Flash: Joint Flow Scheduling and Congestion Control in Data Center Networks

Chengxi Gao, *Member, IEEE*, Shuhui Chu, Hong Xu, *Senior Member, IEEE*,
Minxian Xu, *Member, IEEE*, Kejiang Ye, *Member, IEEE*, and Chengzhong Xu, *Fellow, IEEE*

Abstract—Flow scheduling and congestion control are two important techniques to reduce flow completion time in data center networks. While existing works largely treat them independently, the interactions between flow scheduling and congestion control are in general overlooked which leads to sub-optimal solutions, especially given that the link capacity is increasing faster than the switch port buffer size. In this paper, we present *Flash*, a simple yet effective scheme that integrates scheduling and congestion control. Specifically, *Flash* puts forward a congestion-aware scheduling scheme to determine the priority of flows based on the latest network congestion extent and the flow's bytes sent. Besides, *Flash* proposes a priority-based packet dropping scheme in switch port buffers and implements a priority-aware congestion control scheme. Experiment results show that *Flash* has superior performance: (1) it has 35.8% lower tail latency than PIAS and performs similar with pFabric in a 10G network without knowing the flow size, (2) in 100G networks with shallow buffers, the information agnostic *Flash* has 6.8% lower average FCT than the information-aware pFabric, (3) it outperforms pFabric by 13.5% in FCT if flow size is also known to *Flash*.

Index Terms—Data center networking; Flow scheduling; Congestion control

1 Introduction

With the widespread utilization of cloud computing and its improvements from VM to containers and serverless [1], [2], [3], [4], various services and applications are continuing to move to the cloud, thus it is increasingly important and challenging to build data center networks (DCN) to satisfy their diverse and stringent performance requirements [5], [6], [7], [8], [9], [10]. The widely used distributed machine learning system is a recent example [11], [12]. These applications usually generate lots of network flows, and it is crucial to minimize the flow completion time (FCT) to ensure low latency.

Congestion control and flow scheduling are two important techniques to reduce the FCT. Congestion control decides how to adjust the sending rate according to network congestion, and scheduling determines the priorities of competing flows. In recent years, there is extensive literature on these two subjects, respectively. As for congestion control, DCTCP [5] marks packets in switch buffers and estimates network congestion extent at TCP senders for congestion window size update. L2DCT [8] adjusts the TCP sender's congestion window size according to both the network congestion extent and the flow bytes sent to realize the Least Attained Service (LAS) scheduling. MQ-ECN [13] and TCN [14] are two congestion control schemes in multi-service multi-queue scenarios. D2TCP [15] is a deadline-aware scheme that adjusts congestion window size in order to meet flow deadlines. As for scheduling, pFabric [9] performs shortest remaining size first scheduling and implements priority-based scheduling and dropping scheme. PIAS [16] proposes an information-agnostic solution which implements a multi-level feedback queue using priority queues in switches. Karuna [17] is a mixed-flow solution which sets minimum bandwidth to deadline flows to meet their deadlines just in time and leaves the remaining bandwidth to deadline-free flows. Aemon [18] extends PIAS by considering the deadline agnostic scenario, and proposes a new flow urgency based mixed-flow scheduling scheme.

However, existing DCN works consider flow scheduling and congestion control *separately* without taking their interactions into account. For example DCTCP simply assumes FIFO as the scheduling discipline. L2DCT incorporates flow priority to congestion control, but does not use priority scheduling at switches. Thus there lacks an in-depth systematic analysis of the impact of these two mechanisms on each other, possibly leading to suboptimal solutions.

Specifically, the limitations manifest in two ways:

First, scheduling without considering congestion control issues. Almost all existing scheduling designs leverage shortest job first (like PIAS) or shortest remaining size first scheduling (like pFabric), which means they use flow size information as the only criterion to determine the priority of a flow. However, given a multi-tier topology of the data center network and the contention among flows, a flow could be blocked or congested by any link along its path if there are higher priority flows on it. Thus, setting the flow priority according to flow size *only* could result in inferior performance (more in Sec. 2.1). A better solution should estimate the network congestion extent and determine the priority of a flow accordingly in addition to flow size.

- *C. Gao, M. Xu and K. Ye are with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China. E-mail: chengxi.gao@siat.ac.cn, mx.xu@siat.ac.cn, kj.ye@siat.ac.cn.*
- *S. Chu is with the Department of Computer and Information Science, University of Macau, Macau 999078, China. E-mail: sh.chu@siat.ac.cn.*
- *H. Xu is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China. E-mail: hongxu@cuhk.edu.hk*
- *C. Xu is with the State Key Lab of IoTSC, Faculty of Science and Technology, University of Macau, Macau 999078, China. E-mail: czxu@um.edu.mo.*

The corresponding author is Kejiang Ye.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3129511, IEEE Transactions on Cloud Computing

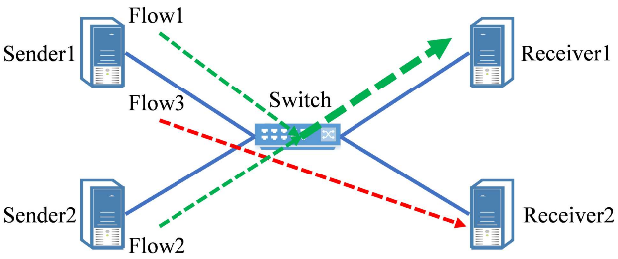


Fig. 1. Topology for case studies

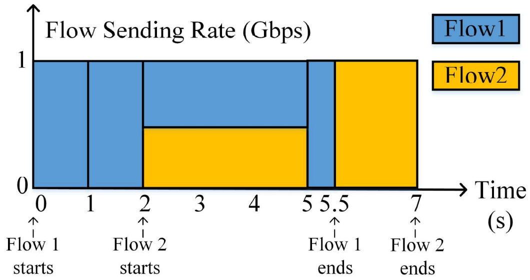


Fig. 2. pFabric: shortest remaining size first scheduling

Second, congestion control without considering scheduling does not work well either. When updating the congestion window size, short latency-sensitive flows are throttled with the same extent as long elephant flows, which restrains the performance of short flows. Besides, switches should also incorporate scheduling into priority-aware congestion control at senders to achieve better performance (more in Sec. 2.2). Packet dropping should also take flow priority into consideration for congestion control purposes. This is due to the fact that the buffer size per port per Gbps is dropping [19]. Given that data center network is bursty in nature, packet drop will become a more severe problem. Traditional dropping mechanisms do not distinguish short flows from long flows. As short flows are latency-sensitive, dropping their packets will have a far more severe impact.

In this paper, we propose *Flash*, a simple yet effective solution that integrates flow scheduling and congestion control in order to solve the problems aforementioned. For scheduling module, *Flash* proposes a congestion aware scheme to determine the priority of flows based on the estimated network congestion extent it experiences and the flow bytes sent. For congestion control module, *Flash* puts forth a priority-based packet dropping scheme and implements a priority-aware scheme to adjust the congestion window size at a TCP sender.

The main **contributions** of this work are as follows.

- First, we identify the drawbacks of treating scheduling and congestion control separately, thus motivating the need of jointly considering the two.
- We propose *Flash* to re-design the flow scheduling and congestion control schemes from the perspective of each other. As discussed, *Flash* adjusts flow priority according to the network congestion extent in addition to bytes sent, and enforces priority-based congestion window size update and packet dropping schemes.
- Finally, we conduct extensive experiments to evaluate the performance of *Flash*. The results show that *Flash* delivers outstanding performance: (1) it has 35.8% lower tail latency than PIAS and performs similar with pFabric in a 10G network without knowing the flow size, (2) in 100G networks with shallow buffers, the information agnostic *Flash* has 6.8% lower average FCT than the information-aware pFabric, (3) it outperforms pFabric by 13.5% in FCT if flow size is also known to *Flash*.

The rest of this paper is organized as follows. We introduce the motivation for *Flash* in Sec. 2. We present the design of *Flash* in Sec. 3. Extensive experiments are conducted to evaluate the performance of *Flash* in Sec. 4. We summarize related work in Sec. 5 and conclude the paper in Sec. 6.

2 Motivation

In this section, we use both case studies and experiments to show the drawbacks of treating scheduling and congestion control separately, thus motivating the need of jointly considering two schemes.

2.1 Scheduling without considering congestion control

In this section, we use a case study to identify the problem of current flow scheduling schemes without the consideration of congestion control.

Here, we utilize the network topology shown in Fig. 1. In this example, Sender 1 is sending Flow 1 (4 Gb) to Receiver 1 and Sender 2 is sending Flow 2 (3 Gb) to Receiver 1. All link capacity is 1 Gbps and Flow 1 starts at time 0 while Flow 2 starts 2 seconds later. Assume that at time 2s (the same time as Flow 2 starts), Sender 1 sends Flow 3 to Receiver 2, and Flow 3 has higher priority than Flow 1 and Flow 2, and occupies 0.5 Gbps bandwidth for 3 seconds. Therefore, during the 3 seconds when Flow 3 is alive, Flow 1 can only use 0.5 Gbps bandwidth, which means Flow 1 is congested by other flows. Here, we use bandwidth occupation by higher priority flows to denote the network congestion that the flow is experiencing.

Currently, the best scheduling scheme is pFabric, which adopts the *Shortest remaining size first* scheduling scheme, and the scheduling result of Flow 1 and Flow 2 is shown in Fig. 2 (we exclude Flow 3 in the figures since it is always occupying 0.5 Gbps bandwidth when it is alive). As we can see, when Flow 2 starts, Flow 1 has transmitted 2 Gb and has 2 Gb remaining, and Flow 2 has 3 Gb remaining. Therefore, according to the shortest remaining size first discipline, Flow 1 is scheduled first. Since Flow 3 occupies 0.5 Gbps, Flow 1 can only get 0.5 Gbps, thus Flow 2 gets another 0.5 Gbps bandwidth (due to the bandwidth competition and sharing on the link between switch and Receiver 1). After 3 seconds when Flow 3 finishes, Flow 1 has 0.5 Gb remaining and Flow 2 has 1.5 Gb remaining, then Flow 1 is scheduled first using 1Gbps bandwidth. After 0.5 s when Flow 1 finishes, Flow 2 transmits the remaining 1.5 Gb with 1 Gbps, and finishes transmission after 1.5 s. As a result, FCTs of Flow 1 and Flow 2 are 5.5 s and 5 s, thus the average FCT is 5.25 s.

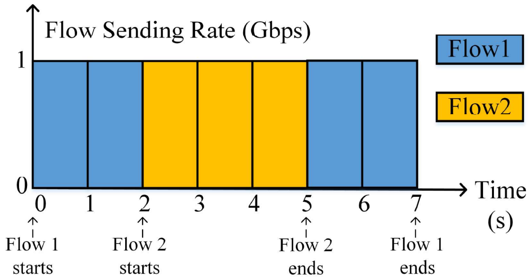


Fig. 3. A congestion-aware scheduling scheme

However, in fact, if the scheduling scheme can detect the network congestion (bandwidth occupied by higher priority flows in this example) to determine the priority, a better scheduling result can be obtained, shown in Fig. 3. That is, when Flow 2 starts, Flow 1 detects that the available bandwidth drops to 0.5 Gbps thus the remaining time is (2/0.5=4) s, and the remaining time for Flow 2 is (3/1=3) s. Therefore, Flow 2 is scheduled first. When Flow 3 finishes, Flow 1 can use 1 Gbps to transmit the remaining 2 Gb. As a result, FCTs of Flow 1 and Flow 2 are 7 s and 3 s, and the average FCT is 5 s, which is shorter.

***Remark 1**: When determining the priority for flow scheduling, it is important to obtain and utilize the real-time network information in addition to flow size.*

2.2 Congestion control without considering scheduling

In this section, we use experiments to show the problem of congestion control schemes without the consideration of flow scheduling.

(1) Problem of congestion window update without the cooperation of scheduling at switches

Here, we start with L2DCT [8], which is a congestion control scheme that aims to minimize the average flow completion time through updating flow's congestion window size according to the flow's bytes sent. Although L2DCT considers scheduling issue (priority according to the bytes sent) at senders, it simply implies a first-in-first-out scheduling at switches, without priority scheduling. On the other size, PIAS [16] implememts priority scheduling at switches but missing differentiated congestion window update at TCP senders. To see how priority scheduling could help congestion control, we implement a new scheme, called *L2DCT+PIAS*. As its name indicates, this scheme leverages the congestion window size update scheme of L2DCT at senders and implements priority scheduling of PIAS at switches, and both of them use *bytes sent* to determine the congestion window size and priority for flows. We compare the performance of L2DCT, PIAS and *L2DCT+PIAS*, and the result is shown in Fig. 4 (refer to Section 4.1 for more details about the experiment setup). As is seen, *L2DCT+PIAS* performs better than L2DCT, which indicates the necessity of congestion window update cooperating with priority scheduling at switches. In fact, the better performance of *L2DCT+PIAS* than PIAS also shows that flow scheduling needs priority aware congestion control at end hosts, to achieve better performance.

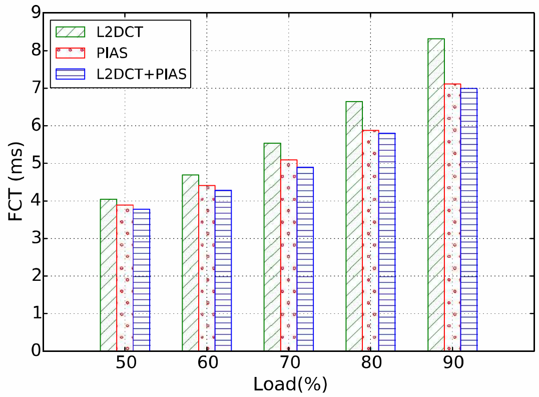


Fig. 4. Comparision of L2DCT, PIAS and L2DCT+PIAS

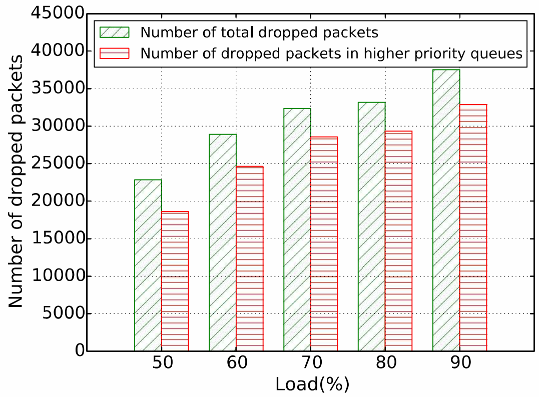


Fig. 5. Packet drops.

***Remark 2**: Congestion window update should cooperate with priority scheduling to achieve better performance.*

(2) Problem of packet dropping without the consideration of priority at switches

With the industry trend [19] that link capacity is increasing quickly but the switch buffer size is increasing slowly, the buffer size per port per Gbps is dropping. Since DCN is bursty, packet drop will become a more severe problem. Especially, short flows are more sensitive to packet dropping, therefore, dropping mechanism should take flow priority into consideration. However, even schemes with priority scheduling at switches, like PIAS, fail to identify this problem and drop short flows (high priority flows) when there are long flows in low priority queues, as long as the buffer is full, which has a negetive impact on short flows in high priority queues. To show how severe the problem is, we implement PIAS in a 100G network (refer to Section 4.5 for more details about the experiment setup), and measure the *number of total dropped packets* and the *number of dropped packets in high priority queues*[1] when lower priority queues are non-empty, and the result is shown in Fig. 5. As is seen, both the *number of total dropped packets* and the *number of*

[1] When a packet is dropped because the buffer is full, if there are packets in lower priority queues whose priority is lower than that of the dropped packet, we call this packet as *dropped packet in high priority queues*. Here, we count the total number of dropped packets in this case for this experiment and comparison.

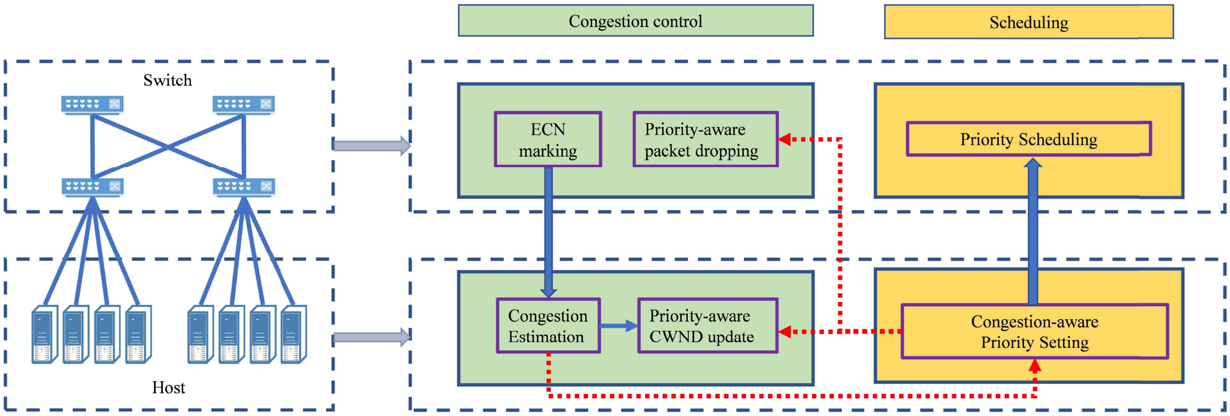

Congestion control
Scheduling
Switch
Host
ECN marking
Priority-aware packet dropping
Priority Scheduling
Congestion Estimation
Priority-aware CWND update
Congestion-aware Priority Setting

Fig. 6. *Flash* framework (the red dotted lines are the core designs of *Flash*).

dropped packets in high priority queues are increasing with the increase of network load, and the percentage of the *number of dropped packets in high priority queues* over the *number of total dropped packets* is also increasing with the increase of network load, from 81.54% to 88.42%, which greatly degrades the performance of short flows in high priority queues. From the perspective of scheduling, short flows should be given higher priority, thus when they arrive at the switch and the buffer is full, long flows in lower priority queues should be dropped, to leave space for the short flows to enqueue.

Remark 3*: It is necessary to consider flow priority when dropping packets.*

In the next section, we solve all problems aforementioned with the proposed *Flash* scheme.

3 Design

In general, *Flash* integrates flow scheduling and congestion control in data center networks, and strives to optimize each mechanism from the perspective of each other.

3.1 Overview

Inside data centers, servers are inter-connected by layers of switches. According to the functions of scheduling and congestion control in servers and switches, we summarize the framework of *Flash* in Fig. 6. There are two modules in the system, scheduling module and congestion control module.

Scheduling module. We set multiple priority queues inside switch buffers and implement strict priority scheduling among these queues. Packets carry priority information in the packet header and are enqueued into the corresponding queue that matches their priorities. Priority is set at TCP senders and take the estimated network congestion extent into consideration. The determined priority is also utilized for packet dropping and congestion window update in the congestion control module. The **core** design in *Flash*'s scheduling module is that priority setting is based on the estimated network congestion derived from *Flash*'s congestion control module.

Congestion control. We set a threshold for the total queue length of all queues in the switch port buffer and implement a per-port ECN marking scheme. At TCP senders, *Flash* utilizes the estimated network congestion extent and flow priority to determine the updated congestion window size, and the congestion extent is also used to determine flow priorities in the scheduling module. For packet dropping in switch buffers, priority determined in the scheduling module is considered which aims to drop packets in lower priority queue to leave space for packets for higher priority queues when the buffer is full. The **core** design in *Flash*'s congestion control module is that both congestion window update and packet dropping are based on flow priorities derived from *Flash*'s scheduling module.

3.2 *Flash* Design in Detail

In this section, we describe *Flash* in detail to solve the problems in Section 2.

3.2.1 *Congestion-aware Priority Setting*

Like PIAS, our *Flash* is an information-agnostic scheme which does not assume flow size information is known. Therefore, we determine priority based on the flow's bytes sent (instead of flow size or remaining flow size, which both require that the total size of the flow is known at the begining of flow transmission). Combining with the congestion extent estimator α derived from DCTCP congestion control scheme, we use W in Eqn.(1) to determine the priority of the flow.

$$W \leftarrow \frac{Bytes\_sent}{1-\alpha} \tag{1}$$

How to comprehend Eqn.(1)? (1) Many recent works [5], [20], [21] have shown that the DCN traffic has heavy-tailed distribution property, which means the majority of flows are short flows but the majority of bytes come from a small number of long flows. Therefore, according to [22], the bytes sent can be used as an approximation of the remaining flow size. (2) When the network congestion extent is high, it indicates that the network is highly utilized, thus there is less available bandwidth, and when the network congestion extent is low, it means that the network is under-utilized, thus there is more available bandwidth. Thus $(1-\alpha)$ in

Eqn.(1) estimates the extent of available bandwidth[2]. Therefore, W in Eqn.(1) is the approximation of *remaining flow size* over *available bandwidth*, which results in the estimated *remaining time* of the flow. Thus, in fact, our *Flash* follows the *shortest remaining time first* scheduling discipline, unlike pFabric using *shortest remaining size first* scheduling without considering the real-time network information.

In this paper, we set N priority queues inside each switch port buffer, thus the total number of priorities is also N, from 0 to $(N-1)$, where 0 denotes the highest priority and $(N-1)$ denotes the lowest. Therefore, we set $(N-1)$ thresholds T_i ($i \in [1, N-1]$), where $T_i \leftarrow T_1 \times E^{i}$[3]. Therefore,

- when W is smaller than T_1, the priority is set to 0;
- when W is larger than or equal to T_{N-1}, the priority is set to $(N-1)$;
- otherwise, the priority is set to i, if W is in $[T_i, T_{i+1})$.

Therefore, everytime when sending a new window of packets, we need to set the priority of a flow. We count the bytes sent of a flow and get the latest network congestion extent value α, then calculate W according to Eqn.(1), and determine the flow priority. Then the priority is set to the Differentiated Services Code Point (DSCP) field in the IP header of flow packets, which is used to classify flows into different priority queues in switch buffers for scheduling.

3.2.2 Priority-aware Congestion Control

In the congestion control module, we need to determine how to update congestion window size and how to drop packets with respect to flow priority. For congestion window size updating, we find L2DCT [8] is a good option, as it considers both network congestion extent and flow bytes sent (which can represent flow priorities and is also subject to the information-agnostic scenario). Therefore, *Flash* implements L2DCT at TCP senders to update the congestion window size to incorporate scheduling into congestion control. Next, we show how *Flash* drops packets with respect to flow priorities.

The majority of existing works simply drop the enqueued packet if the buffer is full, which greatly hurts the performance if the dropped packet belongs to high priority flows. pFabric, the best scheduling scheme until now, drops the packet of the flows with the longest remaining flow size (or say, with the lowest priority). Comparing flow remaining size to find the longest one (lowest priority) is not supported by current switches, thus pFabric is not implementable.

In *Flash*, we set N priority queues inside each switch port buffer. Flows inside the lowest priority queues have the lowest priority and should be dropped when the buffer is full. Therefore, motivated by pFabric's dropping scheme and with our priority queues, we design the dropping (and enqueueing) mechanism as in Algorithm 1.

Algorithm 1 *Flash*'s dropping (and enqueueing) mechanism

```
1: if $TotalQueueLength + Packet.Size <= BufferSize$ then
2:     Enqueue this incoming packet to its corresponding queue;
3: else
4:     $LNQ \leftarrow (N-1)$; // LNQ is the index of a queue.
5:     for $i = LNQ\ to\ 0$ do
6:         if $Q_i.length > 0$ then
7:             $LNQ \leftarrow i$;
8:             Break;
9:         end if
10:    end for
11:    if $Packet.priority >= LNQ$ then
12:        Drop this incoming packet;
13:    else
14:        Drop the tail packet in $Q_{LNQ}$;
15:        Enqueue this incoming packet to its corresponding queue;
16:    end if
17: end if
```

When a packet arrives at the switch port buffer, if the total queue length of all queues plus the packet size does not exceed the buffer size, this incoming packet is enqueued (line 1 to line 2), otherwise, a packet needs to be dropped. Here, we first need to find the Lowest Non-empty Queue (whose index is LNQ) (line 4 to line 10) which represents the lowest priority queue that has packets inside. If this incoming packet belongs to this queue or its priority is lower than that of LNQ (line 11)[4], then we drop this packet (line 12), otherwise, we drop the tail packet in the lowest non-empty queue (line 14) and enqueue this incoming packet to its corresponding queue (line 15). In general, packets (or flows) in the same queue share the same priority, and only preempt packets (or flows) in lower priority queues.

4 Experiments and Evaluations

In this section, we implement extensive experiments in NS2 [24] to evaluate the performance of *Flash*. We mainly compare our *Flash* with the following widely used schemes:

- DCTCP [5] which is a congestion control work with First in First out (FIFO) scheduling.
- L2DCT [8] which performs priority-based congestion control scheme to approximate the Least Attained Service (LAS) scheduling.
- PIAS [16] which mimics shortest job first (SJF) scheduling with the multi-level feedback queue in the information agnostic scenario.
- pFabric [9] which requires flow size is known and realizes the shortest remaining size first scheduling at switches, and has the currently best performance.

[2] Firstly, although there are some works that can measure the real-time bandwidth to make the bandwidth estimation more accurate, we find that they all incur unnecessary system overhead or network traffic. In this paper, we aim to leverage built-in functions of current congestion control schemes, without any excessive system cost. Secondly, although a positive value of α needs queue build up in switch port buffer to take effective, its relative value can represent the idleness of the network, and the change of α denotes the varying bandwidth in the network. Therefore, we find α is a suitable indicator for estimating the available bandwidth.

[3] Here, the threshold settings are motivated by [23], and we utilize the unified settings of thresholds for general cases.

[4] A large value of priority indicates lower priority.

TABLE 1
Default Parameter Setting

Parameter	Value
Link capacity	10 Gbps
Packet size	1.5 KB
Port buffer size	240 packets
Queue length threshold for port buffer	65 packets
T_1	1.2 Mb
E	3

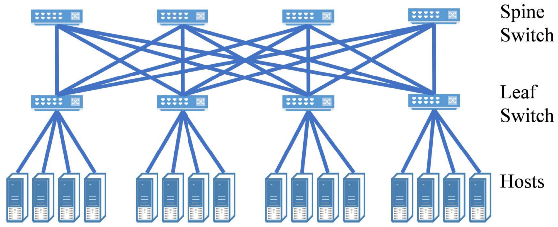


Fig. 7. Experiment topology: Leaf-Spine.

We mainly compare the average Flow Completion Time (FCT) of all five schemes to evaluate their performance. Our experiments need to answer the following key questions:

How does *Flash* perform in reducing the average FCT? Generally, *Flash* performs much better than DCTCP and L2DCT, outperforms PIAS, and achieves similar performance or better performance in some cases, when compared with pFabric. For example, *Flash* reduces the overall average FCT by up to 21.2%, 16.3%, 8.2% when compared with DCTCP, L2DCT, PIAS, and achieves similar performance with pFabric, and even has 0.1% better performance at 60% load. Given that *Flash* is informantion-agnostic while pFabric requires flow size in advance, *Flash* is a quite effective solution.

How does *Flash* perform in short flows which dominate the DCN traffic and are latency-sensitive? The performance of *Flash* in short flows is outstanding. For example, *Flash* reduces the average FCT of short flows by up to 54.8%, 48.7%, 12.2% when compared with DCTCP, L2DCT, PIAS, and reduces the 99th percentile FCTs of short flows by up to 67.2%, 55.5%, 35.8% respectively. *Flash*'s performance is very close to that of pFabric.

How does *Flash* perform in different workloads? *Flash* can maintain its superior performance for different individual workloads.

How does *Flash* perform in information aware scenario? If flow size is known in advance just like that in pFabric, *Flash* can outperform pFabric by up to 13.5%.

How does *Flash* perform in high speed network with shallow buffers? We implement *Flash* in a 100 Gbps network with shallow buffers and the results show that *Flash* outperforms existing works like reducing the overall average FCT by 6.8% and 26.0% when compared with pFabric and PIAS.

Is *Flash* robust? We implement *Flash* with different numbers of queues, and the results show that the performance difference under different numbers of queues is small, which indicates that *Flash* is robust.

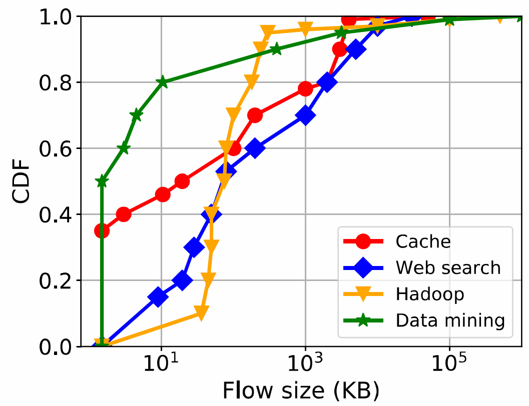


Fig. 8. Flow size distributions for individual workload

TABLE 2
Mean Flow Size (MFS) and Number of Flows of Individual Workloads in Mixed Workload Scenario

Workload	MFS (KB)	Number of Flows (S/M/L)
Cache	914	26505 (16034/10253/218)
Web Search	1671	14453 (7881/6180/392)
Hadoop	4149	5829 (4131/1523/175)
Data Mining	7495	3213 (2638/420/155)

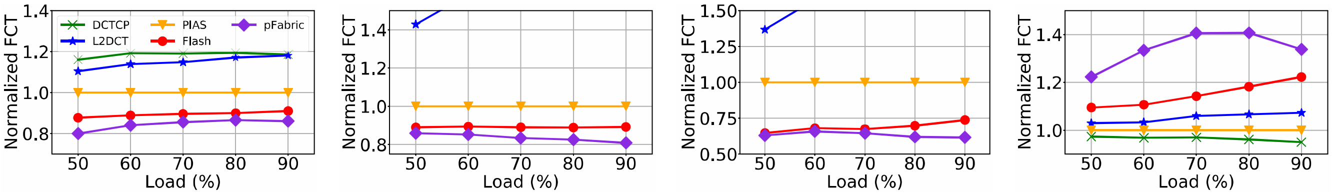
4.1 Experiment Setup

To ensure fair comparison, the majority of experiment settings are consistent with those in the PIAS paper [16], including the same Leaf-Spine topology, the same workloads and so on.

Parameters: Unless stated explicitly, we set the parameters to the default values shown in Table 1. For DCTCP, L2DCT, PIAS and pFabric, we set all relevant parameters to the default values in their original papers [5], [8], [16], [9].

Network topology: We use one of the most popular topologies, Leaf-Spine topology shown in Fig. 7 as our experiment topology. In our experiments, we have 12 Leaf switches, 12 Spine switches and 144 hosts (servers). Each Leaf switch connects to 12 Spine switches through 10 Gbps uplinks, and connects to 12 hosts through 10 Gbps downlinks, thus forming a non-blocking network. ECMP [25] is adopted for routing and load balancing in the multipath environment. We set 8 queues for each switch port by default for flow scheduling. We enable per-port ECN marking instead of per-queue ECN marking, to ensure both high throughput and low latency in the multiple queue scenario [16], [13]. We set initial TCP window size to 10 packets, and both initial and minimum RTOs to 5ms for all schemes, as many recent works recommend [14], [13], [26].

Workloads: In this paper, we run four different workloads for experiments, including a cache workload [21], a web search workload [5], a Hadoop workload [21], and a data mining workload [20], and the flow size distributions for these four workloads are shown in Fig. 8. We first run two sets of experiments, each with one single workload where the first is web search workload and the other is data mining worload. We generate 50000 flows in total for each experiment. Then, we run all four workloads together to



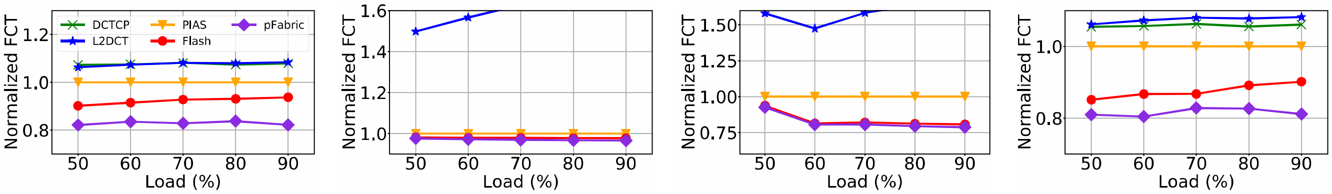
(a) Overall

(b) Short flows

(c) 99th percentile FCTs of short flows

(d) Long flows

Fig. 9. Comparison of FCT in web search workload. DCTCP's performance and parts of L2DCT's performance are outside the plotted range of (b) and (c), due to their inferior performance.



(a) Overall

(b) Short flows

(c) 99th percentile FCTs of short flows

(d) Long flows

Fig. 10. Comparison of FCT in data mining workload. DCTCP's performance and parts of L2DCT's performance are outside the plotted range of (b) and (c), due to their inferior performance.

mimic the mixed workload scenario, and generate totally 50000 flows. For each workload, the inter-flow arriving interval t is inversely proportional to the network *Load*, *i.e.*,

$$\ell = \frac{C \times Load}{MeanFlowSize}, \quad t = \frac{1}{\ell} \tag{2}$$

where C is the link capacity, so ℓ represents the number of flows arriving in unit time. As we can see, the higher the load, the lower the inter-flow arriving interval, leading to higher flow contention. And the inter-flow arriving intervals for different workloads are also different, *i.e.*, proportional to the mean flow size. Thus the smaller mean flow size, the smaller inter-flow arriving interval, and the larger number of flows given the same experiment running time for all workloads. We summarize the mean flow size and the number of flows for each workload in the mixed workload scenario, as shown in Table 2.

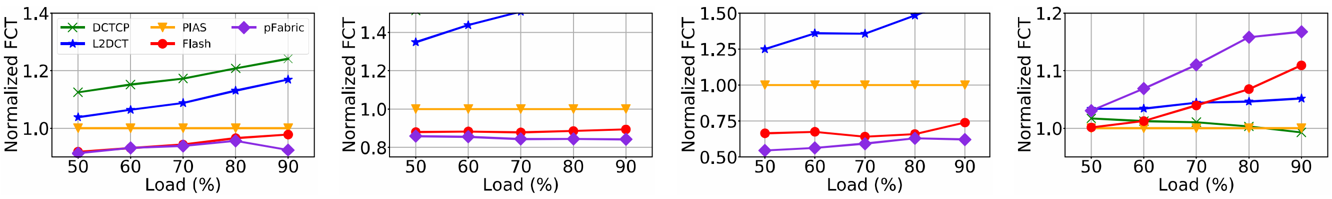
Comparison metrics: According to the flow sizes, flows are divided into three classes including short flows ((0,100] KB), medium flows ((100 KB,10 MB]) and long flows ((10 MB,∞)). For example, in the mixed workload scenario, the numbers in the brackets in Table 2 show the numbers of flows in each class, *e.g.*, Web Search workload has 14453 flows in total, among which there are 7881 short flows, 6180 medium flows and 392 long flows. We compute the overall average FCT for all flows, the average FCT of flows in each class, and the 99th percentile FCTs of short flows as the comparison metrics.

4.2 Performance in Average FCT

To evaluate *Flash*'s performance in reducing the average FCT, we first implement experiments for the five schemes based on the default settings aforementioned using web search workload and data mining workload, and the experiment results are shown in Fig. 9 and Fig. 10. In this paper, experiment results of all schemes are normalized to the results of PIAS, for convenient and intuitive comparison. For example, if the FCTs for L2DCT, PIAS and *Flash* are 2.2s, 2s, and 1.8s, the normalized FCTs are 1.1, 1 and 0.9 respectively (thus, normalized FCTs for PIAS are always 1). According to the comparison results, we have the following observations:

Overall: As shown in Fig. 9(a) and Fig. 10(a), *Flash* performs better than DCTCP, L2DCT and PIAS across all loads. For example, *Flash* has 9.0% to 12.3% lower FCT than PIAS in web search workload, and has 6.3% to 9.8% lower FCT than PIAS in data mining workload. When compared with DCTCP and L2DCT, the performance improvement is more significant. For example, *Flash* has average 21.3% lower FCT than L2DCT in web search workload and average 14.5% lower FCT than L2DCT in data mining workload, and when compared with DCTCP, the improvements are 24.1% in web search workload and 14.3% in data mining workload. When compared with pFabric, the performance gap is from 5.1% to 7.6% for web search workload and 10.1% to 15.7% for data mining workload. The main reason for the performance gap is because our *Flash* is information-agnostic which does not require the flow size information for scheduling while pFabric needs flow size to determine priority (if flow size is also aware in *Flash* like that in pFabric, *Flash* can perform better than pFabric (see more in Section 4.4)). Therefore, *Flash* is effective in reducing the average FCT.

Short flows and tail: *Flash* performs better than PIAS, and greatly outperforms DCTCP and L2DCT in short flows. For example, in web search workload, *Flash* reduces the FCT for short flows by 10.9% to 11.02%, 37.7% to 57.2%, and 42.9% to 61.3% when compared with PIAS, L2DCT and DCTCP respectively, and the improvements in the 99th percentile are more significant, which are 26.3% to 35.3%, 52.6% to 62.6% and 62.9% to 69.5% when compared with PIAS, L2DCT and DCTCP respectively. *Flash* is slightly inferior to pFabric in short flows, and the reason is that *Flash*



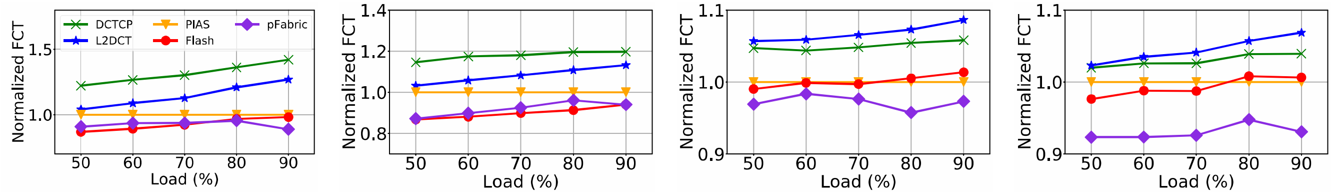
(a) Overall

(b) Short flows

(c) 99th percentile FCTs of short flows

(d) Long flows

Fig. 11. Comparison of FCT in mixed workload. DCTCP's performance and parts of L2DCT's performance are outside the plotted range of (b) and (c), due to their inferior performance.



(a) Cache

(b) Web search

(c) Hadoop

(d) Data mining

Fig. 12. Comparison of FCT for individual workload in mixed workloads scenario.

is information agnostic. In data mining workload, PIAS, *Flash* and pFabric outperform L2DCT and DCTCP a lot because PIAS, *Flash* and pFabric all implement scheduling at switches while L2DCT and DCTCP do not, but the performance gap between *Flash* and pFabric is minimal. This is because data mining workload is more skewed. In fact, in data mining workload, around 82% flows are short flows which are smaller than 100KB. Thus they finish transmission in the highest priority queue (or the highest few queues) before they are demoted to lower priority queues, thus the scheduling mechanism does not function much for these flows.

Long flows: In web search workload, both *Flash* and pFabric perform worse than PIAS, L2DCT and DCTCP. This is expected because both *Flash* and pFabric more significantly prioritize short flows over long flows through dropping low priority flows, thus having a reverse impact on long flows. However, to our surprise, in data mining workload, both *Flash* and pFabric perform better than PIAS, L2DCT and DCTCP. We analyze the reason as follows. Since data mining workload is more skewed, whose 82% flows are short flows smaller than 100KB, therefore, short flows leave little network footprints and seldom occupy network bandwidth or switch port buffers. Therefore, they have little impact on long flows in data mining workload. In this case, long flows are gradually demoted to lower priority queue as more bytes are transmitted, during which time flow scheduling using multiple priority queues take effect for these flows. Therefore, *Flash* and pFabric could have better performance in long flows in data mining workload.

4.3 Performance for Mixed Workload

To evaluate *Flash*'s performance for mixed workload scenario, we repeat above experiments while mix four worloads together and generate 50000 flows in total according to Section 4.1, and the mean flow size and the number of flows are shown in Table 2. The comparison results are shown in Fig. 11. As we can see, *Flash* maintains the superior performance in the mixed workload scenario. For example, For example, *Flash* has 2.2% to 8.2% lower FCT than PIAS. When compared with DCTCP and L2DCT, the performance improvement is more significant. For example, *Flash* has average 16.3% lower FCT than L2DCT and 21.2% lower FCT than DCTCP. For short flows, *Flash* reduces the FCT for short flows by 10.8% to 12.2%, 34.7% to 48.7% and 41.8% to 54.8% when compared with PIAS, L2DCT and DCTCP respectively. The performance improvement in the 99th percentile is more significant, which has a reduction of 26.1%-35.8%, 46.8%-55.5% and 61.7%-67.2%. In the meanwhile, *Flash* has similar performance with pFabric. For long flows, both *Flash* and pFabric perform slightly worse than PIAS, L2DCT and DCTCP.

Furthermore, to evaluate *Flash*'s performance for individual workload in the mixed workload scenario, we break down the comparison into individual workload in the mixed workload scenario, and the results are shown in Fig. 12. As we can see, *Flash* maintains its better performance in most cases, especially in cache workload and web search workload which have short mean flow size, and the performance improvement is less significant in Hadoop workload and data mining workload. This is expected, since the percentage of long flows ((10 MB,∞)) in Hadoop workload and data mining workload are higher than those in cache workload and websearch workload, and the mean flow sizes of Hadoop workload and data mining workload are also larger than those of cache workload and websearch workload. Since *Flash* has better performance in short flows ((0,100] KB), the performance in cache workload and websearch workload is better.

4.4 Performance with information aware

Note that the currently best scheme pFabric requires flow size is known while our *Flash* does not. We would like to see if *Flash*'s performance could be further improved if flow size is known. Like pFabric, if we know the flow size in advance and count the bytes sent as flows are

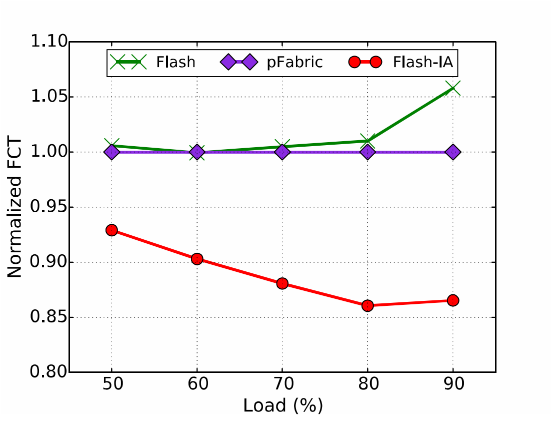


Fig. 13. Performance of *Flash* if flow size is known.

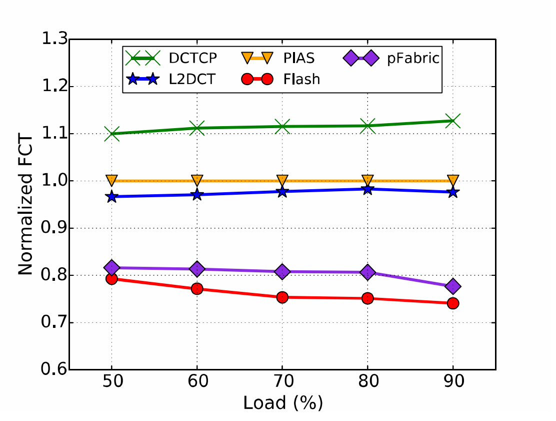


Fig. 14. Comparison of FCT in high speed network with shallow buffers.

transmitting, we can obtain the remaining flow size. We use *remaining flow size* to replace the $Bytes\_sent$ in Eqn.(1), and all other mechanisms remain un-changed. We call this new scheme as *Flash-IA*, which is an **I**nformation-**A**ware version of *Flash*. We compare *Flash*, pFabric and *Flash-IA*, and the comparison results are shown in Fig. 13 where all results are normalized to the results of pFabric. As we can see, *Flash-IA* ourperforms pFabric by 7.1%-13.5% in overall performance, which shows the superiority of our scheme, which jointly considers flow scheduling and congestion control.

4.5 Performance in High Speed Network with Shallow Buffers

As [19] indicates, DCN link capacity is increasing fast while switch buffer size is increasing slowly, leading to a high speed shallow buffer scenario. Here we implement experiments to evaluate *Flash*'s performance in high speed networks with shallow buffers. We set link capacity to 100 Gbps, per port buffer size to 512K and per port ECN marking threshold to 250K, all according to [19], then repeat the experiments with four workloads in Section 4.2 and the comparison results are shown in Fig. 14. As we can see, *Flash* achieves better performance than pFabric and PIAS with 2.8%-6.8% and 20.7%-26.0% overall improvement respectively, and also outperforms other schemes more.

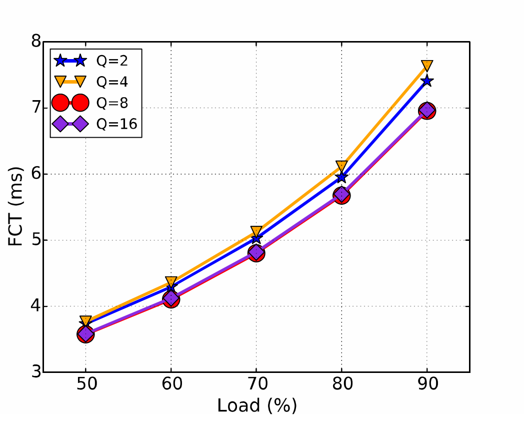


Fig. 15. Performance of *Flash* with different numbers of queues.

4.6 Impact of Number of Queues

Finally, to assess *Flash*'s robustness, we evaluate *Flash*'s performance with different numbers of queues. Since we use the priority queues in switch buffer for flow scheduling, the number of queues limits the number of priorites for scheduling. Current commodity switches support 8 queues, but system operators may reserve some queues for other purpose, for example, reserve highest priority queue for control traffic in SDN area [14]. Thus, the number of available queues becomes less. It is also expected that in the near future, switches can support more queues. To this end, we implement experiments with different numbers of queues, ranging from 2, 4, 8 and 16, to see the robustness of *Flash*, and the results are shown in Fig. 15. In general, more queues lead to better overall performance, since better separations are possible between long flows and short flows with more queues, leading to less competition among different classes of flows, thus resulting in better scheduling performance. But the performance gap among differnt numbers of queues is marginal, which indicates that our *Flash* is robust.

4.7 Impact on Timeouts

Algorithm 1 aims to leave space for high priority queues thus drop packets in the lowest priority queue when the buffer is full, which might lead to more timeouts for lower priority flows. However, this has little impact on the overall performance. This is because DCN traffic follows heavy-tailed distribution, thus short flows contribute a small fraction of the overall traffic, hence prioritizing them has little impact on long flows and helps them because they complete quickly which reduces network contention.

To quantify the effect, we repeat the experiments in Section 4.3 with mixed workloads and compare the numbers of timeouts of Flash with PIAS and pFabric in four categories including overall flows, short flows, medium flows and long flows, and the comparison results are shown in the Table 3. For example, at 50% load, Flash has totally 118 timeouts, among which there are 0, 32 and 86 timeouts for short flows, medium flows and long flows respectively.

From Table 3, we can see that, Flash did increase the overall timeouts when compared with PIAS, but much less than pFabric. To break down the comparison, we see that for short flows and medium flows, Flash successfully reduces

TABLE 3
The comparison of numbers of timeouts (overall(short/medium/long) flows)

Scheme	50% load	60% load	70% load	80%load	90%load
PIAS	34(0/23/11)	80(1/51/28)	129(1/78/50)	177(0/80/97)	291(2/110/179)
Flash	118(0/32/86)	187(0/29/158)	345(0/44/301)	481(0/57/424)	821(0/104/717)
pFabric	37K(0/13K/24K)	51K(0/17K/34K)	70K(0/24K/46K)	95K(1/32K/63K)	123K(0/42K/81K)

the numbers of timeouts when compared to PIAS. The main increase of timeouts derives from long flows. However, we can see that Flash only increases the numbers of timeouts by 3 to 6.8 times more when compared the PIAS, unlike pFabric which increases by 452 to 2188 times more.

In fact, when the buffer is full, Flash drops packets in the lowest priority queues which may lead to more timeouts for long flows. However, Flash mitigates this side-effect through per-port ECN marking. Switches mark packet for congestion notification when the total queue length exceeds the threshold, and the threshold is typically smaller than the buffer size. In this case, when a packet of a short flow (higher priority) enqueues and the total queue length of all packets in the buffer (including all packets in high and low priority queues) exceeds the per-port threshold, this packet will be marked. Therefore, this short flow will be slowed down by reducing the CWND. In this case, high priority queues are more likely to be empty and packets in low priority queues will be transmitted, thus the size-effect to lowest priority queue is mitigated.

5 Related Work

Flow scheduling and congestion control are two important techniques in DCN areas. Here we summarize main works in these areas respectively.

5.1 Flow Scheduling

DeTail [27] leverages cross-layer information to prioritize latency-sensitive flow and distribute network load, in order to reduce the long tail of FCT. PDQ [6] utilizes switch arbitration and allows flow preemption which performs flow scheduling in a distributed manner, in order to reduce FCT and meet deadlines. Karuna [17] considers the mixed-flow scenario where some flows are with deadlines and some are not, and proposes to set minimum bandwidth for deadline-sensitive flows to finish flow transmission just in time and leave remaining bandwidth to deadline-free flows, thus satisfies both types of flows. PASE [10] is a synthesized solution which combines self-adjusting endpoints, in-network prioritization and arbitration to reduce average FCT. pFabric [9] is currently the best scheduling scheme in DCN. pFabric requires flow size information to carry in packet header and performs shortest remaining size first scheduling at switches. [28] adopts Least Slack Time First (LSTF) scheduling on top of programmable switches so that it can approximate the state-of-the-art works and achieve different goals like reducing latency and ensuring fairness. Homa [29] proposes a new scheduling scheme which determines the flow priority at receiver side. 2D [30] proposes a workload adaptive scheduling scheme which includes coarse time-scale decisions based on workload and per-flow serialization decisions in a distributed fashion, which can ensure tail performance and also decrease FCT.

However, the majority of existing works assume information is aware, which means flow size is known in advance, and the flow size (or remaining flow size) is utilized as the metric for scheduling. But for many applications, flow size is hard to know in advance, making most of exiting works less effective. To solve this problem, PIAS [16] utilizes the multi-level feedback queue to mimic the shortest job first scheduling scheme without the flow size information in advance. Aemon [18] extends PIAS's information-agnostic scenario and considers the case that some flows have deadlines and some do not, and proposes a mix-flow transport based on flow urgency.

5.2 Congestion Control

DCTCP [5] sets a threshold for the queue length in switch buffers and uses ECN marking to convey network congestion, and end hosts estimate the congestion extent in order to adjust congestion window size. L2DCT [8] adopts the Least Attained Service (LAS) scheduling discipline and uses bytes sent of a flow to adjust the congestion window size, which helps reduce the average FCT. ICTCP [31] targets at TCP incast and proposes to adjust TCP receive window proactively to avoid packet loss. DX [32] measures one-way queueing delay and uses the latency feedback for congestion control. TIMELY [33] leverages the RTT information measured at end hosts to perform congestion control. D2TCP [15] is a deadline-aware scheme which adjusts congestion window size according the flow deadlines. Flowtune [34] proposes to perform congestion control in the granularity of flowlets. PCC [35] puts forward the performance-oriented congestion control scheme, where TCP senders perform congestion control based on the empirically experienced performance.

5.3 Summary

While existing DCN works in these two areas work well, the interactions between flow scheduling and congestion control are overlooked which leads to sub-optimal solutions. For example, while PIAS, pFabric and other works achieve good results, they fail to consider the varying available bandwidth [36] or traffic burst issue [37] for flow scheduling, which leads to inferior performance. Our *Flash* is different from other works in that we use real-time network congestion extent information to evaluate the available bandwidth in order to determine flow priority for flow scheduling, which brings better results in the dynamic and bursty network. Besides, our *Flash* jointly considers flow scheduling and congestion control, and re-designs the two schemes from the perspective of each other, which achieves good results.

While our *Flash* does not explicitly provide bandwidth/fairness guarantee like [38], *Flash* can provide overall better performance and ensure fairness in an implicit way. First, using ECN marking scheme and standard threshold for total queue length can sure both overall high throughput and low latency. Second, as the majority of DCN traffic are short flows, using multiple priority queues and strict priority scheduling can ensure short flows finish transmission in the first few queues, thus ensure low latency for short flows. Besides, since the flow demotion (to lower priority queues) thresholds are uniform for all workloads and flows, fairness is implicitly ensured among flows, in that, (1) When flows are in high priority queues, anytime when the bytes sent of a flow exceeds the demotion threshold of this queue, the flow will be demoted. Since the demotion thresholds are uniform for all flows, each flow can transmit the same maximum amount of data in each queue (except the lowest priority queue). (2) When flows are demoted to the lowest priority queue, all flows are served in a first in first out manner, which approximates fair sharing among flows.

6 Conclusion

DCN requires low FCT to ensure the quality of service, and flow scheduling and congestion control are two important techniques to reduce the average FCT in DCN. While there are many existing works in these two areas, the interactions between them are in general overlooked, leading to suboptimal solutions. To solve this problem, we propose *Flash*, a *simple yet effective* scheme which integrates scheduling and congestion control and re-designs these two mechanisms from the perspective of each other. First, *Flash* proposes a congestion extent aware scheduling scheme to determine the flow priority based on the latest network congestion extent and flow's bytes sent. Then, *Flash* puts forward a priority-based packet dropping scheme and implements a priority-based congestion window size update scheme. Finally, we implement extensive experiments to evaluate the performance of *Flash*, and the results show that *Flash* delivers outstanding performance: (1) it has 35.8% lower tail latency than PIAS and performs similar with pFabric in a 10G network without knowing the flow size, (2) in 100G networks with shallow buffers, the information agnostic *Flash* has 6.8% lower average FCT than the information-aware pFabric, (3) it outperforms pFabric by 13.5% in FCT if flow size is also known to *Flash*.

Acknowledgment

This work is supported by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B010164003, National Key R&D Program of China under Grant 2019YFB2102100, Science and Technology Development Fund of Macao S.A.R (FDCT) under Grant 0015/2019/AKP, Shenzhen Engineering Research Center for Beidou Positioning Service Technology under Grant XMHT20190101035, and Key Program of Shenzhen under Grant JCYJ20170818153016513.

References

[1] B. Hayes, "Cloud Computing," *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[2] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and Virtual Machines at Scale: A Comparative Study," in *Proc. of ACM Middleware*, 2016, pp. 1–13.

[3] M. Xu and R. Buyya, "BrownoutCon: A software system based on brownout and containers for energy-efficient cloud computing," *The Journal of Systems and Software*, vol. 155, pp. 91–103, 2019.

[4] E. Jonas, J. Schleier-Smith, V. Sreekanti, C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. Popa, I. Stoica, and D. Patterson, "Cloud Programming Simplified: A Berkeley View on Serverless Computing," University of California, Berkeley, Tech. Rep., 2019.

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proc. of ACM SIGCOMM*, 2010, pp. 63–74.

[6] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing Flows Quickly with Preemptive Scheduling," in *Proc. of ACM SIGCOMM*, 2012, pp. 127–138.

[7] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is More: Trading a little Bandwidth for Ultra-Low Latency in the Data Center," in *Proc. of USENIX NSDI*, 2012, pp. 1–14.

[8] A. Munir, I. Qazi, Z. Uzmi, A. Mushtaq, S. Ismail, M. Iqbal, and B. Khan, "Minimizing Flow Completion Times in Data Centers," in *Proc. of IEEE INFOCOM*, 2013, pp. 2157–2165.

[9] M. Alizadeh, S. Yang, and M. Sharif, "pFabric: Minimal Near-optimal Datacenter Transport," in *Proc. of ACM SIGCOMM*, 2013, pp. 435–446.

[10] A. Munir, G. Baig, S. Irteza, I. Qazi, A. Liu, and F. Dogar, "Friends, not Foes: Synthesizing Existing Transport Strategies for Data Center Networks," in *Proc. of ACM SIGCOMM*, 2014, pp. 491–502.

[11] M. Li, D. Andersen, J. Park, A. Smola, A. Ahmed, V. Josifovski, J. Long, E. Shekita, and B. Su, "Scaling distributed machine learning with the parameter server," in *Proc. of USENIX OSDI*, 2014, pp. 583–598.

[12] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. of USENIX OSDI*, 2016, pp. 265–283.

[13] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ecn in multi-service multi-queue data centers," in *Proc. of USENIX NSDI*, 2016, pp. 537–549.

[14] W. Bai, K. Chen, L. Chen, C. Kim, and H. Wu, "Enabling ECN over Generic Packet Scheduling," in *Proc. of ACM CoNEXT*, 2016, pp. 191–204.

[15] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-Aware Datacenter TCP (D2TCP)," in *Proc. of ACM SIGCOMM*, 2012, pp. 115–126.

[16] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-Agnostic Flow Scheduling for Commodity Data Centers," in *Proc. of USENIX NSDI*, 2015, pp. 455–468.

[17] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling Mix-flows in Commodity Datacenters with Karuna," in *Proc. of ACM SIGCOMM*, 2016, pp. 174–187.

[18] T. Wang, H. Xu, and F. Liu, "Aemon: Information-agnostic Mix-flow Scheduling in Data Center Networks," in *Proc. of ACM APNet*, 2017, pp. 106–112.

[19] W. Bai, K. Chen, S. Hu, K. Tan, and Y. Xiong, "Congestion control for high-speed extremely shallow-buffered datacenter networks," in *Proc. of ACM APNet*, 2017, pp. 29–35.

[20] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proc. of ACM SIGCOMM*, 2009, pp. 51–62.

[21] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. Snoeren, "Inside the Social Network's (Datacenter) Network," in *Proc. of ACM SIGCOMM*, 2015, pp. 123–137.

[22] Z. Li, W. Bai, K. Chen, D. Han, Y. Zhang, D. Li, and H. Yu, "Rate-Aware Flow Scheduling for Commodity Data Center Networks," in *Proc. of IEEE INFOCOM*, 2017, pp. 1–9.

[23] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, 2015, pp. 393–406.

[24] "The Network Simulator - NS-2." [Online]. Available: http://www.isi.edu/nsnam/ns/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2021.3129511, IEEE Transactions on Cloud Computing

[25] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," 2000. [Online]. Available: https://tools.ietf.org/html/rfc2992

[26] G. Judd, "Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter." in *Proc. of USENIX NSDI*, 2015, pp. 145–157.

[27] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *Proc. of ACM SIGCOMM*, 2012, pp. 139–150.

[28] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in *Proc. of USENIX NSDI*, 2016, pp. 501–521.

[29] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. of ACM SIGCOMM*, 2018, pp. 221–235.

[30] A. B. Faisal, H. M. Bashir, I. A. Qazi, Z. Uzmi, and F. R. Dogar, "Workload adaptive flow scheduling," in *Proc. ACM CoNEXT*, 2018.

[31] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data-center networks," *IEEE/ACM Transactions on Networking (TON)*, vol. 21, no. 2, pp. 345–358, 2013.

[32] C. Lee, C. Park, K. Jang, S. Moon, and D. Han, "Accurate latency-based congestion feedback for datacenters," in *Proc. of USENIX ATC*, 2015, pp. 403–415.

[33] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, 2015, pp. 537–550.

[34] J. Perry, H. Balakrishnan, and D. Shah, "Flowtune: Flowlet control for datacenter networks," in *Proc. of USENIX NSDI*, 2017, pp. 421–435.

[35] M. Dong, Q. Li, D. Zarchy, P. Godfrey, and M. Schapira, "PCC: Re-architecting congestion control for consistent high performance," in *Proc. of USENIX NSDI*, 2015, pp. 395–408.

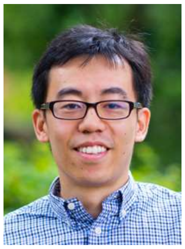
[36] F. Liu, J. Guo, X. Huang, and J. Lui, "eBA: Efficient Bandwidth Guarantee Under Traffic Variability in Datacenters," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 506–519, 2017.

[37] P. Jin, J. Guo, Y. Xiao, R. Shi, Y. Niu, F. Liu, C. Qian, and Y. Wang, "PostMan: Rapidly Mitigating Bursty Traffic by Offloading Packet Processing," in *Proc. of USENIX ATC*, 2019, pp. 849–862.

[38] J. Guo, F. Liu, T. Wang, and J. Lui, "Pricing Intra-Datacenter Networks with Over-Committed Bandwidth Guarantee," in *Proc. of USENIX ATC*, 2017, pp. 69–81.

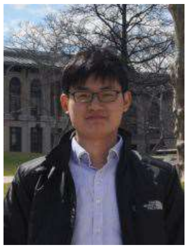
Chengxi Gao is an assistant professor at Shenzhen Institute of Advanced Technology (SIAT), Chinese Academy of Sciences (CAS). Before joining CAS, he was a research associate in City University of Hong Kong. He received his PhD degree from the Department of Computer Science, City University of Hong Kong, and his B.S. and M.S. degrees from the Department of Computer Science, Northeastern University, China. His research interests include data center networking and networking systems.

Shuhui Chu Shuhui Chu received the B.E. and the M.S. degree in computer science and technology from Jilin University, Changchun, China, in 2017 and 2020, and was a visiting student at Shenzhen Institute of Advanced Technology (SIAT), Chinese Academy of Sciences (CAS). She is currently pursuing the Ph.D. degree in computer science in University of Macau. Her research interests include network communications, game theory and reinforcement learning.



Hong Xu (SM19) received the M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto. He is currently an Associate Professor with the Department of Computer Science and Engineering, Chinese University of Hong Kong. His research interests include data center networking, SDN, NFV, and cloud computing. He was a recipient of the Early Career Scheme Grant from the Research Grants Council of Hong Kong, 2014. He also received the best paper awards from the IEEE ICNP 2015 and the ACM CoNEXT Student Workshop 2014. He is a senior member of the IEEE and a member of the ACM.

Minxian Xu is currently an assistant professor at Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences. He received the BSc degree in 2012 and the MSc degree in 2015, both in software engineering from University of Electronic Science and Technology of China. He obtained his PhD degree from the University of Melbourne in 2019, and his thesis was awarded the 2019 IEEE TCSC Outstanding Ph.D. Dissertation Award. His research interests include resource scheduling and optimization in cloud computing.



Kejiang Ye received his BSc and PhD degree from Zhejiang University in 2008 and 2013 respectively. He was also a joint PhD student at The University of Sydney from 2012 to 2013. After graduation, he works as Post-Doc Researcher at Carnegie Mellon University from 2014 to 2015 and Wayne State University from 2015 to 2016. He is currently a Professor at Shenzhen Institute of Advanced Technology, Chinese Academy of Science. His research interests focus on the performance, energy, and reliability of cloud computing and network systems.

Chengzhong Xu received the PhD degree from the University of Hong Kong, in 1993. He is currently the Dean of Faculty of Science and Technology, University of Macau, and a Chair Professor of Computer and Information Science. He is a Chief Scientist of Key Project on Smart City of MOST, China and a Principal Investigator of the Key Project on Autonomous Driving of FDCT, Macau SAR. His research interests include parallel and distributed systems and cloud computing. He has published more than 300 papers in journals and conferences. He serves or served on a number of journal editorial boards, including IEEE Transactions on Computers (TC), IEEE Transactions on Cloud Computing (TCC), IEEE Transactions on Parallel and Distributed Systems (TPDS). He is a fellow of the IEEE.