQRD is also complementary to these methods. Moreover, these flow scheduling approaches are almost clean-slate ones that require new end host protocol stack and switch hardware design, which can be far from getting deployed in reality.

2.3. Switch based solutions

There are also many queue management schemes in the literature for switches/routers to provide fine-grained separation for TCP flows, such as DRR [15] and SFQ [16]. However, they have been designed for traditional Internet routers, and not applicable for (quite different) traffic characteristics in DCN. Furthermore, recent work in [12] has shown that HCF outperforms them in DCN environment.

The most closely related work to CQRD and the state-of-the-art approach in this space is HCF [12] (Hashed Credits Fair). Similar to CQRD, HCF tries to address the flow interference problem by providing switch queue management scheme that is more fine-grained than OQ. HCF sets two separate queues, one high-priority (HP) and one low-priority (LP), at each output. It hashes all the incoming flows into several bins and assigns each bin a credit. Packets belonging to bins which have credit left will be stored in the HP queue, otherwise in the LP queue. Switches serve the HP queue if it is not empty. When the HP queue becomes empty, HCF resets all the credits to the initial and HP queue is swapped with LP queue. However, it is challenging to hash flows uniformly using static hash function. Therefore, HCF needs to change its hash function periodically, which increases the cost on hardware. Furthermore, as will be shown in Section 3.3, HCF is not fine-grained enough to solve the DCN flow interference problem very well.

Besides queue management, there are also various architectures of switch fabrics in traditional Internet routers. Internet routers are very high-end (*i.e.* with many ports), which consist of multiple line cards connected by the switch fabric on the backplane. Each line card hosts an input/output port of the router and processes incoming/outgoing packets. Because incoming Internet traffic has burstiness and there may often be traffic from multiple input ports destined to the same output, the router has to buffer the packets when contention happens. Limited by the semiconductor technology decades ago, they have to put main buffers on the line cards. It is challenging for a routers switch fabric to switch packets stored in many distributed line cards with line rate (*e.g.* 40G). Thus, in order to reach a high switch capacity (*e.g.* 512 40G) using a low hardware speed-up, various switch fabric architectures are proposed, such as virtual output queue (VOQ) [17] and combined-input-and-crosspoint-queue (CICQ) [18]. Although CQ idea was proposed decades ago, not until recently, [10] revisits the pure CQ switch, and first advocates that routers can implement pure CQ fabric enabled by modern semiconductor technology, which puts all the buffer on switch fabric instead of line cards. Note that previous xCQ (*i.e.* CICQ) is not pure CQ switch fabric, but a variant of VOQ. It also puts main buffers in line cards, and each crosspoint-queue in CICQ only can buffer one packet (or cell). All these architectures focus on how to increase the hardware switch capacity, but not on how to decrease TCP flows FCT affected by flow interference. On the contrary, DCN switches are commodity low-end switches (*e.g.* 24 10G). For these DCN switches such as [19], there are no line cards, but only a single switch chip covering all the switches functionality. All packets are buffered on the switch fabric. They are logic OQ switches, where all packets are buffered at each output port of the switch fabric before being switched out.

Although *pure CQ switch fabric* (first revisited by [10]) originally aims to be a better switch fabric architecture for *Internet routers*, we find that the queue management scheme behind CQ (separate queue for each input–output pair) is promising to solve some TCP problems in DCN. To the best of our knowledge, we are the first one to introduce CQ in DCN switches queue management scheme. To further improve performance, we propose a CQ based switch queue management scheme to solve the flow interference problem. Before our paper, HCF is the start-of-art switch based solution to solve TCP flows FCT problem in DCN.

2.4. Ethernet related techniques

Some 802.1/ethernet related techniques also may help alleviate flow interference in DCNs. Current IEEE 802.1p [20] standard uses 3 bit priorities to differentiate and provide bandwidth guarantee for different traffic types when they share the same link. To solve flow interaction, one possible way is to utilize these priority tags to identify flows with different sizes and assign higher priorities to shorter flows. However, it faces substantial deployment difficulties, because it would require a significant change on current end hosts or switches to tag flows with different priorities according to their sizes.

3. Flow interference: causes and impact

In this section, we discuss the causes and performance impact of flow interference. We study different switch queue management schemes, including traditional output-queue (OQ) with tail-drop, the state-of-the-art DCN fairness queue management scheme (HCF [12]), and classic crosspoint-queue (CQ) with tail-drop [10]. Through analysis and a toy example, we will show that CQ is more promising to solve the flow interference problem. To the best of our knowledge, we are the first one to introduce CQ in DCN switches.

3.1. Performance metrics and definitions

Following the convention in [6], we consider two main performance metrics—**flow completion time** and **goodput**. Flow completion time (FCT) is an important metric for short delay-sensitive flows, and reflects how fast the flow has been successfully transmitted. Goodput equals the flow size divided by its FCT, which is crucial to large bandwidth greedy flows.

We define flows smaller than 100 KB as **small/short flows**, flows larger than 100 KB as **large/long flows**, and flows larger than 1MB as **giant flows** (special case of large flows).

At a given switch, when two flows have the same output port and they overlap in time, we say these two flows **output port contending** or simply **output contending**. Then, we define a **switch path** as the pair of input port and output port on the same switch. When two flows on the same switch path overlap in time, we say these two flows are **switch path contending** or simply **path contending**, which is a special case of output contending. For flows that go through multiple switches, we define two flows as path contending when they are path contending at any of these switches. Contention between flows in a switch leads to potential flow interference, and causes potential performance degradation.

In this section, we only focus on the interference caused by output contending (but excluding the path contending).

3.2. OQ switches

Typically, commodity switches in data centers apply OQ with tail-drop scheme [2,21]. Packets from output contending flows are stored in the same queue at the specific output port. This exploits statistical multiplexing and saves memory resources to achieve a certain packet loss rate. However, it may also cause a strong interference of flows which contend for the same output.

Fig. 1(a) shows an toy example with an eight port OQ switch connecting to server 1–8 with each port respectively. Note that each port consists of an input link and an output link shown in the figure. Assume that there are seven flows coming from inputs I1 to I7 respectively. All the flows are destined to the same output O8. Flows from I6 and I7 are *giant* flows and the other five flows are short delay-sensitive flows. While contending for the same output, the large flows quickly occupy the majority of shared memory resource and the capacity of output link. Packets of the small flows have to wait unnecessarily, queuing behind the packets from the long flow. This leads to a significant increase of packet delay for small flows and impairs the upper-layer applications. As the output contention lasts for a while, the output buffer will

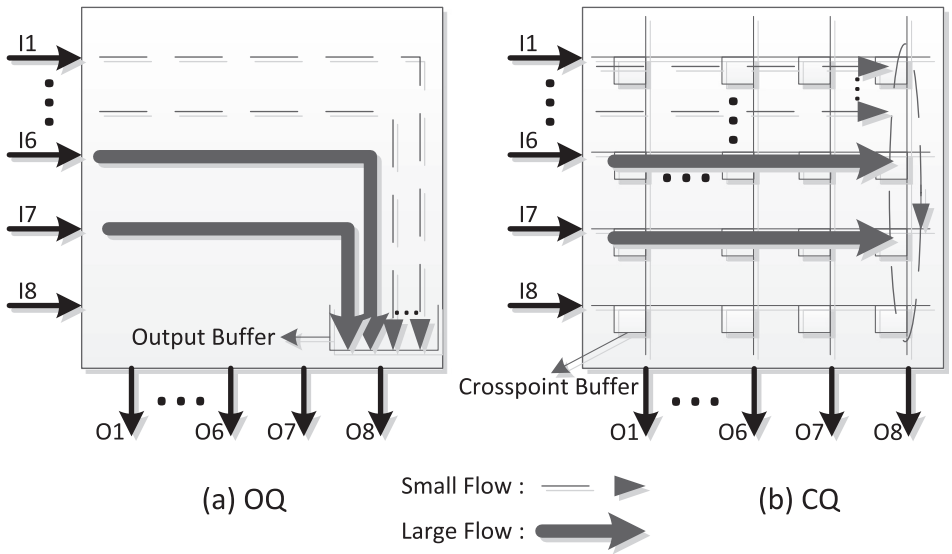


Fig. 1. Flow interference in OQ switch and CQ switch.

be filled up and begin to drop packets of all short flows. This further results in an overall performance degradation.

We use NS2 [22] for simulation and measure the goodput and FCT of each flow in this toy example. Each port has a bidirectional link rate of 10 Gbps and a one-way link delay of 4 μs. Each port has a small output queue with a typical small size of 36 KB [2] (*i.e.*, 288 KB in total). Assume servers 6 and 7 are generating long file-backup traffic to server 8. At time 0, both servers 6 and 7 start to send a 100MB file using TCP (flows 6–7) to server 8. Five milliseconds later, servers 1–5 start five delay-sensitive tasks and each sends a 10 KB TCP flow (flow 1–5) to server 8 at time 0.005 s. TCP SACK [23] are used for all the flows.

The goodput and FCT of each flow are shown in Fig. 2. During this situation, the output queue of port 8 is quickly filled up with packets of flow 6 and 7. Thus, packets of flow 1–5 are continuously dropped when they reach the switch. As a result, their flow completion time soar up to hundreds of milliseconds, while the theoretical ideal FCT should be as low as tens of microseconds. Also, their goodput fall down to lower than 1 Mbps. Apparently, this will cause a dramatic performance degradation of upper-layer applications.

3.3. HCF switches

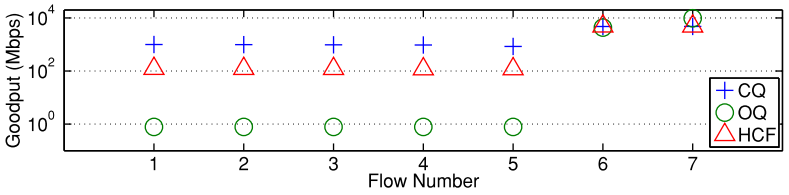
Recently, a hash-based based queue management algorithm for DCN switches called HCF (Hashed Credits Fair) [12] has been proposed. Through hash and assigning credit, flows from different bins can share the HP queue fairly. And when HCF schedules packets out of HP queue, flows from different bins could fairly share the link capacity. That provides a relative good bandwidth fairness. However, it does not provide enough buffer fairness between small flows and giant flows. Many flows coming from different inputs still have to contend for the same buffer resource. While large flows quickly consume the credits of their bins, they fill up the LP queue quickly. If small flows are unluckily hashed to the same bins, they would be dropped. In addition, even if they are not in the same bins, as the number of small flows grows larger, they will also consume their credits and be dropped at the tail of LP queue. That causes packet loss of many small flows with inputs different from the few giant flow.

We simulate HCF switch in the same toy example, with the same 288 KB total memory. All the parameters of HCF are set as the recommended in their paper (two queues with same length, one credit for each of the 20 bins, and periodical XOR hash function). As we can see in Fig. 2, HCF greatly reduces the FCT of short flows and serves two large flows fairly. However, the packets of small flows also have been dropped. That increases their FCT to around 1ms, which should be less than 100 μs without loss.

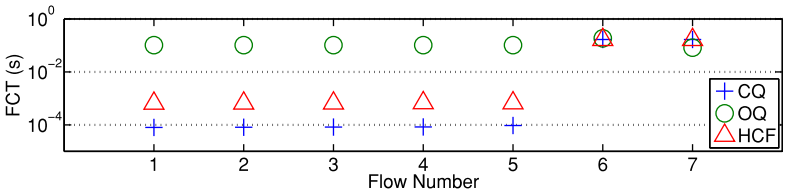
3.4. CQ switches

Recently, the decade-old CQ switching scheme, once considered infeasible for commodity switches when first proposed [24], has been shown to become very feasible using modern semiconductor technologies [10]. CQ offers full flow separation for the flows that are output contending but not path contending. As shown in Fig. 1(b), CQ switch reserves separate memory resources for each pair of inputs and outputs. Packets arrived at each input are first buffered into the crosspoint-buffer (XB). Then each output port, without coordinating with any other ports, picks one of the XBs in its column and schedules the head packet out of the switch. Thus, flows from different input ports have separated buffers. Also, by using simple round-robin (RR) scheduling manner for each output, each flow from different inputs destined to the same output shares approximately the same output link capacity. Therefore, CQ switch can provide a performance separation for flows coming from different inputs.

We simulate CQ switch in the same toy example. With 288 KB memory in total, the same as in OQ switch, each XB of CQ switch has a capacity of 4.5 KB. As shown in Fig. 2, with CQ switch, the FCT of delay-sensitive flows can be three orders of magnitude lower than the OQ switch and one order of magnitude lower than the HCF switch. Also, the goodput of those flows are three orders of magnitude and one order of magnitude higher than the OQ and HCF switch. Although there is only 4.5 KB buffer resource for flows 6 and 7 in CQ switch, the FCT and goodput of these two long flows are



(a) Goodput (Log scale)



(b) Flow Completion Time (Log scale)

Fig. 2. A toy example of flow interference.

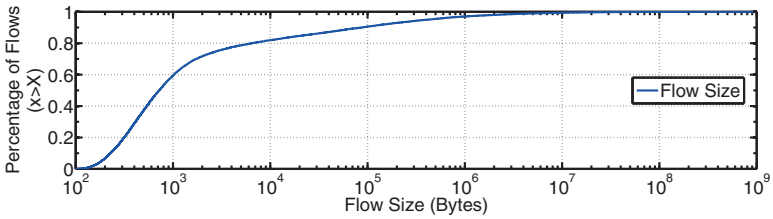


Fig. 3. Traffic workload derived from real data center.

almost the same as in OQ and HCF switches. Meanwhile, CQ switch achieves a very good fairness among flows.

3.5. Summary

Overall, the performance of flows that are output contending with the giant flows is severely degraded by interference by the giant flows in OQ. Although to a less extent than in OQ, HCF still suffers from the same problem. Through separating queues for flows that are output contending but not path contending, a classic CQ switch can achieve much better performance than OQ and HCF in our toy example. We will present how to design a CQ-based scheme for more complex real-world DCNs in the next section.

4. Crosspoint-queue with random-drop scheme

In the previous section, we have shown that CQ performs much better than OQ and HCF in the toy example where there are only very small number of flows and the primary interference is output contending only (excluding path contending). In the real world DCNs, there may be thousands of flows come and go. A large number of flows can overlap in time, and a flow may go through multiple switches. As such, flow interference becomes more complex, and both output contending and path contending can happen. In this section, we will first make an observation about DCN flow characteristics, which motivates our CQRD approach. Then we present CQRD approach and how it addresses the challenges faced in real world DCNs.

4.1. DCN flow characteristics

In Fig. 3, we show an example of the traffic distribution which is derived from the characteristics of real operation data centers [25]. In this workload, about 90% flows are small/short flows, and very few (about 3%) of flows are giant flows. Based on the measurement results of real operational large data centers in [2,3], we make the following observation.

Observation 1. *Long bandwidth-greedy flows traverse a few of switch paths in a DCN switch, while most of the switch paths are transmitting short delay-sensitive flows.* There are always a large number (more than one thousand [3]) of active flows in DCN. However, very few of concurrent flows [2] are larger than 1MB. In a data center running data mining jobs, over 80% flows are less than 10 KB [6]. As a result, while some long bandwidth-greedy flows passing a few of *switch paths* (defined in Section 3), the majority of switch paths are transmitting short flows.

The above observation explains the significant flow interference in current DCNs. For example, a 24×24 aggregation switch has 24×23 switch paths in total (assuming no flow is destined to its coming input port). Assume that there are two giant flows passing two switch paths. Meanwhile, there may be hundreds of short delay-sensitive flows passing the other 550 switch paths. In a commodity OQ switch, 2×23 out of these 550 switch paths have the same output queues as the two giant flows, which interfere (and contend for paths) with the hundreds short flows going through the same 46 switch paths.

4.2. CQRD working scheme

We argue that the above observation calls for a queue management approach that is more fine-grained than OQ to reduce the probability of DCN flow interference. However, it is desirable to maintain a good balance between queue management granularity and the overhead/cost. In the ideal and most fine-grained approach, if we could reserve dedicated buffer and link capacity large enough for every single flow, the flow interference is entirely eliminated. However, the memory and link capacity needed for this ideal approach to deal with the large number of overlapping flows are prohibitive in practice. Moreover, it is hard to dynamically allocate physical resource according to the flow's various needs (e.g. buffer size or bandwidth), because these information is not available to the switch, without big modification to its packet parsing procedure or current network protocol stack.

We thus present crosspoint-queue with random-drop (CQRD), a simple queue management approach that is fine-grained enough to achieve desirable flow separation. The basic idea of CQRD is two-fold:

- Complete separation between flows on different switch paths, because a separate buffer is allocated to each switch path, and packets destined to the same output port but on different crosspoint buffers are scheduled in round-robin fashion.
- When a crosspoint queue is full, random-drop is used to alleviate the flow interference within the same switch path.

Fig. 4 shows the architecture of CQRD scheme. First, like original CQ switches shown in Fig. 1(b), CQRD allocates separate crosspoint buffers (XB) with the same capacity for each pair of inputs and outputs. Arriving packets are first stored in the crosspoint buffer and wait to be scheduled out by the output scheduler. CQRD uses round-robin (RR) scheduling for each output to schedule the packets. It ensures that each XB is fairly served. With separated buffers, flows in different

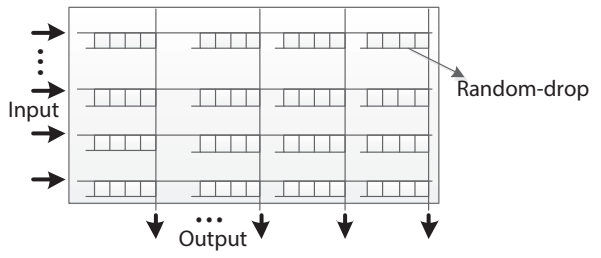


Fig. 4. CQRD overview.

switch paths will not contend for the queue with each other, although they might be destined to the same output. Also, with RR scheduling, flows going to the same output port is allocated with almost the same link bandwidth. This addresses the output contending but not path contending interference, and achieves *switch path separation*.

Second, when the corresponding crosspoint buffer is full, CQRD takes random-drop scheme upon packet arrival. Instead of simply dropping the tail (as in classic CQ), if the XB does not have enough space for the coming packet, CQRD will randomly choose a packet in this XB and drop it. A flow's packets will be more likely to be dropped if this flow occupies most of the XB, and vice versa. As such, if several small flows are contending for the same XB with a large flow, packets of those small flows still have a reasonable chance to get into the buffer even if the buffer are currently filled up by the large flows. That helps to alleviate interference within the same paths.

Next, we present a detailed description to CQRD working scheme. In practice, switches are always time slotted and the basic data unit processed in one time slot is called a *cell*, which is no larger than the minimum packet size. Consider an $N \times N$ CQRD switch with each crosspoint buffers (XB) of size L cells, and I_i and O_j denote the ith input and jth output respectively. Let XB_{ij} denotes the XB for I_i and O_j, and l_{ij} denotes the current length of XB_{ij} (*i.e.* the number of cells currently in XB_{ij}). CQRD buffers and schedules packets (*e.g.* ethernet frame for ethernet network) with variable sizes, which contains several cells. The start-of-packet (SoP) cell of this packet indicates the length of this packet (*i.e.* the number of cells) and the destination output port. All cells belonging to a packet are grouped and buffered/scheduled together. For each processing cycle in a time slot, CQRD scheme contains the following two phases:

- **Arrival phase.** For each input I_i, if there is a newly arriving packet (*i.e.* an SoP cell) destined for output O_j, CQRD first checks if the length of this packet is less than or equal to $L - l_{ij}$. If so, the SoP and rest following cells of this packet are stored in the tail of XB_{ij}, waiting to be scheduled. Otherwise, CQRD randomly picks one of the packets in XB_{ij} (including the newly coming one), then drop all cells of this packet. Packets are picked and dropped with probability proportional to their lengths (number of cells). If the newly coming packet is dropped, the arrival phase is finished. If not, CQRD checks $L - l_{ij}$ again, and repeats the previous random-drop procedure, until $L - l_{ij}$ is no less than the length of the coming packet or the coming packet is dropped. Note that although there can be several rounds of random-drop process in case of one packet arrival, it can be well pipelined in implementation, and incurs no extra processing latency. For example, if a packet with size of C cells come, it always consumes C time slots to insert the whole packet in the crosspoint buffer. Thus, there are C time slots (*i.e.* C rounds) for picking and dropping packets to make room in the buffer for the new packet, while the rest cells of this packet keep coming. Apparently, dropping C packets is enough to empty a buffer space of C cells, since the minimum packet size is larger than a cell.
- **Departure phase.** After arrival phase within the same time slot, for each output O_j, if all cells of the last packet have been scheduled out of the switch, the output round-robinly picks the next non-empty crosspoint buffer in its column. Then it schedules the SoP cell of the head packet in the selected buffer out of the switch, and continue to schedule the rest cells of this packet during the next time slots.

4.3. Influence to TCP performance

Based on above discussions, compared to original OQ switches, we can see that CQRD attempts to decrease the short flows' (the most majority in DCN) FCT, by alleviating the their contention for switch resources (*e.g.* buffer and link capacity) with long bandwidth-greedy flows. To understand why CQRD switches can greatly improve the performance of short flows, we present a simplified analysis below. Also, we briefly discuss why CQRD can reach that goal without sacrificing too much throughput of long flows. Our purpose is not to develop a fully complete model, but just to show some theoretic insights in our design. In the following analysis, we consider standard TCP flows without using other adaptive transport rate control schemes. At last, we briefly discuss how CQRD improves OQ's performance when combined with adaptive transport rate control schemes (*e.g.* DCTCP [2]).

4.3.1. Short flows

Many data center applications such as web search [26] or storage [27], generate delay-sensitive short flows, which contain very small amount of data (*e.g.* less than 100 KB [3]). Typically, these flows only consist of small number of TCP packets. The FCT of such a short TCP flow depends on two factors [28] (1) *how many packets of this flow being lost (lossrate)*, and (2) *the round-trip-time (RTT)*. Without loosing generality, we analyze a particular flow, and compare the average *lossrate* and *RTT* incurred by OQ and CQRD to it. We denote them as $Loss_o$, RTT_o (for OQ) and $Loss_c$, RTT_c (for CQRD) respectively. We further assume the *lossrate* and *RTT* incurred by the rest part during TCP transmission to be definite and the same both in OQ and CQRD analysis. To show how CQRD improves short flows' performance more clearly, we separately analyze the short flows *path contending* with long flows, and those *output contending but not path contending* with long flows.

Since packets are independently buffered and scheduled out by each output port both in OQ and CQRD switches, we perform our analysis on a particular output O without loosing generality. For simplicity, we assume all packets are with same fixed size below. However, our analysis can be easily extended to variable-size packets, by treating each packet to be

a number of fixed-size cells. We consider an OQ and a CQRD switch both with N input ports and N output ports, and every port has the same capacity. We assume that the OQ switch has an output buffer with size of $N \times L$ packets at each output. Accordingly, we assume that the CQRD switch has the same buffer resource in total, which means each crosspoint buffers (XB) with size of L packets. We assume during a certain period of time, there are s short flows ($F_{s1} \sim F_{ss}$) which come from inputs $s_1 \sim s_s$ ($I_{s1} \sim I_{ss}$) and are destined to the output O, where $s < N$ and $s_i = 1, \ldots, N (i = 1, \ldots, s)$. Also, there are l ($l > 1$)[2] long flows ($F_{l1} \sim F_{ll}$) coming from input $l_1 \sim l_l$ ($I_{l1} \sim I_{ll}$) destined to the same output O, where $l < N$ and $l_i = 1, \ldots, N (i = 1, \ldots, l)$.

(1) First, we analyze those short flows *output contending but not path contending* with long flows. We consider the performance of a particular short flow (F_{s1}) which comes from input s_1 (I_{s1}) and is destined to the output O. F_{s1} does not contend path with any long flow, which means $s_1 \neq l_i (i = 1, \ldots, l)$. We analyze the steady status after long flows in congestion avoidance status have filled up the output buffer or crosspoint buffers.[3] To begin with, we compare F_{s1}'s *RTT* incurred by OQ and CQRD. We assume the basic time unit to schedule a packet out of buffer being t, which is the same both in OQ and CQRD. For OQ switch, assume a new packet of F_{s1} arrives and there are L_o packets in the queue at this moment. According to the sawtooth characteristic of TCP congestion control [29], if a buffer is just rightly sized and neither underbuffered nor overbuffered, and if the number of long flows is not large, in the steady status, the average queue size of the buffer is half of the queue capacity. Thus we assume $\bar{L_o} = \frac{N \times L}{2}$, where $\bar{L_o}$ is the average size of the output buffer in the steady status. Because packets are scheduled in first-in-first-out (FIFO) manner, then in average, it will take $\bar{L_o} \times t = \frac{N \times L}{2} \times t$ for OQ switch to schedule this packet out of this output. On the contrary, for CQRD switch, assume there are L_c packets in the short flow's crosspoint buffer (XB_{s1O}) when a new packet of F_{s1} arrives. Because there is no long flow path contending with F_{s1} and XB_{s1O} has not been built up by long flows, then we assume $\bar{L_c} < \frac{L}{2}$, where $\bar{L_c}$ is the average size of XB_{s1O}. With round-robin scheduling, it will take $\bar{N'} \times \bar{L_c} \times t$ for CQRD to schedule this packet out of the output, where $\bar{N'}$ denotes the average number of XBs (corresponding to the output O) which have at least one packet in the buffer in each round[4] during the packet of F_{s1} is scheduled out. Obviously, $\bar{N'} \leq N$, thus

$$RTT_c = \bar{N'} \times \bar{L_c} \times t < N \times \frac{L}{2} \times t = \bar{L_o} \times t = RTT_o. \quad (1)$$

Next, we compare F_{s1}'s *loss rate* incurred by OQ and CQRD. For OQ switch, all packets destined to O are buffered in the same output buffer. This exactly matches the $G/D/1/K$[5] model in queuing theory [30], where the queue length $K = N \times L$. For this $G/D/1/K$ queue, packet loss rate grows extremely fast (*e.g.* exponential [31]) as the arriving rate (*i.e.* utilization) grows. Therefore, since there are l long flows saturating this output of OQ switch, in the $G/D/1/K$ queue, the arriving rate λ is very high (*e.g.* approximating 1). As a consequence, it leads to a dramatic increase of short flow's loss rate ($Loss_o$). On the other hand, in CQRD switch, the l long flows will only fill up the l corresponding crosspoint buffers ($XB_{l1O} - XB_{llO}$), but not fill the short flow's crosspoint buffer XB_{s1O}. Thus for XB_{s1O}, the arriving rate λ keeps low as only short flows arrive to this buffer. As a result, the short flow's loss rate ($Loss_c$) will not be increased and keeps very low.[6] Then, we have

$$Loss_c < Loss_o. \quad (2)$$

From Eqs. (2) and (3) we can see, CQRD incurs both a lower loss rate and a lower RTT than OQ for short flows *output contending but not path contending* with long flows. Thus, the FCT of such short flows is reduced in CQRD compared to OQ.

(2) Second, we analyze those short flows *path contending* with long flows. Similarly, we consider the performance of a particular short flow (F_{s2}) which comes from input s_2 (I_{s2}) and is destined to the output O. We assume F_{s2} contends the same switch path with l' long flows. For OQ switch, the RTT and loss rate of F_{s2} is the same as the former short flow F_{s1} (output contending but not path contending), which we have already analyzed before. For CQRD switch, it is different in this case. The crosspoint buffer XB_{s2} is also going to be filled by long flows. Then we perform another analysis on CQRD for F_{s2}. First, we discuss F_{s2}'s *RTT*. For CQRD switch, assume a new packet (denoted as P_{s2}) of F_{s2} arrives during the steady status (XB_{s1O} has been filled up) and is inserted in the tail. Assume there are L_c packets in XB_{s1O} at this moment, then according to [29], we assume $\bar{L_c} = \frac{L}{2}$, where $\bar{L_c}$ is average size of the crosspoint buffer in the steady status. Then with round-robin scheduling, it will take $\bar{N'} \times (\bar{L_c} - \bar{D}) \times t$ for CQRD to schedule this packet out of the output, where $\bar{N'}$ denotes the average number of XBs (corresponding to the output O) which have at least one packet in each round during P_{s2} is scheduled out of the output. Obviously, $\bar{N'} \leq N$. $\bar{D}$ denotes the average number of packets which are queued ahead of P_{s2} and are dropped due to random-drop scheme, during CQRD scheduling P_{s2} out of the output. Apparently, $\bar{D} \geq 0$, thus

$$RTT_c = \bar{N'} \times (\bar{L_c} - \bar{D}) \times t \leq N \times \frac{L}{2} \times t = RTT_o. \quad (3)$$

[2] We omit the analysis for the condition where $l \leq 1$ here. Both OQ and CQRD behave well because the buffer can hardly build up if there are less than two long flow destined to O.

[3] Due to the buffer-greediness [2] of long TCP flows (without adaptive rate control schemes), the output buffer or crosspoint buffers will be definitely filled up if $l > 1$.

[4] For a certain XB, a round refers to the moment that right after this XB was just selected by its output and a packet of the XB is scheduled out.

[5] G stands for a general distribution of the arrival process. D stands for a definite distribution of the serving process, where the serving rate is definite to be packet/t. 1 means that there is only one server (the output port in this case) for each output queue. K stands for the queue length.

[6] Because there is no long flow traversing from I_{s1} to O, we assume that the arriving rate of only short flows to XB_{s1O} is very low. Note that this assumption is not hold for some extreme cases such as incast scenario [27], where too many concurrent small flows contend the same output. However, such cases (*i.e.* link saturated only by short flows) happen rarely in DCN [3]. Moreover, the incast problem, which many prior works [32,33] focus on, is beyond the scope of which CQRD aims to address in this paper.

Then, we analyze the F_{s2}'s *loss rate*. In the path contending case, the arriving rate λ of CQRD also grows higher due to the existence of long flows, which leads to a relative high loss rate as well. In this case, it is difficult to get a precise comparison between $Loss_o$ and $Loss_c$ from theoretical formula using queuing model, because it is hard to abstract an accurate math model of real TCP arriving statistic process. However, we can draw a rough analysis between two approaches' loss rate. Assuming $PLoss_o$ to be the overall packet loss rate (including all flows destined to O) of the output buffer in OQ switch, then $Loss_o = PLoss_o$ because all packets of different flows share the same loss rate in the output buffer. Similarly, we assume $PLoss_c$ to be the overall packet loss rate of the crosspoint buffer XB_{s2} in CQRD, and assume L_{s2} to be the number of F_{s2}'s packets in XB_{s2}. Then we have

$$Loss_c = PLoss_c \times \frac{L_{s2}}{L}, \tag{4}$$

where $\frac{L_{s2}}{L}$ is the probability of dropping F_{s2}'s packets while XB_{s2} is full. Therefore, we can draw the following result using above equation

$$Loss_c = PLoss_c \times \frac{L_{s2}}{L} < Loss_o = PLoss_o, \tag{5}$$

if

$$\frac{PLoss_c}{PLoss_o} < \frac{L}{L_{s2}}. \tag{6}$$

Intuitively, $\frac{L}{L_{s2}}$ is very large because F_{s2} will occupy only small portion of the buffer while contending with other long flows. However, $PLoss_c$ can be close to $PLoss_o$ under certain arriving and serving rate on XB_{s2} in CQRD and the output buffer in OQ. From the above two equations, we can see that $Loss_c$ is very likely to be smaller than $Loss_o$. To summarize, CQRD incurs a lower RTT and a very likely lower loss rate than OQ for short flows *path contending* with long flows. Thus, the FCT of such short flows is reduced in CQRD compared to OQ.

4.3.2. Long flows

From above analysis, we can see that CQRD improves short flows' FCT by alleviating interference through separate buffers and rand-drop schemes. CQRD reaches that by trading off the ability of buffer statistical multiplexing. Therefore, it raises the concern of sacrificing too much throughput of large flows, because of less capability to accept bursts without statistic multiplexing. In this section, we analyze that how CQRD can maintain high throughput of long flows with moderate buffer size. Similarly as before, we consider an CQRD switch both with N input ports and N output ports, and every port has the same capacity. We assume there are l long flows destined to the same output O.[7] Then we discuss under which condition that the output O's capacity can be saturated, which means that l long flows have reached optimal aggregate throughput.[8]

We consider the most extreme case first. Assuming the l long flows sharing the same switch path (*i.e.* coming from the same input), and l is very small (*e.g.* 2). Then all these long flows must contend for the same XB and other $N-1$ XBs remain to be idle. According to the widely used rule-of-thumb [34], this requires a buffer size of bandwidth-delay product (BDP) to ensure output capacity being saturated. Even in this most extreme case, CQRD is able to provide full aggregate throughput for long flows, by using certain amount of buffer resource through modern application-specific integrated circuit chips (ASIC) technologies [35] . Specifically, a typical BDP in the current production data center is about 25 KB (1 Gbps $\times$ 200 μs) [36]. For a 48 $\times$ 48 switch, CQRD only needs about 58MB memory in total even in the worst case, and current commodity switches already have large enough memory to meet this demand [37].

The former extreme case happens rarely in practice, and usually long flows are coming from several input ports (say l_n ports). Thus, CQRD only needs the sum size of l_n XBs to be approximately BDP, instead of each XB has the size of BDP in the extreme case before. And this could guarantee an 100% utilization of output O (*i.e.* long flows have reached full aggregate throughput). Moreover, according to the tiny buffer model in recent studies [38], an XB needs only to be several KBs to ensure an 80–90% output utilization even in the extreme case. That further indicates less memory in total could be used in a CQRD switch with fixed port number, or a much larger CQRD switch can be implemented using the latest ASIC technology.

4.3.3. Transport with adaptive rate control

Unlike standard TCP perceiving congestion by packet loss, transport with adaptive rate control (*e.g.* DCTCP [2]) uses explicit feedback (*e.g.* ECN [9]) from switches to detect congestion. After the number of packets in the switch buffer exceeds a threshold, the switch will tag congestion information on traversing packets, which lowers down the sending rate before the switch buffer is full and prevent packet loss. Therefore, these methods can perceive congestion earlier and keep switch buffer very low, which enhances performance. Assuming these transport schemes with adaptive rate control have already been deployed in DCN (although they face some deployment hurdle, see Section 2.1), compared to original OQ, CQRD still can offer performance improvement combining with these methods.

To be more specific, in OQ switch, before adaptive rate control starts to work, the buffer can be quickly built up by long flows and exceeds the ECN threshold. This triggers ECN feedback to throttle long flows' sending rate, but also causes short flows' congestion window to be cut down, because they share the same buffer and the packets of short flows are also going to be marked with congestion signal. This greatly increases their FCT. Moreover, the RTT of short flows is also going to be increased due to interference of other long flows. It is because that, although these transport methods try to keep the switch buffer near empty, it is hard to accurately control the rate of long flows [6] due to the bursty traffic in DCN. Packets of short flows still may queue behind a considerable number of long flows' packets due to a burst of these long flows, which increases their queuing delay in OQ switch.

[7] We do not explicitly consider short flows in this section, because they are always very short (less than 100 KB [3]) and infect little on long flows' throughput.

[8] Fairness among different flows is beyond the scope of the problem that CQRD addresses.

On the contrary in CQRD, for short flows output contending but not path contending with long flows, similarly to the former analysis (Section 4.3.1), they will have much less chance to be throttled due to long flows, because separate buffers are used.[9] Also, their queuing delay in CQRD is lower. As for short flows path contending with long flows, our former random-drop scheme may provide little help because of adaptive rate control, and packets are seldom dropped due to buffer overflow. However, in this condition, we accordingly offers an *random-mark* scheme for CQRD besides random-drop. *Random-mark* shares the same philosophy of random-drop, and works similarly as random-drop described in Section 4.2. Specifically, *random-mark* works as follows: whenever a new packet comes and the queue length exceeds the marking threshold, instead of simply marking the newly coming packet with congestion signal, CQRD randomly picks up one of the packet in the buffer (including the newly coming one) with the probability according to the packet's size, and marks it with congestion signal. This gives a much less chance for small flows to be throttled because they occupy little portion of the buffer. Thus, CQRD improves these small flows' FCT. Note that *random-mark* increases the chance of decreasing small flows' FCT statistically, but it cannot guarantee that all small flow can harvest the gain. While on average small flows FCT can be decreased, some unlucky small flows may still be marked (even many times), which leads to a potential bad tail FCT.

5. Implementation

It typically takes a rather long time to implement a new switch prototype chip due to the long cycle (e.g. years) of application-specific integrated circuit chip (ASIC) design [39]. In addition, the cost of ASIC design and implementation will be very high without mass production. For above reasons, we have not been able to implement a CQRD switch with ASIC prototype by now. However, we find that many current commodity switches could be configured to be a small scale CQRD switch, without any modification to the hardware or software. In this section, we first introduce our implementation of a small scale CQRD switch through configuring a commodity switch, and then we discuss the implementation of a large scale CQRD switch through ASICs.

5.1. CQRD by configuring commodity switch

As Fig. 5 shows, we implement an 8 × 8 CQRD switch through configuring a Broadcom BCM56842 switch [19]. BCM56842 is a 32 × 32 10 Gbps switch, with 9MB physical shared buffer. The shared buffer is logically separated into 32 output buffers, which forms a logical OQ switch. For each output, the output buffer can be configured into up to eight priority queues (PrQ), which can be logically separated. And we utilize this priority queues to implement an 8 × 8 CQRD switch, where we only use eight of the 32 ports in BCM56842.

Specifically, first, we enable eight priority queues for each output port, and configure them with the same size

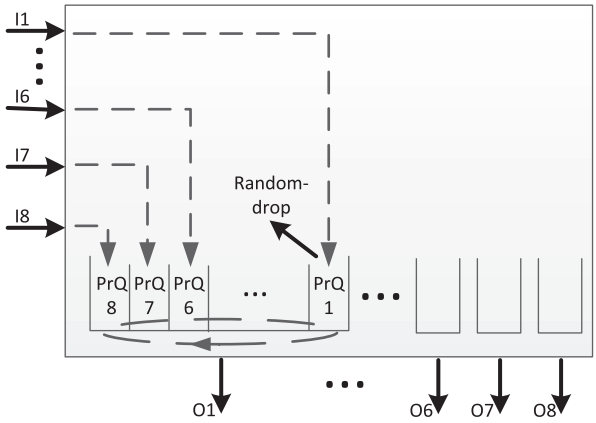


Fig. 5. Implementing CQRD through configuring commodity switch.

(i.e. 9 MB/8^2 ≈ 140 KB). Also, we configure the scheduling manner between the eight priority queues to be weighted round-robin (WRR), with each queue having the same weight. Second, we configure the switch to classify incoming packets according to their input ports, through ingress content aware processor (ICAP) configuration. Third, packets with different input ports are mapped into different priority queues. After that, for each output, packets from different inputs will only be buffered into its own queue. So far, we have implemented a round-robin CQ switch, with each priority queue to be a crosspoint buffer. At last, we emulate the CQRD's random-drop scheme through enable random early detection (RED) [40] configuration. Note that there may be a little difference between RED and our random-drop design. RED starts to randomly drop subsequent packets after the queue size exceeds a certain threshold. However, statistically, RED is also able to proportionally drop packets of different flows according to how many they come into the buffer, which emulates the CQRD's random-drop scheme.

5.2. ASIC implementation

Crosspoint queue with round-robin scheduling and random-drop are relatively simple to implement in ASIC using widely used hardware primitives. And from the above section we see that, current switching ASICs already have the capability of supporting the CQRD's underlying techniques, showing the feasibility of CQRD's simple implementation. The engineering details of hardware ASIC design (*e.g.* including register-transfer level design and system, timing and logic verification *etc.*) are not our contribution and beyond the scope of the paper. There is a rich body of literature [11,41–43] in this domain.

Considering the need of high speed processing, switches commonly use ASIC on-chip memory as packet buffers. As we discussed in Section 4.3.2, CQRD needs certain size of crosspoint buffer (larger than a BDP in the worst case) to ensure long flows' throughput saturating port capacity. That raises a possible concern: can current ASICs offer memory large enough to implement a CQRD switch with practical size? Similar to [10], we conduct a back-of-envelope of calculation below. We assume a high-performance switching

[9] ECN independently works for each XB in CQRD.

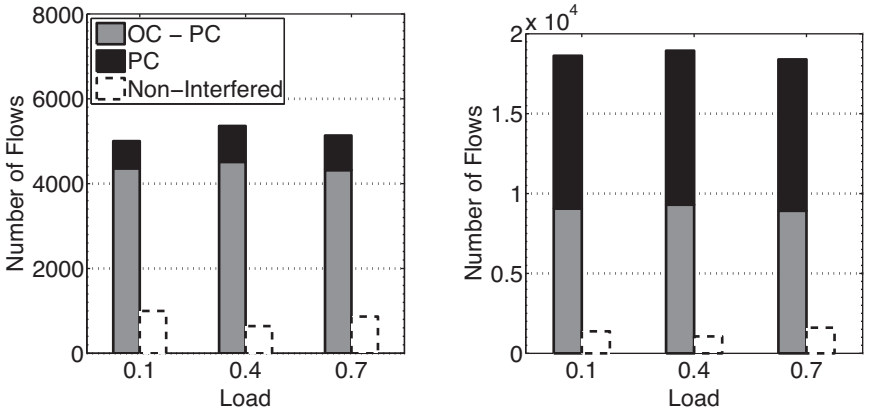


Fig. 6. The number of flows that are output contending but not path contending (OC-PC), path contending (PC), and not interfered (non-interfered) with giant flows in the experiments.

ASIC design using 18 mm $\times$ 18 mm die size in total. Considering it consists of 70% packet memory and 30% switching and memory-decoder logic die area [44], and an SRAM cell size of 0.06 μm^2 [35], then the ASIC can contain about 470MB on-chip memory in total. According to our former analysis (Section 4.3.2), considering the worst case, we can easily build a 128 $\times$ 128 CQRD switch with each XB's size larger than a typical BDP (25 KB) in current production data centers. Moreover, even considering future production data centers with a larger BDP (100 Gbps $\times$ 20 μs = 250KB [6]), a 48 $\times$ 48 CQRD switch can be built using modern ASIC technology, with the size of each XB larger than a BDP.

6. Performance evaluation

To evaluate CQRD's performance, we implement and simulate CQRD in NS2 [22], and compare its performance with two other switch-based approaches, OQ and HCF. Also, we compare CQRD with original classic CQ switch. Moreover, we simulate four hybrid approaches: the most well-known transport layer method DCTCP [2] combined with CQRD, CQ, OQ, HCF, respectively. The evaluation is based on the following five experiments. In *experiment 1*, we simulate a single DCN aggregation/core switch. In *experiment 2*, a classic multi-stage DCN switching topology with 480 servers has been simulated. In *experiment 3*, we discuss the scenario of incremental deployment. In *experiment 4*, we study the impact of different buffer sizes on CQRD's performance. And in *experiment 5*, we show that a combination of CQRD and transport layer methods can further improve performance.

We first describe the traffic workloads, simulation parameters, and performance metrics, followed by the detailed results of the five experiments. Since giant flows are the triggers of DCN's performance degradation [2], we are mainly interested in the flows interfered by the giant flows. We will show that OQ, HCF, CQ and CQRD perform differently because they differ in queue management granularity and how they deal with the output contending and path contending flows.

6.1. Experiment setup

Traffic workloads. We derive our workloads from the characteristics of real operation data center traffic [3,25] as shown earlier in Fig. 3. During the simulations, source and destination of the flows are randomly chosen among all the switch ports (in experiment 1) or among all hosts (in experiments 2–5). The inter-arrival time of the flows obeys the log-normal distribution [3]. We scale the flow inter-arrival time to simulate the moderate (0.1), heavy (0.4), and extreme (0.7) loads in the network.

Fig. 6 shows the ratio of flows that are output contending but not path contending (OC-PC), path contending (PC), and not interfered (non-interfered) with giant flows (as defined in Section 4) in all the simulations. This figure shows that the giant flows interfere with more than 80% of all flows although they contribute to less than 3% of all flows. As shown in Fig. 6(b), giant flows interfere with more flows in multi-stage DCN than in a single switch. This is because flows pass longer paths in this topology and more flows tend to intersect in ToR and aggregation switches.

Simulation parameters. All the parameters of HCF are set as the recommended in HCF paper [12] (same as Section 3.3). During all the simulations, we use the SACK [23] version of TCP, which has already been widely implemented in most of the operating systems. The initial window size and min retransmission-time-out (RTO) are set to be 4 and 200 μs respectively, which is typical in high-speed DCN [6].

Performance metrics. Following the convention in [6], we consider two main performance metrics—flow completion time for short flows and goodput for large flows. These two metrics reflect the key performance of these two kinds of flows.

6.2. Experiment 1: single aggregation/core switch

Setup. Our first experiment is to simulate the aggregation and core switches with huge amount of flows passing by, and compare different approaches in this situation. We simulate a 24 $\times$ 24 switch (a typical DCN switch) shown in

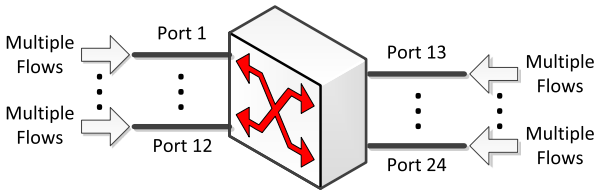
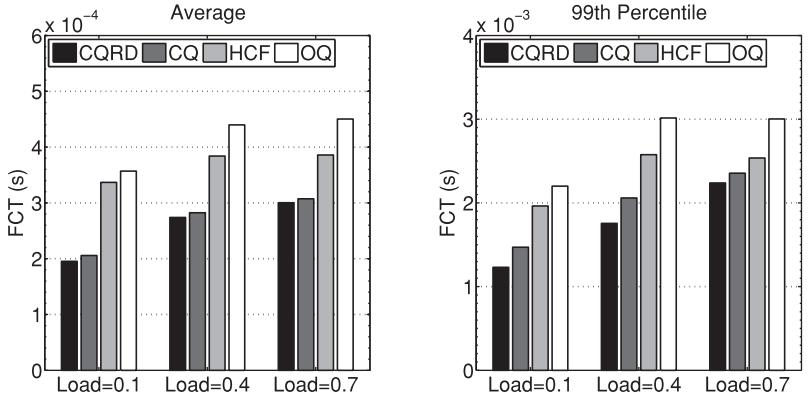


Fig. 7. Experiment 1: a 24 × 24 switch used in simulations.

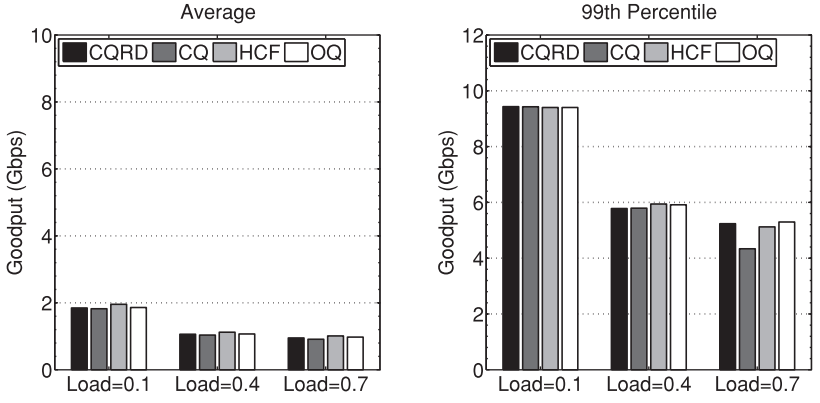
Fig. 7. Each port has a 10 Gbps link rate and 4 μs link delay. This generates a ∼16 μs end-to-end round-trip time (RTT) without queueing, which is realistic in the real DCN environment according to [6]. In all three schemes, we assume the on-chip memory for packet buffers are 5MB, which can be easily implemented by commodity FPGAs. As a result, each output buffer has ∼210 KB in OQ, which conforms to the convention [6]; HP and LP queue each has ∼105 KB in HCF; each crosspoint queue buffer has ∼9 KB in CQRD.

During the simulation, 6000 flows are generated to 24 ports. The workload is as described before.

Results. Fig. 8 shows the overall FCT of all short flows and goodput of all long flows at various loads of 10%, 40% and 70%, which represent moderate, heavy and extreme loads. We observe that, with any load, CQRD has a much better overall FCT of short flows, which contribute to about 90% of all flows according to Fig. 3. The average of CQRD's FCT is more than 25% lower than HCF and OQ schemes at all the loads. Also, the 99th percentile of CQRD's FCT is still more than 10% better than HCF and OQ schemes. As we can see, a simple CQ without random-drop scheme can reach a much better performance than HCF and OQ. And CQRD can further improve CQ's performance by ∼5–7% and ∼6–10% in average and 99th percentile of all short flows respectively. As the load grows higher (e.g. ≥ 40%), more and more concurrent flows come into the switch and flow interference becomes severe. In this situation, it gets harder to provide good flow separation and the performance of these three schemes tends to get similar at the tail. However, CQRD still performs the best as we can see in Fig. 8(a). This is because that there are several giant flows with sizes of more than 1MB among all the flows. In OQ switch, they occupy most of the buffer resource and lead to packets of many other short flows dropped, which greatly increase the overall FCT. Although HCF sets two separate queues at each output and tries to fairly serve all the flows using hash and credit schemes, it does not provide enough buffer separation for different flows. Many flows are output contending but not path contending, and in HCF they still have to contend for the same buffer resource. This results in the packet losses

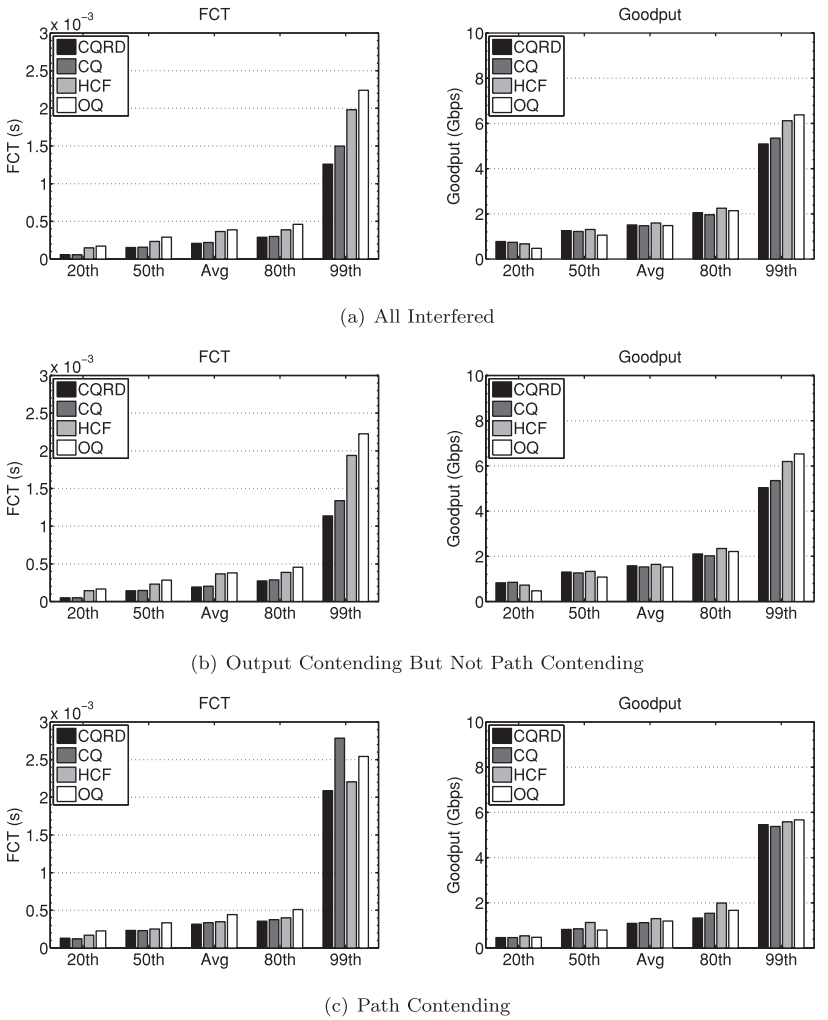


(a) FCT of short flows



(b) Goodput of long flows

Fig. 8. Experiment 2: overall FCT of all short flows and goodput of all long flows at various loads.



(a) All Interfered

(b) Output Contending But Not Path Contending

(c) Path Contending

Fig. 9. Experiment 1: average and various percentile FCT of all short flows and goodput of all large flows **interfered** by the giant flows at moderate load.

for short flows that are output contending but not path contending with giant flows, which is avoided in CQRD.

As for flows larger than 100 KB, CQRD has almost the same overall performance (i.e., goodput) as HCF and OQ (Fig. 8(b)). These results show that the small size of separated buffer in CQRD will not significantly impair the large flows' goodput.

So far, we have shown that the overall performance of all short flows in CQRD greatly outperforms the other two approaches, at the cost of only a minor goodput decrease of very few large flows. This is because CQRD alleviates the performance degradation of the flows interfered by a few giant flows. As Fig. 6(a) shows, although less than 3% of all flows are giant flows, they interfere with more than 80% of all flows. And most of the interfered flows are in different switch paths with the giant flows. CQRD uses separated buffers and random-drop schemes respectively, to guarantee the performance of those interfered flows having the same outputs or paths with giant flows. We now investigate all the flows (including giant flows) interfered by giant flows and show how CQRD improves their performance. Those flows which are not interfered by giant flows perform well and similarly in all the three schemes, so we omit the results here due to page limit.

In Fig. 9 we show the average and 20th to 99th percentile FCT of short flows and goodput of large flows which are interfered by giant flows, at a typical moderate load. We have also done these simulations at other loads from 10% to 80% and the comparison results are similar. We only present the results at typical moderate load (10%) here due to page limit.

As Fig. 9(a) shows, for all interfered short flows, FCT in CQRD is much lower than CQ, HCF and OQ switches. On the other hand, the goodput of all interfered large flows is only a little lower than the other two approaches. For example, the average goodput in CQRD is only about 5% lower than HCF's. These results show that CQRD is able to deal with flow interference much better than the other two approaches.

Recall that, as discussed in Section 4, CQ provides (1) complete separation for *output contending (excluding those path contending) flows*, by using separated buffers and RR scheduling, and (2) interference alleviation *for path contending flows* by using random-drop when queue is full. We now show how well CQRD performs for these two types of flows.

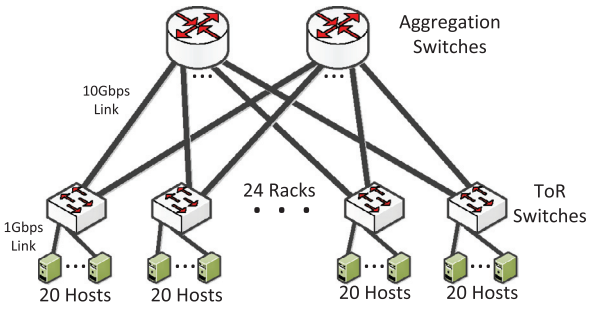


Fig. 10. Experiment 2: a multi-stage DCN topology used in simulations.

Fig. 9(b) shows the results for flows output contending (but not path contending) with giant flows. CQRD greatly exceeds others' performance for these flows just as we expect. These flows contribute to the majority of all interfered flows as shown in Fig. 6(a), therefore, their performance dominates the performance of all interfered flows. Although these small flows are also separated from giant flows in CQ as the same as in CQRD, they still interfere with other small and large (but not giant) flows with the same switch paths. Thus, with random-drop schemes, CQRD increases the chance of small flows to be stored in the buffer at the trade of dropping a little more packets in large flows. That makes CQRD perform better than CQ in these small flows output contending (but not path contending) with giant flows.

As for flows path contending with giant flows, CQRD successfully reduces the FCT of short flows significantly (left part of Fig. 9(c)), by random-drop scheme. This is evidenced by the difference between CQRD and CQ. CQRD has a lower FCT than CQ, which shows CQRD successfully alleviates the interference among flows path contending with giant flows. However, without random-drop scheme, small flows are interfered by giant flows with same switch paths in pure CQ. And that leads to a bad performance (especially in 99th percentile) of CQ, as shown in Fig. 9(c).

6.3. Experiment 2: multi-stage DCN switching fabric

Setup. In the second experiment, we simulate a two layer multi-root topology with full bisection bandwidth (see Fig. 10). This topology is one of the most commonly used topologies in large-scale DCN [25,45]. The network consists of 480 end hosts allocated in 24 racks, which are interconnected by two aggregation switches and 24 ToR switches. Aggregation switches have 5MB memory for packet buffer as before, and twenty-four 10 Gbps ports connected to each ToR switch. Each ToR switch has less memory with size of 4MB, and two 10 Gbps ports connected with two aggregation switches, and twenty 1 Gbps ports connected to 20 hosts respectively. These are typical parameters of ToR switches [46]. The delay of each link is 2 μs, which means a $\sim$16 μs end-to-end RTT across racks and a $\sim$8 μs RTT within a rack. We use Equal Cost Multi-path (ECMP) [47], which is the de facto network load-balancing scheme [48] in modern data centers. CQRD (CQ, HCF, OQ) are used in all switches in the topology when simulating CQRD (CQ, HCF, OQ)'s performance. During the experiment, 20000 flows are generated according to the work load described in Section 6.1.

Results. First, we show the overall FCT of all short flows and goodput of all long flows at various loads. As shown in Fig. 11(a), CQRD has the best FCT performance at various loads. The average FCT of all short flows in CQRD are about 30–50% lower than HCF and OQ, and 6–8% lower than CQ at moderate (10%) and heavy (40%) loads. And the 99th percentile of CQRD's FCT is also much lower than others at these loads. CQRD performs more better than other schemes in this two-layer DCN environment than the single switch before. It is because much more flows generated (20,000 here and 6000 before) and more flow interference occur among all the ToR and Agg switches.

While load grows to extreme (70%), the performance gap of all the methods becomes narrower. As we stated before, flow interference becomes severe as the load grows higher. In this situation, it becomes difficult to provide good flow separation and the performance of these three schemes are similar. However, CQRD still performs the best at the average FCT as we can see in Fig. 11(a). In addition, Fig. 11(b) shows that, in CQRD, the average and 99th percentile goodput of all large flows perform almost the same as in HCF and OQ. These results show that a DCN built with CQRD switches has a much better overall performance, only at the cost of a minor goodput decrease of very small portion large flows. Using separated buffer and random-drop scheme, CQRD not only provides flow separation for a single switch environment as shown in the former subsection, but also performs well in a multi-stage switch data center network.

Next, we also dive into the performance of the flows interfered by giant flows. Those flows which are not interfered by giant flows are omitted here similar to experiment 1, because they almost have the same low delay and high goodput. Results at various loads are similar, so we only present the results at moderate load here.

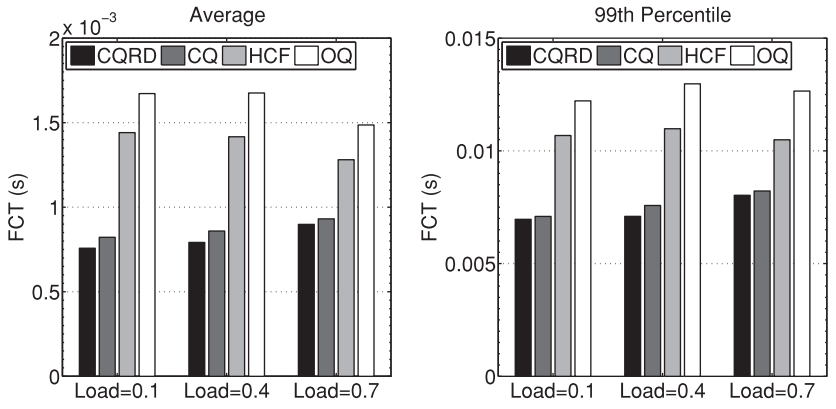
Fig. 12(a) shows the overall performance of all the interfered flows at moderate load. The average and 99th percentile FCT in CQRD are 30% and 48% lower than HCF and OQ. CQRD also has a much lower FCT for all interfered short flows at other percentiles. On the other hand, the goodput of large flows in CQRD is almost the same as other methods. This shows that in data center, for a multi-stage switch network built by CQRD, the flow interference caused by giant flows can be significantly alleviated.

We repeat the same simulation as experiment 1, to reveal the performance of the flows that are output contending but not path contending with giant flows, in multi-stage switch DCN environment. As we can see in Fig. 12(b), CQRD greatly improves the performance of these kind of flows by switch path separation. CQRD performs similar to CQ with these flows, which uses the same crosspoint buffer scheme.

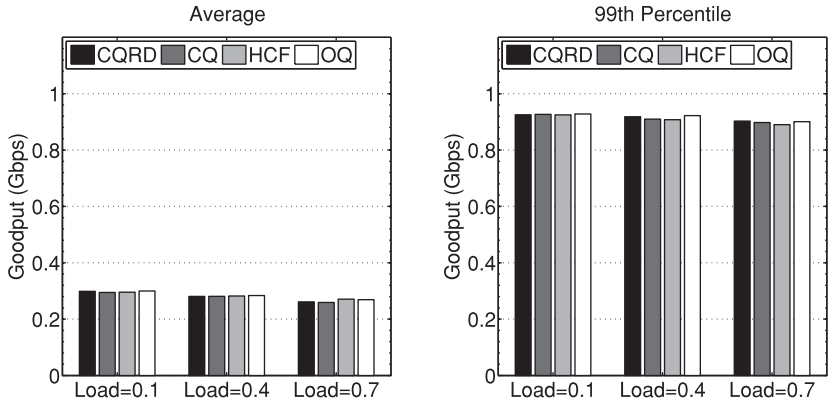
As for flows path contending with giant flows shown in Fig. 12(c), CQRD has the best FCT for average and all percentile. And CQRD significantly outperforms CQ with these kind of flows, thanks to the random-drop scheme instead of tail-drop.

6.4. Experiment 3: incremental deployment

We have shown in the former experiment that, by fully deploying CQRD in all DCN switches, flow interference could



(a) FCT of short flows



(b) Goodput of long flows

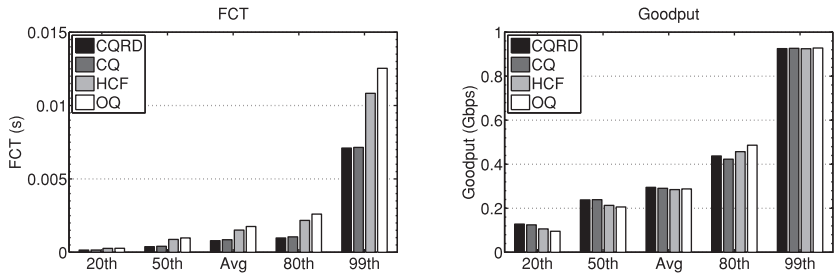
Fig. 11. Experiment 2: overall FCT of all short flows and goodput of all long flows at various loads.

be alleviated and the overall DCN performance can be greatly improved. However, it usually takes cost and time to replace all the switches in an operational data center. As such, DCN operators often prefer to incrementally deploy a new solution. For instance, they may incrementally deploy new switches by partition/clusters or tier-level. In this experiment, we will show that CQRD can also offer substantial performance improvement for DCN while being incrementally deployed.

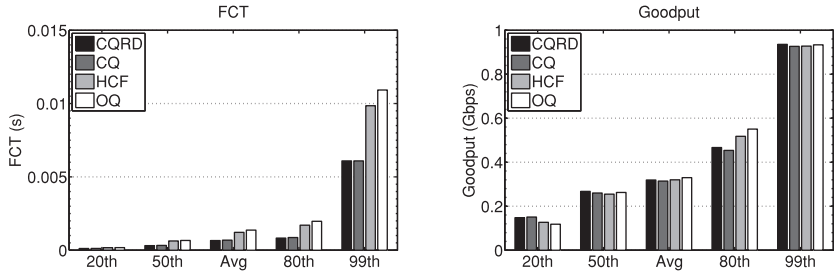
Setup. We simulate the same DCN environment (Fig. 10) as Section 6.3, except incrementally deploying CQRD/CQ/HCF switches instead of full deployment. We consider two scenarios of incremental deployment: (1) By partition/clusters. We assume four neighbor ToR switches constituting a small partition/cluster and deploy CQRD/CQ/HCF switches by different number of partition/clusters. (2) By tier-level. We assume all switches in one certain tier (ToR or Agg) are replaced and the rest are remained to be OQ switches. The traffic load and all other parameters are set as the same as experiment 2.

Results. Fig. 13 shows the results at the same moderate, heavy and extreme loads used in the former experiment. For the first scenario (by partition/clusters), as we can see in Fig. 13, even with only part of partition/clusters deployed, CQRD and CQ still substantially outperforms HCF and OQ in average FCT of all short flows. Meanwhile, the goodput of all large flows remains almost the same as HCF and OQ. For instance, with only 50% of partition/clusters (12 ToRs) switches deployed, the average FCT of all short flows in CQRD can be ~10–24% lower than HCF and OQ methods respectively, at different loads. CQRD and CQ behaves similar while small part of partition/clusters are deployed. As more partition/clusters are deployed, CQRD substantially outperforms CQ.

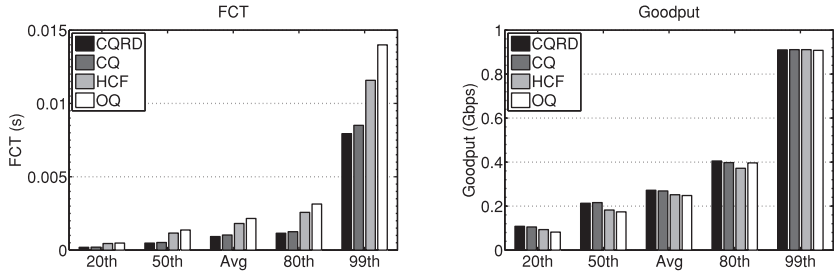
For the second incremental deployment scenario (by tier-level), CQRD also provides reasonable performance improvements compare to other methods. As the right most points show in Fig. 13, while all ToR switches are deployed, CQRD performs much better than other methods, with 30–50% lower FCT of short flows than HCF and OQ, and 4–8% lower than CQ at various loads. The goodput of large flows remains to be similar as well. While only deploying in all Agg switches (the left most points in Fig. 13), the four methods behave similar, and CQRD offers less performance benefit. It is because that only replacing Agg switches, flows still may be interfered with each other in ToR switches. And most flow interference occur in ToR switches. This also leads CQRD to behave similar to CQ. However, CQRD and CQ still perform better than HCF and OQ switches, which benefits from output contending flows' separation through crosspoint buffer.



(a) All Interfered



(b) Output Contending But Not Path Contending



(c) Path Contending

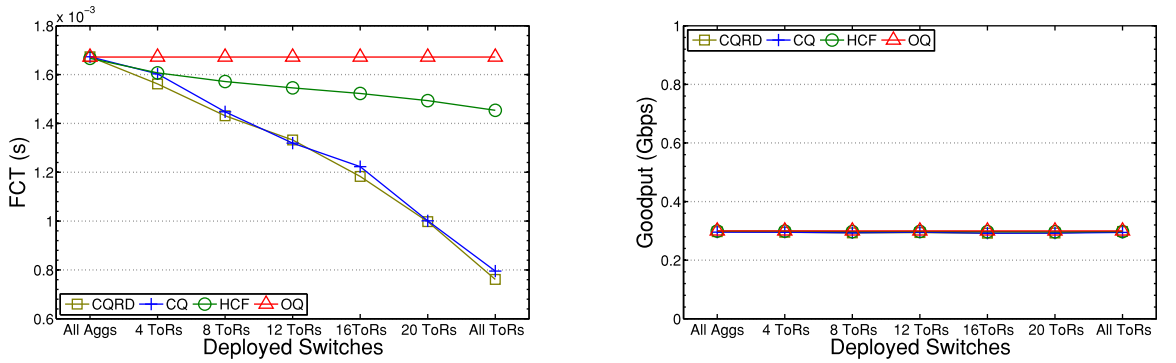
Fig. 12. Experiment 2: average and various percentile FCT of all short flows and goodput of all large flows **interfered** by the giant flows at moderate load.

6.5. *Experiment 4: impact of different buffer sizes to CQRD*

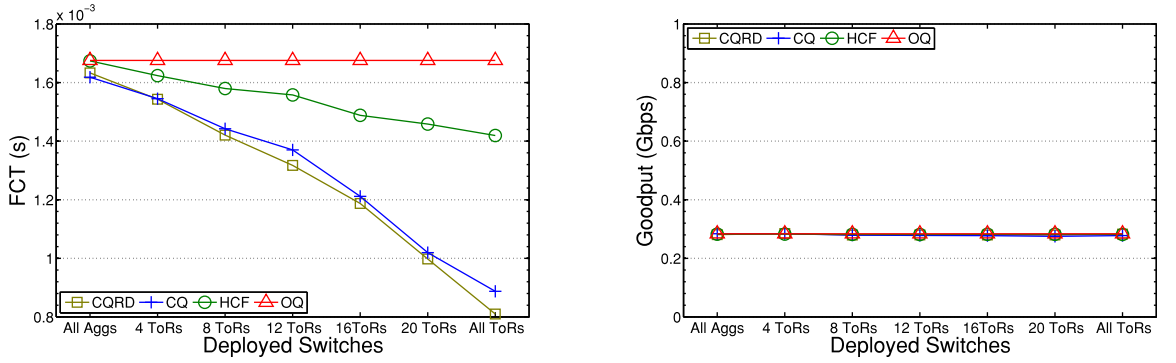
As analyzed before, CQRD use separated crosspoint queues to offer separation for flows with different switch paths. In addition, with random-drop scheme, CQRD could greatly alleviate interference between different flows and improve the performance. However, this improvement is reached through sacrificing OQ switch's ability of statistically multiplexing buffer resource for flows coming from different input ports. That may raise the concern that CQRD will consume too much buffer resource. In this experiment, we study the performance of CQRD using different buffer sizes. We will show that CQRD's performance improvement can be achieved with a very reasonable amount of buffer resource consumed, which verifies our former analysis (Section 4.3.2).

Setup. We also use the same DCN environment (Fig. 10) as Section 6.3, except using different buffer sizes instead of a fixed one for all the switches. For simplicity, we assume all aggregation and ToR switches having the same buffer size, and each crosspoint buffer has equal size. We study CQRD's performance as the buffer size changes. Input traffic are generated according to the traffic model derived from real DCN, as the same as experiment 2.

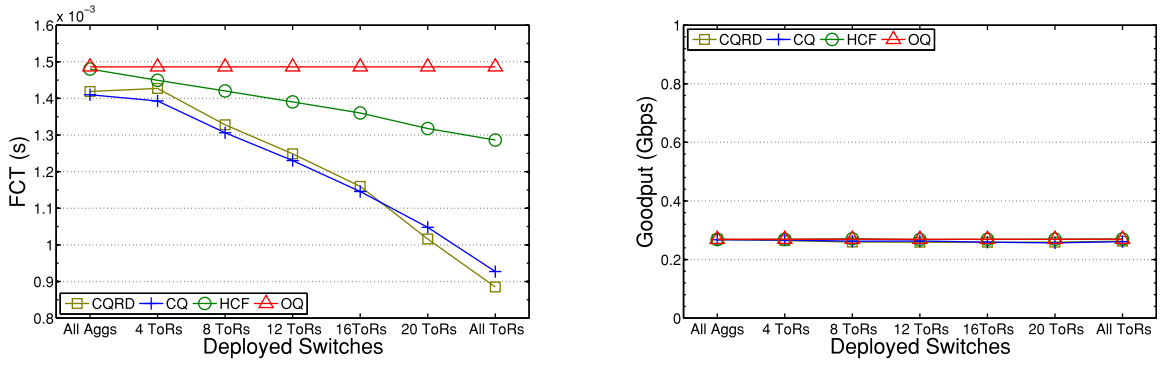
Results. Fig. 14 shows the average and 99th percentile FCT of all short flows and goodput of all large flows, while using different buffer size for CQRD switch, at various loads. The buffer size in the figure refers to the *total size* of those crosspoint buffers corresponding to the same output port in CQRD, which is the counterpart of the output queue size in OQ switch. It varies from 30 KB to 400 KB. As we can see, both CQRD's FCT and goodput performance remains stable while the buffer size varies from 400 KB to about 120KB, at moderate, heavy and extreme load. When the buffer size becomes smaller than 90 KB, CQRD's performance starts to slightly degrade at heavy and extreme load, but at moderate load it still performs reasonably well. Performance of CQRD sharply degrades while the buffer size is reduced to be 30 KB. At this size, each crosspoint queue in a 24-port CQRD switch can store only approximately one maximum-size packet, which is too small for CQRD to reach a good performance. From this experiment we can see that CQRD could reach a stable good performance at a minimum buffer size of about 120 KB for each output port, and increasing buffer size does



(a) Load=0.1



(b) Load=0.4



(c) Load=0.7

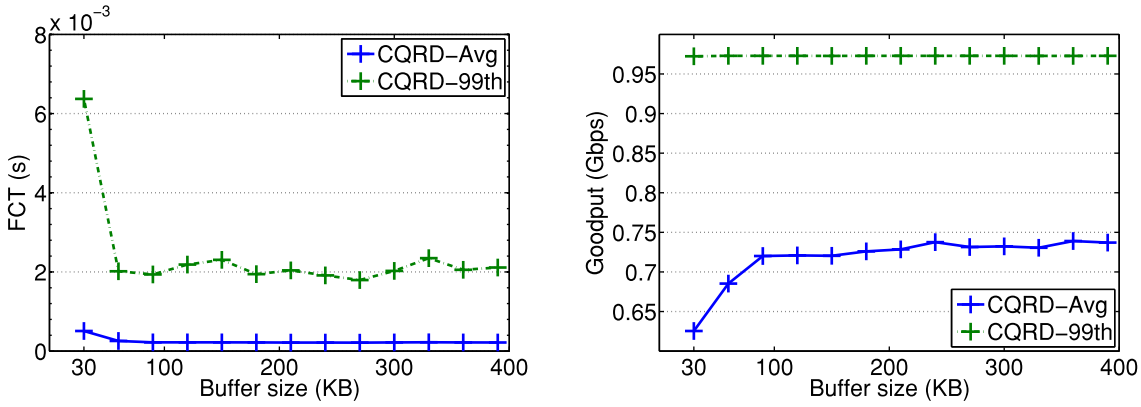
Fig. 13. Experiment 3: average FCT of all short flows and goodput of all large flows, while incrementally deploying ToR or Agg switches by CQRD/CQ/HCF, at various loads.

not obviously benefit the performance. Such a small buffer size (e.g. 120 KB) for each switch output port is totally within the capacity of commodity switches [21,46], which shows the feasibility of the CQRD solution.

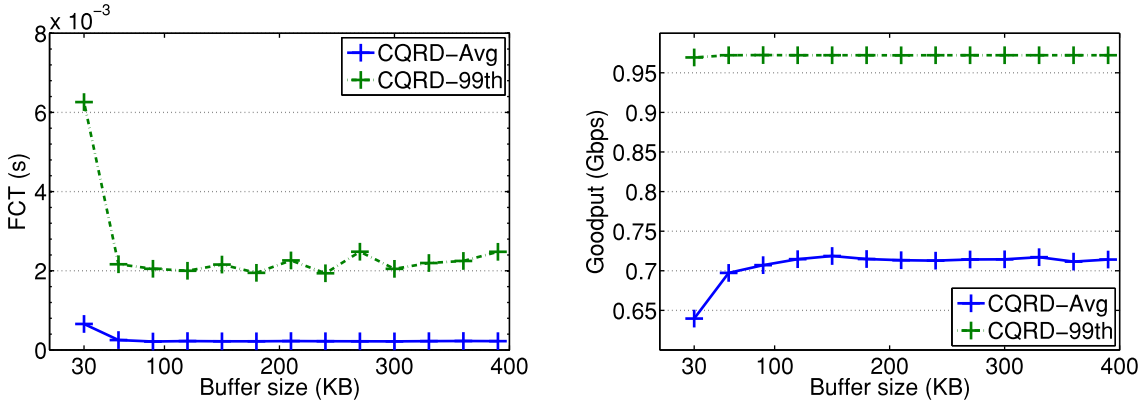
6.6. Experiment 5: CQRD with DCTCP

Through former analysis and experiments, we have shown that CQRD can improve the performance of DCN with TCP flows. Moreover, as discussed in Section 4.3.3, CQRD could also offer performance improvement to DCN using transport with adaptive rate control (e.g. DCTCP). In this section, we carry experiments to show how CQRD behaves under this environment.

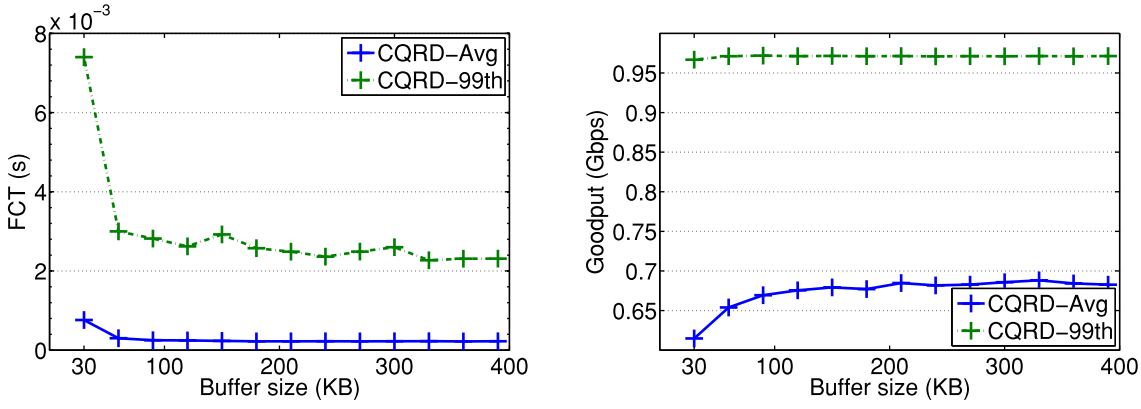
Setup. We use the same DCN environment (Fig. 10) and same input workload as Section 6.3. Instead of the TCP SACK used before, in this experiment, we use the well-known DCTCP as the transport layer protocol. The weight of new



(a) Load=0.1



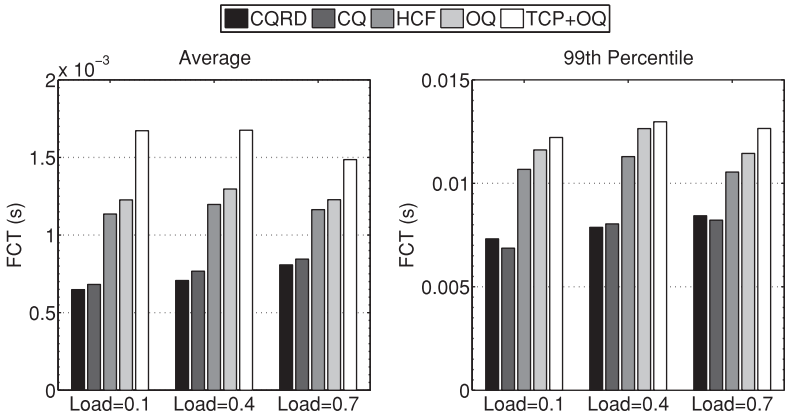
(b) Load=0.4



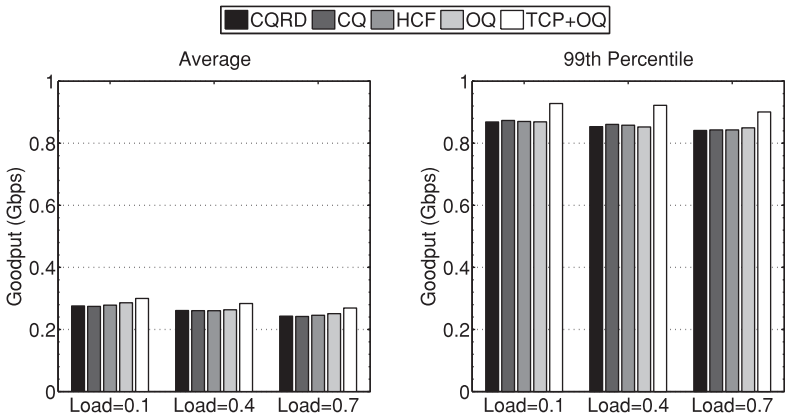
(c) Load=0.7

Fig. 14. Experiment 4: average and 99th percentile FCT of all short flows and goodput of all large flows, while using different buffer sizes for CQRD switch, at various loads. The buffer size in the figure refers to the *total size* of those crosspoint buffers corresponding to the same output port in CQRD. Each crosspoint buffer has the same size. .

sample in DCTCP is set to be 6.25E−3, according to the suggestion in DCTCP paper [2]. Accordingly, we set the ECN marking threshold of the four compared methods (CQRD, CQ, HCF and OQ) to be the same, which is 40% of the queue's capacity. CQRD and CQ switches start to mark packets with congestion experiencing (CE) flag if the corresponding crosspoint queue size has exceeded the threshold. CQRD uses the random-mark scheme (see Section 4.3.3) designed for DCTCP environment, while CQ simply marks the subsequent packets. HCF [12] has no special design for DCTCP environment,



(a) FCT of short flows



(b) Goodput of long flows

Fig. 15. Experiment 5: overall FCT of all short flows and goodput of all long flows at various loads, with DCTCP. Legend CQRD/CQ/HCF/OQ in the figure denotes the hybrid of DCTCP and CQRD/CQ/HCF/OQ, and legend TCP+OQ denotes the hybrid of TCP SACK and OQ.

so we simply make it mark subsequent packets with CE flag, when the queue (HQ or LQ) that the packets are going to be buffered into exceeds the threshold. We compare the four methods using DCTCP, and the OQ method using original TCP SACK.

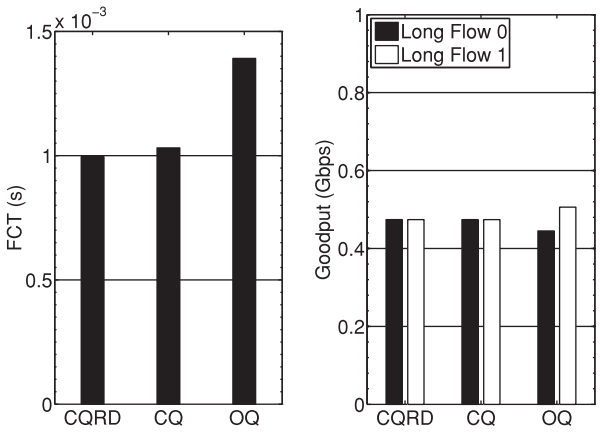
Results. Fig. 15 shows the overall FCT of all short flows and goodput of all long flows at various loads. Thanks to the adaptive rate control schemes, unlike original TCP which always fills up buffers, DCTCP detects congestion before buffer overflow and control its sending rate, thus keeps the switch buffer length in a much lower state. Therefore, simple OQ with DCTCP can greatly outperform the original TCP SACK for short flows, as shown in Fig. 15(a), by reducing the queueing delay and also the loss rate of short flows. CQRD is able to further improve the performance, by separating most long flows from the short flows using crosspoint buffer and random-mark. As we can see in Fig. 15(a), CQRD has reduced the average short flows' FCT by ~30–40% from HCF and OQ, and ~5–10% from CQ. Also, the 99th percentile of short flows in CQRD is much lower than HCF and OQ. However, in this condition, the 99th percentile of short flows is similar in CQRD and CQ, and even a little higher in CQRD. It is because that some unlucky short flows have been throttled more than once by the random-mark scheme in CQRD, which leads to a higher 99th percentile.

Similar to the original TCP SACK, the goodput of long DCTCP flows remains to be similar as well. Fig. 15(b) shows that OQ with original TCP has a slight higher goodput for long flows than methods using DCTCP. It is because that, DCTCP is hard to accurately control the rate of long flows [6] to exactly fully utilize the capacity without overflowing the buffer. In our case, DCTCP sacrifices some link utilization to ensure low buffer state, which verifies our former analysis.

7. Testbed experiment

In this section, we evaluate CQRD's performance using a simple experiment in a small-scale testbed, to verify our implementation as well as the effectiveness of CQRD to alleviate flow interference in real environment.

Testbed setup. We build a small testbed with a single switch of eight ports, and eight servers, each of which is connected to one of the eight ports. The original switch is a Broadcom BCM56842 switch [19], which is a logical OQ switch. And during the experiment, we configure the switch to be a CQRD/CQ switch according to the method introduced



(a) FCT of short flows (b) Goodput of long flows

Fig. 16. Testbed experiment: average FCT of all short flows and goodput of each long flow.

in Section 5. The capacity of original switch port is 10 Gbps, but each port works in the rate of 1 Gbps in our testbed, limited by the servers' network interface card (NIC) speed. Each output buffer in the switch has a size of ∼1.13MB, which means each crosspoint buffer in CQRD/CQ has a size of ∼140 KB. Standard TCP SACK without ECN is used throughout the experiment. We compare the performance of CQRD, CQ and OQ in this testbed experiment. Because HCF cannot be implemented by configuring existing commodity switches, we do not compare with it in this experiment.

Input traffic. In CQRD/CQ/OQ switch, flows are potentially interfered with each other only when they are destined to the same output, and each output port buffers packets separately and schedules them independently. So we can evaluate the switch's performance by picking out a certain output port and evaluate the performance of the flows destined to it. In this experiment, we pick the zeroth port without loosing generality. Specifically, assuming H_i denotes the server connected to the ith switch port, during the experiment, we generate two long bandwidth-greedy flows from H_1, H_2 to H_0, using Iperf [49] to send long TCP flows to H_0 at the maximum rate allowed. Also, on H_1–H_7, we write a simple TCP socket program in each server to randomly generate short flows with size of 10 KB to H_0. The inter-arrival time of short flows from each server is randomly distributed within 500 ms. We conduct the experiment for 20 s, and about 600 short flows have been generated in total.

Results. Fig. 16 shows the average FCT of all short flows and the goodput of the two long flows (denoted as long flow 0 and 1). Similar to the former simulation results, in the testbed experiment, CQRD outperforms than CQ, and performs much better than OQ. The average FCT of short flows in CQRD is about 5% lower than CQ, and 28% lower than OQ. As for long flows, similar to the simulation results in Section 3, they have almost the same goodput in CQRD, CQ and OQ. The aggregate goodput of these two long flows have saturated the bandwidth (1 Gbps) in all the three methods, which confirms our analysis in Section 4.3. Moreover, as analyzed in Section 3, CQRD/CQ provides a better fairness among these two long flows than OQ. Although it is only a small testbed, the results in this experiment have verified our implementation in Section 5 and further validated the former simulation results.

8. Conclusion

In this paper, we advocate that modern DCN flow characteristics call for fine-grained queue management in switches. Along this direction, we propose a switched based solution called CQRD to address DCN flow interference problem, without any modification to end hosts or any coordination among different switches. CQRD is simple to implement, and is more fine-grained than traditional OQ scheme and current state-of-arts HCF scheme. It only requires some minor changes to the buffering and scheduling scheme in DCN switches. We have implemented an 8 × 8 logical CQRD switch based on configuring existing commodity switch, which validates the simplicity and feasibility of CQRD's implementation. And we show through NS2 simulations, that when flow interference happens, CQRD can improve the flow completion time of short and delay-sensitive flows by up to ∼50%, at the cost of only a minor goodput decrease of large flows. Moreover, we show that a partial deployment of CQRD is still able to greatly improve the DCN performance. Furthermore, we show that a hybrid of CQRD and transport layer methods can offer better performance over transport layer only methods. Finally, through small-scale testbed experiments, we have verified our CQRD implementation and the former simulation results.

Acknowledgments

This work has been supported in part by the National High Technology Research and Development Program of China (863 program) under grant no. 2013AA013302, the National Key Basic Research Program of China (973 program) under grant no. 2013CB329105 and the State Key Program of National Natural Science of China under grant no. 61233007.

References

[1] G. Chen, D. Pei, Y. Zhao, CQRD: a switch-based approach to flow interference in data center networks, in: 2014 IEEE 39th Conference on Local Computer Networks (LCN), IEEE, 2014, pp. 107–115.
[2] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), Proc. SIGCOMM 40 (4) (2010) 63–74.
[3] T. Benson, A. Akella, D.A. Maltz, Network traffic characteristics of data centers in the wild, in: Proc. of IMC, ACM, 2010, pp. 267–280.
[4] C. Wilson, H. Ballani, T. Karagiannis, A. Rowtron, Better never than late: meeting deadlines in datacenter networks, in: Proc. of SIGCOMM, 41, ACM, 2011, pp. 50–61.
[5] C.-Y. Hong, M. Caesar, P. Godfrey, Finishing flows quickly with preemptive scheduling, Proc. SIGCOMM 42 (4) (2012) 127–138.
[6] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, PFABRIC: minimal near-optimal datacenter transport, in: Proc. of SIGCOMM, ACM, 2013, pp. 435–446.
[7] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, M. Yasuda, Less is more: trading a little bandwidth for ultra-low latency in the data center, in: Proc. of NSDI, USENIX Association, 2012, p. 19.
[8] B. Vamanan, J. Hasan, T. Vijaykumar, Deadline-aware datacenter TCP (D2TCP), Proc. SIGCOMM 42 (4) (2012) 115–126.
[9] K. Ramakrishnan, S. Floyd, A Proposal to Add Explicit Congestion Notification (ECN) to IP, Technical Report, RFC 2481, January 1999.
[10] Y. Kanizo, D. Hay, I. Keslassy, The crosspoint-queued switch, in: INFOCOM, 2009, pp. 729–737.
[11] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, et al., Recommendations on Queue Management and Congestion Avoidance in the Internet, RFC 2309 (1998).

[12] A. Shpiner, I. Keslassy, G. Bracha, E. Dagan, O. Iny, E. Soha, A switch-based approach to throughput collapse and starvation in data centers, Comput. Netw. 56 (14) (2012) 3333–3346.
[13] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, D. Ongaro, G. Parulkar, et al., The case for RAMCloud, Commun. ACM 54 (7) (2011) 121–130.
[14] M. Beck, M. Kagan, Performance evaluation of the RDMA over ethernet (ROCE) standard in enterprise data centers infrastructure, in: Proceedings of the Third Workshop on Data Center-Converged and Virtual Ethernet Switching, International Teletraffic Congress, 2011, pp. 9–15.
[15] M. Shreedhar, G. Varghese, Efficient fair queuing using deficit round-robin, IEEE/ACM Trans. Netw. 4 (3) (1996) 375–385.
[16] P.E. McKenney, Stochastic fairness queueing, in: Proc. of INFOCOM, IEEE, 1990, pp. 733–740.
[17] N. McKeown, The ISLIP scheduling algorithm for input-queued switches, IEEE/ACM Trans. Netw. 7 (2) (1999) 188–201.
[18] F. Abel, C. Minkenberg, R.P. Luijten, M. Gusat, I. Iliadis, A four-terabit packet switch supporting long round-trip times, IEEE Micro 23 (1) (2003) 10–24.
[19] Broadcom BCM56840_PLUS Switching Technology. https://www.broadcom.com/collateral/pb/56840_PLUS-PB00-R.pdf, 2015.
[20] N. Ek, IEEE 802.1 P, Q-QoS on the MAC Level (April 24, 1999) 0003–0006.
[21] Cisco Catalyst 4500-x Series Fixed 10 Gigabit Ethernet Aggregation Switch Data Sheet. http://www.cisco.com/en/US/prod/collateral/switches/ps10902/ps12332/data_sheet_c78-696791.html, 2014.
[22] K. Fall, K. Varadhan, The Network Simulator (ns-2). URL: http://www.isi.edu/nsnam/ns, 2007.
[23] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgment Options, Technical Report, RFc 2018, October 1996.
[24] P. Goli, V. Kumar, Performance of a crosspoint buffered ATM switch fabric, in: INFOCOM, IEEE, 1992, pp. 426–435.
[25] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, in: Proc. of SIGCOMM, 39, ACM, 2009, pp. 51–62.
[26] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, C. Yan, Speeding up distributed request-response workflows, in: ACM SIGCOMM Computer Communication Review, 43, ACM, 2013, pp. 219–230.
[27] Y. Chen, R. Griffith, J. Liu, R.H. Katz, A.D. Joseph, Understanding TCP incast throughput collapse in datacenter networks, in: Proceedings of the First ACM Workshop on Research on Enterprise Networking, ACM, 2009, pp. 73–82.
[28] M. Mellia, H. Zhang, I. Stoica, TCP model for short lived flows, IEEE Commun. Lett. 6 (2) (2002) 85–87.
[29] G. Appenzeller, I. Keslassy, N. McKeown, Sizing router buffers, in: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, in: SIGCOMM '04, ACM, New York, NY, USA, 2004, pp. 281–292, doi:10.1145/1015467.1015499.
[30] L. Kleinrock, Queueing Systems, Volume 1: Theory, 1975.
[31] F. Yegenoglu, B. Jabbari, Performance evaluation of MMPP/D/1/K queues for aggregate ATM traffic models, in: INFOCOM'93. Proceedings. Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE, IEEE, 1993, pp. 1314–1319.
[32] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D.G. Andersen, G.R. Ganger, G.A. Gibson, B. Mueller, Safe and effective fine-grained TCP retransmissions for datacenter communication, in: Proc. of SIGCOMM, 39, ACM, 2009, pp. 303–314.
[33] H. Wu, Z. Feng, C. Guo, Y. Zhang, ICTCP: incast congestion control for TCP in data center networks, in: Proc. of CoNEXT, ACM, 2010, p. 13.
[34] C. Villamizar, C. Song, High performance TCP in ANSNET, ACM SIGCOMM Comput. Commun. Rev. 24 (5) (1994) 45–60.
[35] S.I. Association, et al., Test and Test Equipment, International Technology Roadmap for Semiconductors (ITRS) 2011 Update [OL], 2013.
[36] D. Zats, T. Das, P. Mohan, D. Borthakur, R. Katz, Detail: reducing the flow completion time tail in datacenter networks, ACM SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 139–150.
[37] Arista 7048—A Gigabit Ethernet Data Center Switch. http://www.arista.com/assets/data/pdf/Datasheets/7048_A_Datasheet.pdf, 2015.
[38] A. Vishwanath, V. Sivaraman, M. Thottan, Perspectives on router buffer sizing: recent results and open problems, ACM SIGCOMM Comput. Commun. Rev. 39 (2) (2009) 34–39.
[39] D. Chinnery, K. Keutzer, Closing the Gap between ASIC & Custom: Tools and Techniques for High-performance ASIC Design, Springer Science & Business Media, 2002.
[40] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Trans. Netw. 1 (4) (1993) 397–413.
[41] A. Silberschatz, B. Ozden, J. Bruno, H. Saran, Early Fair Drop Buffer Management Method. URL https://www.google.com.hk/patents/US6556578, US Patent 6,556,578, 2003.
[42] G. Aybay, P. Ferolito, Method and Apparatus for Fair and Efficient Scheduling of Variable-size Data Packets in an Input-buffered Multipoint Switch. URL https://www.google.com.hk/patents/US6044061, US Patent 6,044,061, 2000.
[43] F. Diaz, J. Stanley, Buffered Crosspoint Matrix for an Asynchronous Transfer Mode Switch and Method of Operation, https://www.google.com.hk/patents/US5537400, US Patent 5,537,400, 1996.
[44] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, N. Chrysos, Variable packet size buffered crossbar (CICQ) switches, in: 2004 IEEE International Conference on Communications, vol. 2, IEEE, 2004, pp. 1090–1096.
[45] C. D. C. Infrastructure, 2.5 Design Guide, 2007.
[46] IBM rack switch. http://www-03.ibm.com/systems/networking/switches/rack.html, 2014.
[47] C. Hopps, Analysis of an Equal-cost Multi-path Algorithm, RFC 2992, 2000.
[48] A. Dixit, P. Prakash, Y.C. Hu, R.R. Kompella, On the impact of packet spraying in data center networks, in: Proc. of INFOCOM, IEEE, 2013, pp. 2130–2138.
[49] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, K. Gibbs, IPERF: The TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/Projects, 2005.

Guo Chen received his B.S. degree from Wuhan University (China) in 2011. He is currently a Ph.D. student in the Department of Computer Science at Tsinghua University, Beijing, China. His current research interests include data center networking and high performance switching.

Youjian Zhao received the B.S. degree from Tsinghua University in 1991, and M.S. degree in Shenyang Institute of Computing Technology, Chinese Academy of Sciences in 1995. He received the Ph.D. degree in Computer Science from Northeastern University, China in 1999. He is now a professor of CS Department at Tsinghua University. His research mainly focuses on high speed internet architecture, switching and routing, and high-speed network equipment.

Dan Pei received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, in 1997 and 2000, respectively, and the Ph.D. degree from the University of California, Los Angeles (UCLA), in 2005. He is now an associate professor at Tsinghua University. His current research interests are management and improvement of the performance and security of the networked services, through big data analytics with feedback loop. Right now he is focusing on improving the Mobile Internet performance over Wi-Fi networks and data center networks.