D-SRTF: Distributed Shortest Remaining Time First Scheduling for Data Center Networks

Chengxi Gao, Victor C.S. Lee, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

Abstract—Many recent works utilize scheduling to minimize the Flow Completion Time (FCT) in Data Center Networks (DCN), like PIAS using Shortest Job First (SJF) scheduling and pFabric using Shortest Remaining Size First (SRSF) scheduling. However, they *only* consider the flow size information, without consideration of available bandwidth of the network, leading to inferior performance when the network is congested. Besides, information on flow size is hard to obtain in practice. Moreover, although a centralized scheduler may have optimal scheduling decisions, it suffers from high system overhead. Therefore, a new DCN scheme is expected which is deployment-friendly and implements SRTF scheduling in a distributed manner. In this paper, we propose *D-SRTF*, a *light-weight yet effective* DCN scheme to implement SRTF scheduling. *D-SRTF* determines the remaining time of each flow according to the estimated remaining flow size and the available bandwidth, in order to determine the priority of each flow. Switches perform Strict Priority (SP) scheduling according to the priority of each flow, in order to realize SRTF scheduling. Experiments show that *D-SRTF* performs better than the currently best implementable scheme, PIAS, and could perform better than pFabric if information on flow size is available.

Index Terms—Data center networks, network protocols, scheduling

1 Introduction

1.1 Background

In the era of cloud computing, tremendous amount of services, like web service, social networking, data mining, Hadoop, recommendation systems and data storage, are deployed in cloud data centers, and they all rely on high performance data centers to improve the quality of service (QoS) [1], [2], [3], [4], [5], [6]. Each service generates large amount of network flows consisting of numbers of packets, and minimizing the average flow completion time (FCT) has always been one of the most important goals for data center networks (DCN). A lot of works have been proposed to minimize average FCT, like DCTCP [4] which reduces queue length of switch buffers, HULL [5] which trades network bandwidth for low latency, and so on. These works follow a "fair-share" manner, as flows can evenly share the link capacity.

Some recent works, like PDQ [1], have shown that, through prioritizing short flows over long flows, the average FCT could be further reduced, with shortest job first (SJF) scheduling. pFabric [3] considers the remaining flow size and performs shortest remaining size first (SRSF) scheduling to approximate shortest remaining time first (SRTF) scheduling, and achieves theoretically the best performance. However, these works assume that the flow size is known as a *priori*. In fact, it is difficult to obtain the flow size information, thus schemes with this assumption are hard to realize in practice [7]. Then PIAS [7], an information-agnostic flow scheduling scheme is proposed. PIAS does not rely on flow size information, and leverages multiple priority queues in existing commodity switches to implement the multiple level feedback queue (MLFQ), so that a flow is demoted from the highest priority queue to the lowest during its lifetime based on its bytes sent. Since short flows are short-lived, they are likely to finish transmission in the first few high priority queues, thus in general, PIAS mimics SJF.

However, all above works and many recent proposals implicitly assume that each flow can use the whole link capacity for transmission. But this is not true in many scenarios. For example, (1) some bandwidth is reserved for special usage, like delivering the control traffic in SDN area [8], or (2) in multiple-service scenario [8], [9], [10], [11], when multiple services co-exist in the network, the traffic for each service can only use a portion of the link capacity. As a result, existing works fail to capture the real-time available bandwidth for flow transmission and scheduling, thus simply scheduling a short flow (or a flow with shorter remaining size) over a long flow (or a flow with longer remaining size) will lead to inferior networking performance (see Section 3.1.2). Therefore we need to estimate the available bandwidth of each flow, decide the remaining time according to the remaining flow size and the available bandwidth, and schedule the flows with the shortest remaining time first.

However, to realize SRTF scheduling in practice, we face the following challenges.

First, flow size is unknown for many applications, thus it is hard to obtain the remaining flow size.

- *C. Gao is with the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, and was with the Department of Computer Science, City University of Hong Kong. E-mail: chengxi.gao@siat.ac.cn.*
- *V.C.S. Lee is with the Department of Computer Science, City University of Hong Kong, Hong Kong. E-mail: csvlee@cityu.edu.hk.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*

Manuscript received 29 Jan. 2018; revised 22 July 2018; accepted 30 Sept. 2018. Date of publication 2 Nov. 2018; date of current version 4 June 2021. (Corresponding author: Chengxi Gao.)
Recommended for acceptance by V. C. Leung.
Digital Object Identifier no. 10.1109/TCC.2018.2879313

Second, bandwidth measurement will increase the burden and is hard to realize. Usually, to measure the available bandwidth, we need to measure the overall throughput of all existing flows on one link, and the available bandwidth is the difference between the overall throughput and the link capacity. However, since DCN is dynamic, it is difficult and expensive to accurately measure the overall throughput on each link, let alone find the least available bandwidth of all links on a flow's path, which is the bottleneck.

Third, ideal SRTF assumes a centralized scheduler which knows the state of the whole system [3]. However, this is impossible for DCN which is a distributed network system, where such a centralized scheduler would induce significant overhead and delay the scheduling decisions.

Therefore, an effective DCN scheme should be: (1) *Information-agnostic:* The scheme could not assume that flow sizes and bandwidth are known from applications. (2) *Distributed scheduling:* Although a centralized scheduler could deliver optimal solution in theory [3], it suffers from high overhead and delays the scheduling decisions, especially for short flows which dominate the DCN traffic [4], [5]. So scheduling decisions should be made in a distributed manner. (3) *Deployment-friendly:* The scheme should work with existing commodity switches, and be compatible with legacy TCP/IP stacks. (4) *FCT minimization:* The scheme should minimize the overall FCT. Since short flows dominate the DCN traffic and are more delay-sensitive [4], [5], [7], [12], it is also expected that the scheme could significantly reduce the average FCT and tail FCT for short flows, without jeopardizing the performance of long flows much.

1.2 Our Contributions

In this paper, we propose *D-SRTF*, a *light-weight yet effective* DCN scheme to implement SRTF scheduling in a distributed manner. *D-SRTF* solves aforementioned problems as follows.

- *D-SRTF* utilizes the bytes sent of each flow to estimate the remaining flow size, which is light-weight.
- *D-SRTF* implements DCTCP [4] as the congestion control scheme (which is already deployed by many production data centers [13], [14]). In DCTCP, each sender maintains a congestion indicator to estimate the congestion extent of the network for congestion control. *D-SRTF* leverages this congestion indicator in DCTCP to estimate the available bandwidth.
- *D-SRTF* calculates the remaining time of each flow as the remaining flow size over the available bandwidth. At each sender, *D-SRTF* sets a fixed set of thresholds for flow remaining time, and assigns priorities to packets based on the remaining time and thresholds. The less remaining time a flow has, the higher priority is assigned to packets of the flow.
- In each switch port, *D-SRTF* sets multiple queues with different priorities (as shown in Fig. 4c), and packets are enqueued into the queue that matches the *priority* in the packet header. *D-SRTF* implements *strict priority* scheduling among these queues to deliver packets with the highest priority first. In this case, packets with less remaining time will be delivered first in general, thus *D-SRTF* realizes SRTF scheduling in a distributed manner.

We also conduct extensive experiments to show the performance of *D-SRTF*. We first run a web search workload [4] and a data mining workload [15] separately on one of the most widely deployed topologies, Leaf-Spine topology, to compare *D-SRTF* with PIAS [7] which is currently the best deployment friendly scheme that mimics SJF scheduling. Then we add two more workloads, a cache workload [16], a Hadoop workload [16], and run four workloads together to explore the performance of *D-SRTF* in heterogeneous workloads. Experiment results shown that *D-SRTF* performs better than PIAS and other schemes. For example, *D-SRTF* improves the FCT of short flows for web search workload by 8∼13 percent and improves the 99th percentile by 12∼33 percent, when compared to PIAS, and in data mining workload, *D-SRTF* has only 1.6 percent performance gap to pFabric, which is currently the best (but not supported by existing commodity switches) DCN scheme, and if information on flow size is available which is the case for pFabric, *D-SRTF* could perform better than pFabric.

Besides, we also implement some targeted experiments to explore the properties of *D-SRTF*, and the experiment results show that *D-SRTF* is robust to different network settings, and the estimations about remaining flow size and available bandwidth are effective.

The rest of this paper is organized as follows. Related works are summarized in Section 2. We introduce the motivation for *D-SRTF* in Section 3. We describe the algorithm design in detail, and analyze the properties of *D-SRTF* in Section 4. Extensive experiments are conducted to show the superior performance of *D-SRTF* in Section 5. Finally the paper is concluded in Section 6.

2 Related Works

Lots of works have been proposed using effective scheduling schemes, to decrease FCT, which is always the motivation to improve DCN performance.

DeTail [17] prioritizes latency-sensitive flows and distributes network load, which can reduce the FCT tail. PDQ [1] leverages switch arbitration and performs flow scheduling in a distributed manner, based on explicit rate control, so as to reduce FCT and meet deadlines. L2DCT [2] considers bytes sent information and approximates the Least Attained Service (LAS) scheduling scheme, which reduces FCT. Karuna [18] considers the scenarios of mixed flows with and without deadlines, and proposes a systematic solution which sets minimum bandwidth for deadline flows to meet their deadline just in time and leaves remaining bandwidth to deadline-free flows, which satisfies the requirements of both types of flows. PASE [6] synthesizes self-adjusting endpoints, in-network prioritization and arbitration for better performance. pFabric [3] is theoretically the best DCN scheme, which performs shortest remaining size first scheduling. pFabric requires packets to carry their priority number, and switches implement priority-based scheduling scheme. Fastpass [19] proposes a scalable timeslot allocation algorithm to determine when to send packets at endpoint. [20] discusses about using multiple layers of queueing

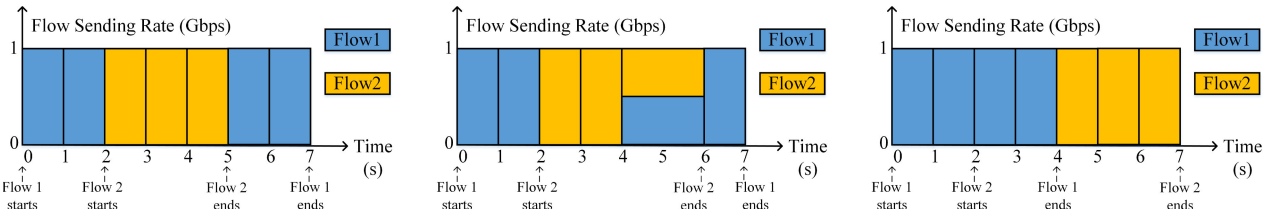


Fig. 1. SRTF versus SJF scheduling.

in intermediate servers and network switches. [21] utilizes Least Slack Time First (LSTF) scheduling to approximate the state-of-the-art works which achieves various goals like reducing average FCT, tail latency and ensuring fairness.

However, most of existing works assume that the flow size is known at the flow start, which is not true for many applications. PIAS [7] is proposed which mimics the shortest job first discipline on the premise that flow size is unknown. PIAS tags packets according to the bytes sent and leverages multiple priority queues in switches to prioritize short flows, which achieves good results. Aemon [22] considers the information-agnostic scenario with flow size unknown while there are some flows with deadlines and some without, and proposes a new mix-flow transport based on flow urgency.

While PIAS, pFabric and other works achieve good results, they fail to consider the varying available bandwidth [23] for flow scheduling, which leads to inferior performance. Our *D-SRTF* differs from other works in that we consider the available bandwidth to calculate the remaining time explicitly in order to realize SRTF scheduling, and *D-SRTF* is also on the premise that flow size is unknown, which applies to the information-agnostic scenario.

Some other works use different scheduling schemes for different DCN scenarios. For example, MQ-ECN [9] utilizes round-robin schedulers to ensure the fairness among different services in DCN, which sets a dynamic threshold for the queue length of the traffic for each service in switch port. TCN [8] generalizes MQ-ECN so that it can support different kinds of schedulers like round-robin schedulers, weighted fair queueing scheduler and strict priority scheduler. DEME [10] decouples packet marking from enqueuing to ensure service level fairness in the multiple-service multiple-queue scenario. DemePro [11] extends DEME [10] with proactive congestion control, which further improves the DCN performance. Falloc [24] proposes a fair network bandwidth allocation scheme based on a cooperative game approach, which can achieve the bandwidth fairness while balancing the tradeoff between bandwidth guarantee and proportional bandwidth sharing. TATS [25] proposes a time-aware task scheduling (TATS) algorithm which investigates the temporal variation to schedule task. Like TATS, our work also considers the bandwidth variation to determine the flow priority for scheduling. There are also some recent works [26], [27], [28], [29], [30], [31] to improve the co-flow performance in DCN. For example, Varys [27] proposes a co-flow scheduling algorithm based on concurrent open shop scheduling with coupled resources problem, and solves the inter-coflow scheduling with effective heuristics. D-CLAS [26] considers the problem of unknown co-flow information, and proposes Discretized Coflow-Aware Least-Attained Service (D-CLAS) which separates co-flows to different priority queues and schedules them based on the bytes they have sent.

3 Motivation

In this section, we first leverage case studies to show the advantages of SRTF scheduling over SJF scheduling and SRSF scheduling, which shows the importance of using remaining flow size and the available bandwidth for flow scheduling. Then we discuss about the difficulties of applying SRTF scheduling in practice, which motivates our *D-SRTF* design.

3.1 Case Studies

Consider a simple topology in Fig. 2. This "many to one" pattern mimics many typical data center applications. For examples, (1) in storage clusters, many storage nodes respond to a data query from a single node; (2) in web search, servers respond to a query submitted by one client; or (3) in MapReduce [32], multiple Mappers transfer key-value pairs to the same Reducer [33].

3.1.1 SRTF versus SJF

In this case study, we have two TCP senders and one TCP receiver. Each of the two senders is sending one TCP flow to the receiver through the switch. Assume the size of Flow 1 (from Sender 1) is 4 Gb, and the size of Flow 2 (from Sender 2) is 3 Gb. All link capacity is 1 Gbps, and Flow 1 starts at time 0, and Flow 2 starts 2 seconds later. The link from the switch to the receiver will become the bottleneck, thus the scheduling in the switch will have significant impact on the performance of networking, i.e., average flow completion time. The scheduling results of different schemes are shown in Fig. 1. Note that, for both SRTF and SJF, each flow can use the whole link capacity, so this comparison shows the superiority of using remaining flow size instead of total flow size for flow scheduling.

For *information aware* SJF (in Fig. 1a), flow sizes are known. When Flow 2 starts, scheduler knows that Flow 2 is shorter than Flow 1, thus stops Flow 1 and schedules Flow 2 first. As a result, FCTs of Flow 1 and Flow 2 are 7 s and 3 s, thus the average FCT is 5 s.

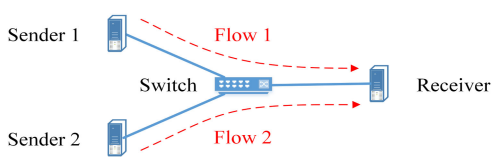


Fig. 2. Topology for case studies.

For *information agnostic* SJF (in Fig. 1b), flow sizes are unknown (PIAS [7] implements the *information agnostic* SJF scheduling which always first schedules the flow with *less bytes sent*). Thus, when Flow 2 starts, scheduler stops Flow 1 and schedules Flow 2 first (the bytes sent of Flow 1 are 2 Gb while the bytes sent of Flow 2 are 0). But when Flow 2 transmits for 2 seconds, its bytes sent become 2 Gb, the same as Flow 1. So next, Flow 1 and Flow 2 will share the bandwidth, and each flow has 0.5 Gbps bandwidth. This is because, once Flow 1 is scheduled to deliver packets, Flow 2 will become the one with less bytes sent and scheduler has to stop Flow 1 to transmit Flow 2. It also applies if Flow 2 is scheduled first. This "fair sharing" lasts for 2 seconds until Flow 2 finishes. Then Flow 1 will utilize the whole link capacity to transmit the remaining 1 Gb. As a result, FCTs of Flow 1 and Flow 2 are 7 s and 4 s, thus the average FCT is 5.5 s.

For SRTF (in Fig. 1c), when Flow 2 starts, scheduler knows that Flow 1 has less remaining time ($2/1 = 2s$) than Flow 2 ($3/1 = 3s$), so it will first transmit the remaining 2 Gb of Flow 1, then transmit Flow 2. As a result, FCTs of Flow 1 and Flow 2 are 4 s and 5 s, thus the average FCT is 4.5 s, which is shorter than both *information aware* SJF and *information agnostic* SJF.

Remark 1. It is better to use the remaining flow size (instead of total flow size) for scheduling.

3.1.2 SRTF versus SRSF

In this section, we leverage case study to show the performance difference between SRTF scheduling and SRSF scheduling (pFabric [3] utilizes SRSF scheduling). Note that SRTF considers both the remaining flow size and the available bandwidth, while SRSF only considers the remaining flow size. So the case study in this section shows the impact of available bandwidth on the performance of scheduling schemes (we also validate this in Section 5.6.3).

In this case study, we also leverage the topology in Fig. 2. All link capacity is 1 Gbps. Flow 1 (4 Gb) starts at time 0 and Flow 2 (3 Gb) starts at time 2 s. At the same time when Flow 2 starts, the available bandwidth between sender 1 and the switch drops to 0.5 Gbps, and this condition lasts for 3 seconds, and after 3 s, the available bandwidth on this link rises back to 1 Gbps. This mimics many DCN scenarios, like (1) DCN system administrator reserves some bandwidth for special usage, like delivering the control traffic in SDN area [8], (2) in multiple-service scenario [8], [9], [10], [11], when traffic from different services coexists in the network, each service needs to be limited to a certain amount of bandwidth (a portion of the link capacity) to ensure service-level fairness.

The scheduling results of SRTF and SRSF are shown in Fig. 3.

For SRSF (in Fig. 3a), when Flow 2 starts, Flow 1 has transmitted 2 Gb and has 2 Gb remaining, while Flow 2 has 3 Gb remaining. Thus, SRSF scheduler will schedule Flow 1 first. However, the available bandwidth for Flow 1 is only 0.5 Gbps, so Flow 2 will share another 0.5 Gbps bandwidth. This sharing lasts for 3 seconds until the available bandwidth for Flow 1 rises back to 1 Gbps. Then Flow 1 has 0.5 Gb remaining and Flow 2 has 1.5 Gb remaining, thus SRSF scheduler will schedule Flow 1 first using 1Gbps bandwidth. After 0.5 s, Flow 1 finishes, and Flow 2 will use 1 Gbps bandwidth to transmit the remaining 1.5 Gb, which lasts for 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.

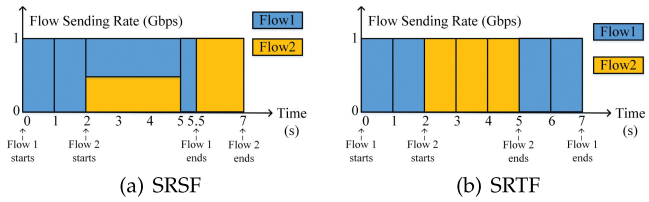


Fig. 3. SRTF versus SRSF scheduling.

For SRTF (in Fig. 3b), when Flow 2 starts, Flow 1 has transmitted 2 Gb and has 2 Gb remaining, while Flow 2 has 3 Gb remaining. Since the available bandwidth for Flow 1 is only 0.5 Gbps, while the available bandwidth for Flow 2 is 1 Gbps, thus the remaining times for Flow 1 and Flow 2 are (2/0.5=4) s and (3/1=3) s. Therefore SRTF scheduler will schedule Flow 2 first, which will end 3 s later.[1] When Flow 2 finishes, the available bandwidth for Flow 1 rises to 1 Gbps, thus Flow 1 will need another 2 s to transmit the remaining 2 Gb. As a result, FCTs of Flow 1 and Flow 2 are 7 s and 3 s, thus the average FCT is 5 s, which is shorter than that achieved by SRSF.

Remark 2. It is important to know the available bandwidth for scheduling.

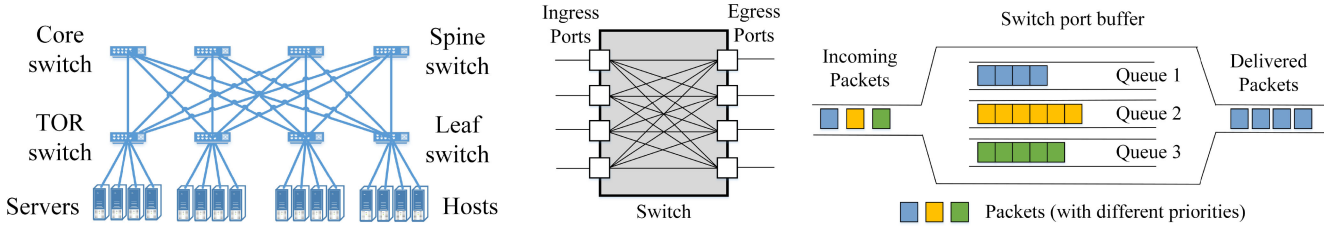
3.2 Discussion

In previous section, we have shown that it is better to utilize the remaining flow size and the available bandwidth for flow scheduling, and if information aware, a centralized scheduler which knows the global state could implement SRTF scheduling that achieves better performance than SRSF and SJF. However, it is difficult to implement the ideal SRTF in practice, due to the following reasons.

Unknown Remaining Flow Sizes. In many cases, the flow size is not available. For example, (1) for chunk transfer in HTTP 1.1 [34], each flow consists of multiple chucks which are generated dynamically, so the total flow size is unknown at the transmission start. (2) in database query response, partial query results are transferred before all results are obtained [12], thus flow size is also unknown at the beginning of data transfer.

Difficulty in Measuring Available Bandwidth. In multi-layer DCN topology, each flow has to traverse multiple links from the sender to the receiver, and among all these links, any one could become the bottleneck for flow transmission. Thus, to know the available bandwidth of each flow, we need to measure the available bandwidth of all links that the flow passes, and find the smallest one (bottleneck) as the available bandwidth for this flow. The measurement overhead is very high. Besides, since DCN is dominated by short flows and traffic burst is inevitable [7], available bandwidth may change frequently and dramatically, which further increases the difficulty of bandwidth measurement.

1. Assume that at time 2 s, the scheduler knows that the available bandwidth for Flow 1 will rise back to 1 Gbps after 3 seconds, then the remaining time for Flow 1 is $3 + (2 - 0.5 * 3)/1 = 3.5$ s, which is still longer than that for Flow 2. So SRTF scheduler will also schedule Flow 2 first.



(a) DCN architecture: Leaf-Spine topology (b) Switch architecture (c) Priority queues in switch port buffers

Fig. 4. System model.

Shortcomings of Centralized Scheduler. To implement ideal SRTF scheduling, the scheduler should collect the remaining time statistics of all flows. Thus a single centralized scheduler is necessary, so that it can compare the remaining time of all flows and schedule the flow with the shortest remaining time first. However, in practise, such centralized scheduler would incur high transmission and computation overhead, since each sender has to send the remaining time information to the centralized scheduler, which costs bandwidth, and the scheduler has to compare the remaining time of all flows to find the shortest one, which is computation-intensive and could delay the scheduling decisions. Thus, a distributed scheduler is preferred to alleviate aforementioned problems, which reduces the transmission cost and computation overhead.

In next section, we solve all these problems by proposing *D-SRTF* scheme.

4 *D-SRTF* Design

4.1 System Model

While the real-world DCN system is much more complex, we outline the architecture and simplify the model as in Fig. 4, which is widely used in existing works like [2], [4], [7], and can capture the essential elements of a real-world DCN for evaluating the performance of the proposed scheme. More specifically, we leverage a DCN architecture shown in Fig. 4a, which is one of the state-of-the-art topologies, Leaf-Spine topology [35]. Servers (hosts) are interconnected by layers of switches that form the network fabric. Each switch has multiple ports[2] to deliver packets, as shown in Fig. 4b. There are multiple queues in each switch port buffer, as shown in Fig. 4c, and these queues have different priorities (in the example in Fig. 4c, Queue 1 has the highest priority and Queue 3 has the lowest). Packets are classified and enqueued to corresponding queues, and packets in the highest priority queues will be delivered first.

4.2 Design Rationale

Note that scheduling without flow size information is non-clairvoyant scheduling [36], and Least Attained Service is effective in reducing the average FCT [37]. LAS estimates the remaining time of a job via the time the job has attained. LAS is also effective in DCN area, since the sizes of flows follow the heavy-tailed distributions [38], [39], i.e., the majority of flows are short flows, but the majority of bytes come from a small number of long flows.

2. We classify ports into *ingress ports* and *egress ports* for better explanation for their functions.

D-SRTF is partially motivated by LAS in estimating the remaining time of each flow, but could not directly implementing LAS in practice. The reasons are explained as follows.

First, each packet needs to carry the remaining time information in packet header for switches to compare, to perform scheduling. However, to store *accurate* values of remaining time, a large number of reserved bits in packet header should be utilized, which is not expected.

Second, without centralized scheduler, each switch needs to perform scheduling independently. Assume each packet can carry the remaining time information. Thus when each packet enqueues, the switch needs to compare the remaining time of each flow for scheduling, but this is not supported in current commodity switches [12].

D-SRTF solves above two problems with the following two techniques.

First, *D-SRTF* classifies flows into several fixed categories according to the remaining time of each flow, and tags each sent packet based on the classified categories. Thus, instead of carrying the accurate flow remaining time, *D-SRTF* lets packets to carry the category ID (or say, priority) for switches to schedule. Much fewer bits in packet headers are required to carry the category ID. For example, to denote 8 categories, 3 bits are enough ($2^3 = 8$).

Second, *D-SRTF* utilizes the multiple priority queues in current commodity switches, which currently support up to 8 queues per switch port [3]. *D-SRTF* sets different priorities to different queues, and enqueues the packets to corresponding queues according to the priority (category ID) of each packet. Then *D-SRTF* performs strict priority scheduling to realize SRTF scheduling.

Consequently, in general, flows with less remaining time have higher priority tagged (by TCP senders) in packet header, and are enqueued into higher priority queues in switch ports, thus are delivered first, therefore SRTF scheduling is realized.

However, some challenges need to be solved to make *D-SRTF* work. (1) How to classify flows into different categories? The common practice is to set several fixed thresholds for the value of remaining time, like PIAS [12]. However, PIAS determines the thresholds based on the flow size distributions of individual workloads, thus any set of thresholds for one workload is less effective for another workload. Therefore, a general version of thresholds are expected. (2) How to make sure that *D-SRTF* is compatible with legacy TCP/IP stacks?

In next sections, we will describe the detailed mechanism of *D-SRTF* to address all above challenges.

4.3 Detailed Mechanism

At the core, *D-SRTF*'s mechanism consists of remaining time calculation, priority setting and switch design.

4.3.1 Remaining Time Calculation

To calculate the remaining time of each flow, we need to estimate the remaining size of the flow and the available bandwidth. Note that, we expect the estimation technique to be light-weight without incurring high overhead. Thus, estimating flow size using complex predicting algorithm and measuring link utilization to calculate the available bandwidth are beyond our consideration in this paper.

Estimating Remaining Flow Size. Many recent works [4], [15], [16] have shown that the flow sizes of DCN traffic follow heavy-tailed distribution, which means the majority of flows in DCN are short flows but the majority of bytes come from a small number of long flows. Therefore, without flow size information, it is still possible to use the number of bytes sent to distinguish between short flows and long flows.

For short flows, both the number of bytes sent and the number of remaining bytes are small numbers and their difference is also small. So, the number of bytes sent could be a close approximation of the remaining flow size during their short lifetime. In fact, both favor short flows that can complete quickly. Note that it is impossible for short flows to have a large number of bytes sent. So, it is impossible to misclassify short flows as long flows. In contrast, it is possible to misclassify long flows as short flows in the very beginning of their lifetime. However, the impact on other real short flows is insignificant because the number of long flows is very small compared to the number of short flows and the duration of misclassification is also short. As the number of bytes sent is increasing, the difference between the number of bytes sent and the remaining flow size is diminishing. Although, after a certain period of time, the number of bytes sent could exceed the remaining flow size, the impact of this difference becomes insignificant near the end of long flows because the number of long flows is small such that the competition among long flows becomes very light.

Thus, the bytes sent of each flow is a good candidate to estimate the remaining flow size. In this case, we can use the number of bytes sent as the estimated remaining size, and the number of bytes sent can be easily counted by TCP senders when packets are delivered.

Estimating Available Bandwidth. At TCP senders, *D-SRTF* leverages the feedback from network to estimate the available bandwidth. Note that the majority of existing works [2], [7], [8], [9] implement DCTCP [4] for congestion control. That is, if the queue length exceeds a threshold, switches will mark packets using ECN [40] marking scheme, and if the TCP receivers receive marked packets, they will also mark the returned ACKs. When TCP senders receive the ACKs, they will calculate the percentage of marked ACKs corresponding to the last window of packets, and estimate the network congestion extent, and perform congestion control accordingly, as follows:

$$\alpha \leftarrow (1-g) \times \alpha + g \times F \tag{1}$$

$$CWND \leftarrow CWND \times (1 - \alpha/2), \tag{2}$$

where α (in $[0, 1)$) is the estimator of network congestion extent, g (typically set to 0.0625) is the weight given to new samples against the previous estimation of α, F is the *fraction* (or percentage) of marked ACKs corresponding to the last window of packets, and $CWND$ is the congestion window size.

We find that α is suitable for estimating the available bandwidth. This is because α reflects the congestion extent of the network. According to Eqn (1), α is calculated based on F, the fraction of marked packets. Basically, a packet is marked only when the total queue length exceeds the threshold. Therefore, a small value of α indicates small percentage of packet marking and short time during which the queue length exceeds the threshold, thus low network congestion extent. When $\alpha = 0$, no packet is marked, that is, the queue length is shorter than the threshold, or there is no packet in the queue (even though some bandwidth has been utilized). In a distributed networking system, a TCP sender is agnostic to the network dynamics, so it probes the network with the additive-increase/multiplicative-decrease (AIMD) algorithm to update the window size. Therefore, when there is no packet marking, from the TCP sender's point of view, the link is "empty" and the sender can additively increase the window size according to the AIMD algorithm employed by current TCP. A positive value (non-zero) of α indicates the network experiences some extent of congestion. For example, $\alpha = 0.2$ means, on average (according to exponential weighted moving average (EWMA) in Eqn (1)), 20 percent of packets are marked inside the network, indicating network congestion, while 80 percent of packets are not marked, therefore it is safe to utilize 80 percent of the network capacity. When $\alpha = 1$, all packets are marked, meaning that the queue length always exceeds the threshold, indicating highest network congestion extent and no available bandwidth, thus congestion control has to be performed aggressively at end host.

As a result, D-SRTF estimates the available bandwidth as $B \leftarrow L \times (1 - \alpha)$ where B is the estimated available bandwidth and L is the link capacity.

Then, we can calculate the remaining time by dividing remaining flow size over the available bandwidth.

4.3.2 Priority Setting

Given the remaining time of each flow, we need to set the priority for each packet to classify each flow to different categories. Since current commodity switches support up to 8 queues in each switch port, we set 8 priorities for flow scheduling, from 0 to 7, to match the number of queues, where 0 denotes the *highest* priority and 7 denotes the *lowest*. In this case, we need 7 thresholds.

Motivated by [26], we utilize the following exponential thresholds:[3]

$$T_i \leftarrow T_0 \times E^i, i \in [0, 6], \tag{3}$$

3. PIAS [7] formulates the threshold setting problem as a Sum-of-Linear-Ratios (SoLR) problem [41], and uses M/M/1 queues to simplify the analysis and gets a closed form solution. Since the formulation considers the traffic load and flow size distribution, each set of thresholds can only achieve good results for the given traffic load and flow size distribution, but not for other traffic loads or flow size distributions.

where T_i is the ith threshold, and E determines how much the remaining time of one category is longer than that of another. In this paper, we use $T_0 = 120\mu s$ and $E = 3$, which are the best combinations according to our extensive experiments. We find that this threshold setting works well in our experiments with practical production workloads. Thus,

- if the remaining time of a flow is smaller than T_0, the priority is set to 0;
- if the remaining time of a flow is larger than or equal to T_6, the priority is set to 7;
- otherwise, the priority is set to $(i + 1)$, if the remaining time is in $[T_i, T_{i+1})$.

Given the priority of each flow, TCP senders set the priority to the Differentiated Services Code Point (DSCP) field in the IP header. Thus, when the packet reaches the switch port, it will be enqueued into the queue that matches the packet's DSCP field.

Note that this set of thresholds are distributed to all TCP senders, thus *D-SRTF* relieves switch's burden of implementing SRTF by comparing the remaining time, which is not supported by current commodity switches. In this case, *D-SRTF* works well in a distributed manner.

4.3.3 Switch Design

With packets enqueued into distinct queues according to their priorities which reflect the remaining time, *D-SRTF*'s switches perform *strict priority* scheduling, which is a built-in function of existing commodity switches [6]. In this case, packets in the highest priority queues are scheduled first, i.e., flows with the shortest remaining time will be delivered first.

Besides, when the queue length in each switch port buffer exceeds the queue length threshold (as that in DCTCP), we also perform ECN marking on subsequent packets, which has two major functions in *D-SRTF*. First, by marking packets and performing congestion control at senders, the queue length can be kept low, which alleviates the risk of buffer congestion and overflow. Second, by marking packets, TCP senders can obtain the feedback from network to know the network congestion extent, which is useful for estimating the available bandwidth, as described in Section 4.3.1.

4.4 Analysis and Discussion

In this section, we discuss the implementation issues and some properties of *D-SRTF*.

Implementation Compatible with Commodity Switches and Legacy TCP/IP Stacks. All operations in *D-SRTF* are supported by existing commodity switches and TCP/IP stacks, in that, (1) bytes sent can be tracked as the data is delivered, consistent with [7]; (2) the congestion extent estimator α is updated per window of packets, which is supported and widely utilized by the majority of recent schemes implementing DCTCP; (3) thresholds are distributed to all end hosts, and by comparing the remaining time and the thresholds, the priority can be determined for each packet, and is set into the DSCP field in the packet header; (4) each switch sets a threshold for the port buffer queue length, and performs ECN marking on packets when the queue length exceeds the threshold, which are supported by DCTCP and have already been widely deployed in many production data centers [13], [14].

Local Decision. According to [3], in terms of minimizing FCT, SRTF is optimal in single link, but sub-optimal in multiple links, and there is no simple optimal scheme for simultaneously scheduling flows across multiple links. Therefore, like PIAS [7] and pFabric [3], *D-SRTF* makes local decisions on each switch. These local decisions work well in most cases [3], and only experience some performance loss at high load, like, over 90 percent [6], while in most cases, DCN is operating at moderate loads, like, 30 percent [42]. Our experiments in Section 5.2 confirm that *D-SRTF* works well in practice.

Network Congestion Aware Scheduling. Although both *D-SRTF* and PIAS implement strict priority scheduling in switches, they mimic different scheduling schemes. PIAS only considers the flow size (bytes sent) information and mimics shortest job first scheduling. On the contrary, *D-SRTF* mimics shortest remaining time first scheduling, so *D-SRTF* considers not only the flow size attribute, but also the state of the network (i.e., the available bandwidth). The consideration of network state makes *D-SRTF* better react to the network dynamics. Therefore, (1) if one flow encounters congestion, its available bandwidth is low, so the estimated remaining time is high, thus the priority is low. In this case, the flow could be enqueued to lower priority queue in switches, allowing other flows in higher priority queues (or say, with less remaining time) to be delivered first. (2) On the contrary, if the flow has large amount of available bandwidth, its remaining time is low and its priority is high, and will be enqueued to higher priority queues in switches and will be delivered first. These could not be realized by the scheduling schemes which only consider flow size information but ignore the available bandwidth, like PIAS which mimics shortest job first scheduling or pFabric which adopts shortest remaining flow size first scheduling.

5 Evaluation

In this section, we conduct extensive experiments in NS2 [43] to evaluate the performance of *D-SRTF*. We mainly compare *D-SRTF* with the following three deployment-friendly DCN schemes:

- DCTCP [4] which adopts a simple First in First out (FIFO) scheduling.
- L2DCT [2] which approximates the Least Attained Service scheduling.
- PIAS [7] which mimics shortest job first scheduling.

We compare all four schemes in average Flow Completion Time. To ensure fair comparison between these four schemes, we employ the same settings as those in the PIAS paper, like the same Leaf-Spine topology, the same workloads and so on. Note that PIAS relies on the flow size distribution of each workload to determine the thresholds for priority setting. Thus, for fair comparison, we first compare *D-SRTF* with PIAS (and DCTCP and L2DCT) in two separate network environment settings, each with one distinct workload. Then we compare all schemes in a mixed workload environment, so as to evaluate the robustness of *D-SRTF*. Our experiments need to answer the following key questions:

How does D-SRTF Perform in Reducing FCT? Generally, *D-SRTF* performs better than all other three schemes, and improves the overall average FCT by 2∼11 percent

TABLE 1
Default Parameter Setting

Parameter	Value
Link capacity	10 Gbps
Packet size	1.5 KB
Port buffer size	200 packets
Queue length threshold for port buffer	65 packets
g (in Eq. (1))	0.0625
T_0 (in Eq. (3))	120 μs
E (in Eq. (3))	3

compared to PIAS. *D-SRTF* also improves the performance for short flows. For example, *D-SRTF* improves the FCT for short flows by 8∼13 percent and improves the 99th percentile by 12∼33 percent, when compared to PIAS in web search workload. For long flows, *D-SRTF* performs similar with PIAS.

How does D-SRTF Perform in Individual Workload and Mixed Workloads? Generally, *D-SRTF* performs better than PIAS (and DCTCP and L2DCT) in individual workload. In 'the mixed workload scenario, the performance improvement is also significant.

How does D-SRTF Perform in Different DCN Topologies? We evaluate the performance of *D-SRTF* on both Leaf-Spine and Fat-tree topologies, and the experiment results show that *D-SRTF* outperforms other schemes in both network topologies.

How does D-SRTF Perform Compared with Ideal Information-aware Schemes? D-SRTF has only 1.6 percent performance gap to the ideal information-aware scheme, pFabric [3], in FCT of short flows, while PIAS has 4.9 percent performance gap.

At last, we also conduct several sets of targeted experiments to explore the properties of *D-SRTF*, and the results show that *D-SRTF* is robust to different network settings, like different numbers of queues in switch ports, and *D-SRTF*'s estimations for remaining flow size and the available bandwidth are effective.

5.1 Experiment Setup

Parameters. Unless stated explicitly, relevant parameters are set to the default values shown in Table 1. For PIAS and L2DCT, all relevant parameters are set to the default values in their original papers [2], [7].

Network Topology. Experiments are running on Leaf-Spine topology shown in Fig. 4a. In our experiments, we have 12 Leaf (Top-of-Rack (ToR)) switches, 12 Spine (Core) 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. We utilize the widely used ECMP [44] for routing and load balancing in the multi-path environment. Currently, the widely used commodity switches support up to 8 queues per port, so in our experiments, we set 8 queues for each switch port.

Workloads. In this paper, we run four different workloads for experiments, including a web search workload [4], a cache workload [16], a data mining workload [15] and a Hadoop workload [16], and the respective flow size distributions are shown in Fig. 5. For each workload, the inter-flow arriving interval t is inversely proportional to the network *Load*, i.e.,

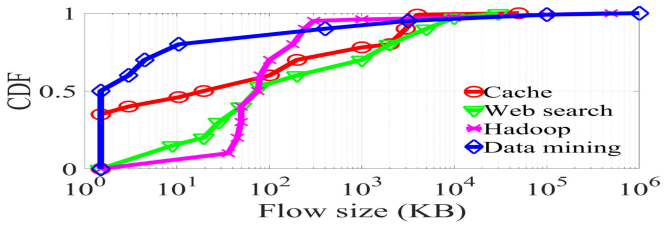


Fig. 5. Flow size distributions for individual workload.

$$\ell = \frac{C \times Load}{Mean\ Flow\ Size}, \quad t = \frac{1}{\ell}, \tag{4}$$

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.

For fair comparison with PIAS which relies on the flow size distribution to determine the flow demotion thresholds, we first run the web search workload and the data mining workload separately based on above experiment settings, as that in the PIAS paper. Each of the experiments lasts for 50,000 flows. Then, to mimic real scenario, we mix all four workloads together to evaluate the practical performance, and we also generate totally 50,000 flows. For each workload, the inter-flow arriving interval is proportional to its 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. Table 2 shows the mean flow size and the number of flows for each workload, in the mixed workload scenario.

Congestion Control. D-SRTF leverages the congestion control scheme of DCTCP at TCP senders. PIAS also utilizes the congestion control scheme of DCTCP, while L2DCT extends DCTCP to update the window size according to the bytes sent of each flow, thus mimics LAS. We set initial and minimum value of TCP RTO to 5ms for all schemes, as many recent works recommend [8], [9], [13]. Initial TCP window size is set to 10 packets.

Comparison Metrics. According to the flow sizes, we divide all flows into three classes including short flows ((0,100] KB), medium flows ((100 KB,10 MB]) and long flows ((10 MB,∞)), and compute the average FCT for each class. For example, in the mixed workload scenario, the numbers in the brackets in Table 2 denote the numbers of flows in each class, e.g., cache workload has 26,505 flows in total, among which there are 16,034 short flows, 10,253 medium flows and 218 long flows. We compare the overall average

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

Workload	MFS (KB)	Number of Flows (S/M/L)
Cache	914	26,505 (16,034/10,253/218)
Web Search	1,671	14,453 (7,881/6,180/392)
Hadoop	4,149	5,829 (4,131/1,523/175)
Data Mining	7,495	3,213 (2,638/420/155)

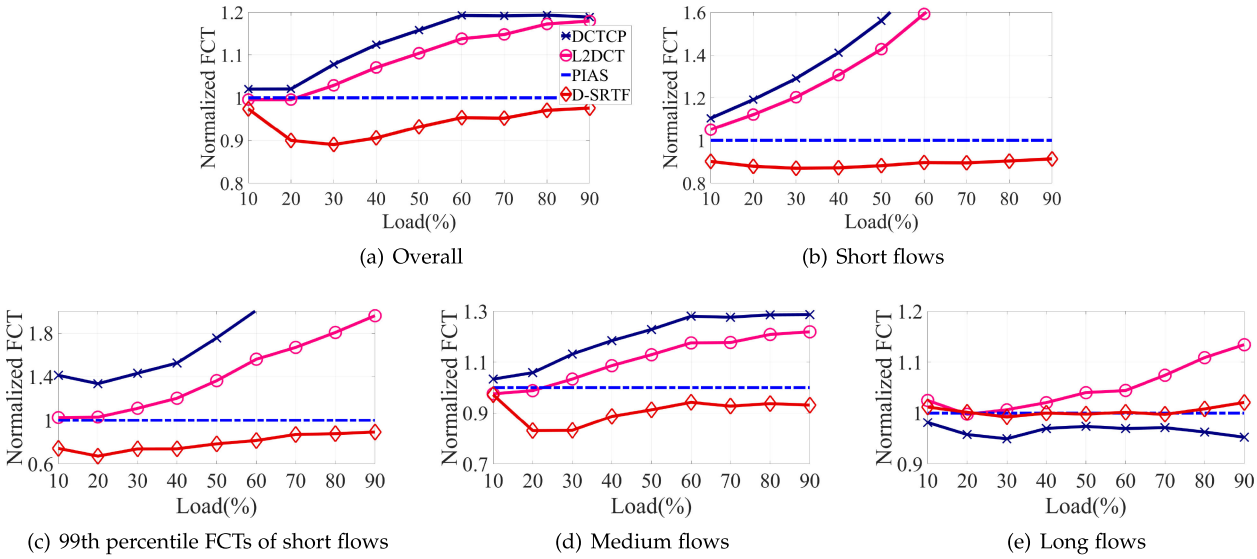


Fig. 6. Comparison of FCT for web search workload.

FCT for all flows, the average FCT of flows in each class, and the 99th percentile FCTs of short flows.

5.2 Performance in FCT

We conduct experiments for the four schemes based on the aforementioned experiment setup, and the comparison results are shown in Figs. 6 and 7. For the sake of easy comparison of performance with reference to PIAS, which is one of the best-performing schemes in the literature, we normalize all results to those achieved by PIAS. For example, if the FCTs for DCTCP, L2DCT, PIAS and D-SRTF are 2.8s, 2.4s, 2s and 1.6s, the normalized FCTs are 1.4, 1.2, 1 and 0.8 respectively (normalized FCTs for PIAS are always 1). According to the comparison results, we have the following observations:

Overall. As shown in Figs. 6a and 7a, *D-SRTF* generally performs best across all loads. For example, for web search workload, *D-SRTF* has about 11 percent lower FCT at 30 percent load and 2 percent lower FCT at 90 percent load, when compared with PIAS. When compared with DCTCP and L2DCT, the performance improvement is more significant. For example, for web search workload, *D-SRTF* has average 20 percent lower FCT than L2DCT and 25 percent lower FCT than DCTCP. For data mining workload, *D-SRTF* also outperforms all other three schemes.

Short Flows. *D-SRTF* performs better than PIAS, and greatly outperforms L2DCT and DCTCP for short flows. For example, for web search workload, *D-SRTF* improves the FCT for short flows by 8∼13 percent and improves the 99th percentile by 12∼33 percent, when compared to PIAS. Since short flows dominate the data traffic in DCN and are more delay-sensitive, it is more important and meaningful for *D-SRTF* to improve the performance of short flows without jeopardizing the performance of long flows much [7]. For data mining workload, the performance improvement is less significant. This is expected, since the data mining workload is more skewed. In fact, 80 percent of the flows in data mining workload are smaller than 10 packets (which is the initial TCP

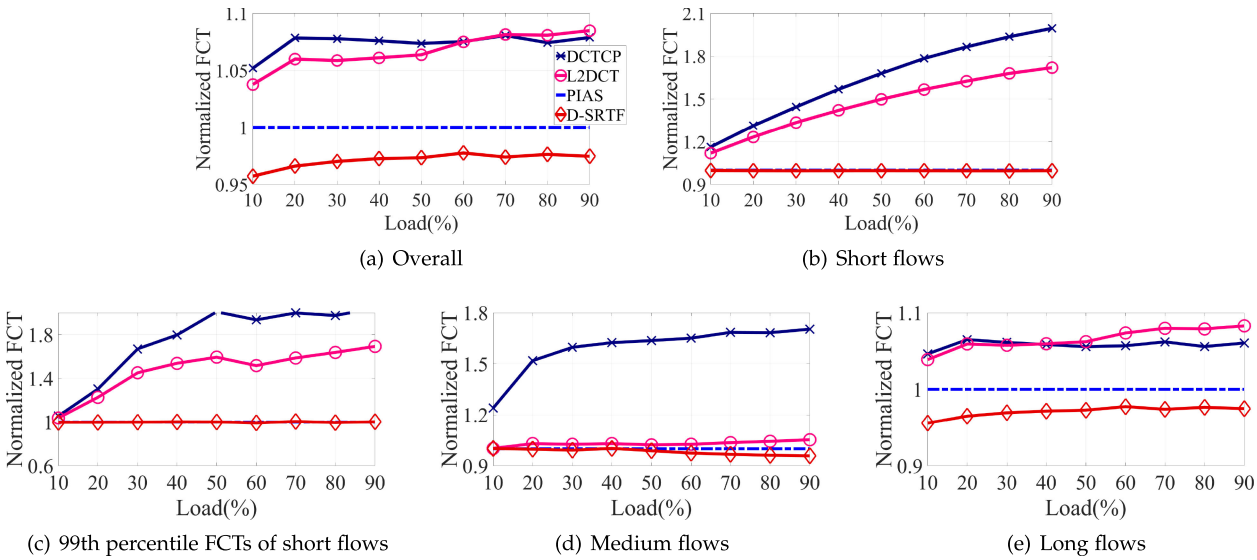


Fig. 7. Comparison of FCT for data mining workload.

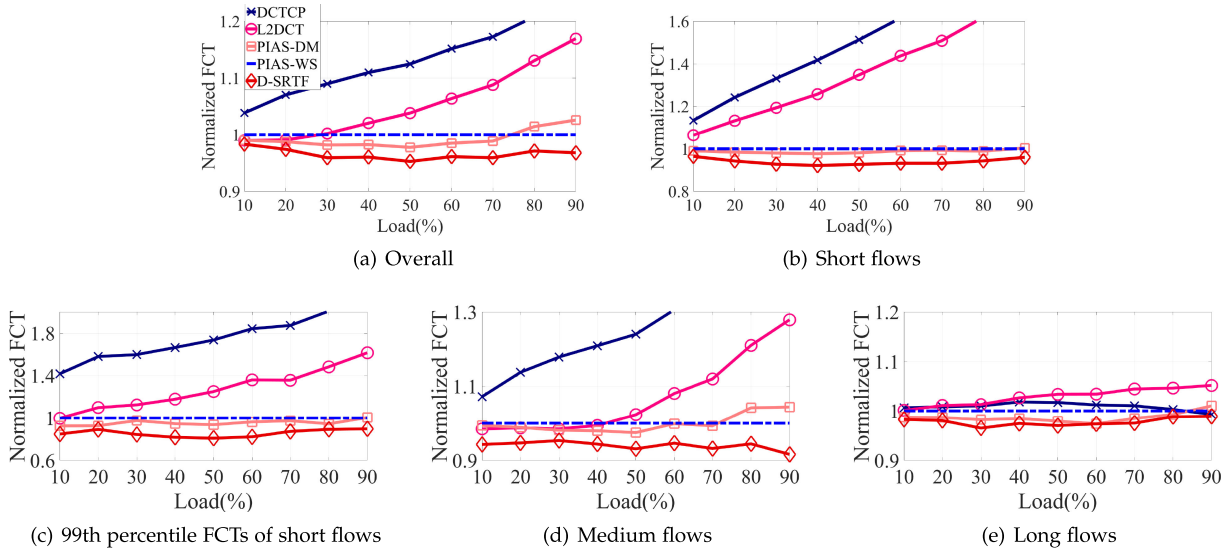


Fig. 8. Comparison of FCT in mixed workloads. PIAS-DM denotes the PIAS scheme with the demotion thresholds for data mining workload, and PIAS-WS represents the PIAS scheme with the demotion thresholds for web search workload.

window size), thus they finish transmission in the first window. Since both PIAS and *D-SRTF* set the priority of packets in the first window to the highest, therefore, their performance is similar in short flows in data mining workload.

Medium Flows. Although the advantage of *D-SRTF* diminishes for medium flows, it still manages to outperform PIAS by about 10 percent for web search workload and 4 percent for data mining workload, in terms of average FCT. The improvement towards DCTCP and L2DCT is more significant.

Long Flows. In general, *D-SRTF* performs similar to PIAS in long flows. For web search workload, *D-SRTF* performs slightly worse than PIAS at high load. This is because, web search workload is less skewed, so it is likely that more flows are contending for the link capacity. As *D-SRTF* gives more priorities to short flows than PIAS, the performance in long flows is slightly degraded. However, in data mining workload, *D-SRTF* performs better. Since data mining workload is more skewed, so there is less contending among flows, thus *D-SRTF* performs better than PIAS.

5.3 Performance in Mixed Workloads

It is common for one data center to host many different applications. To mimic the mixed workload scenario, we run four workloads (web search [4], data mining [15], cache [16] and Hadoop [16]) together, and the results are shown in Fig. 8. For PIAS, we run two experiments, one with the demotion thresholds for web search workload (PIAS-WS), and the other with the demotion thresholds for data mining workload (PIAS-DM). All values are normalized to the results achieved by PIAS-WS.

As we can see, in mixed workloads, *D-SRTF* also outperforms other schemes. For example, *D-SRTF* has 5 percent lower average FCT than PIAS-WS across all loads, and has 1∼7 percent lower FCT than PIAS-DM from low load to high load. The improvement for short flows is more significant. *D-SRTF* has up to 10 percent lower FCT than PIAS-WS and PIAS-DM in short flows, and more than 40 percent improvement towards L2DCT and DCTCP. *D-SRTF* also reduces the tail latency (99th percentile FCTs of short flows) by 15∼20 percent compared to PIAS-WS and PIAS-DM, and more than 60 percent compared to L2DCT and DCTCP. Meanwhile, *D-SRTF* also reserves its superiority in medium flows and long flows. We also present the CDF of the FCT in Fig. 9. As we can see, *D-SRTF* improves the performance of short flows while maintains the performance of long flows.

Moreover, we also break down the comparison into individual workload, to examine *D-SRTF*'s performance for each workload in the mixed workload scenario, and the results are shown in Fig. 10. As we can see, *D-SRTF* still reserves its superiority for each workload. Thus, *D-SRTF* is robust to different workloads.

5.4 Impact of Network Topology

We also repeat the experiment in Section 5.3 on another popular topology, Fat-Tree topology shown in Fig. 11, to examine the impact of network topology on the performance of *D-SRTF*, and the results are shown in Fig. 12. As we can see, similar to the results of experiments running on Leaf-Spine topology, *D-SRTF* performs better than PIAS and other schemes, in all cases. Therefore, *D-SRTF* is robust to different network topologies.

5.5 Comparison with Ideal Information Aware Scheme

We also compare *D-SRTF* with an ideal information aware scheme, pFabric [3]. pFabric assumes that the remaining

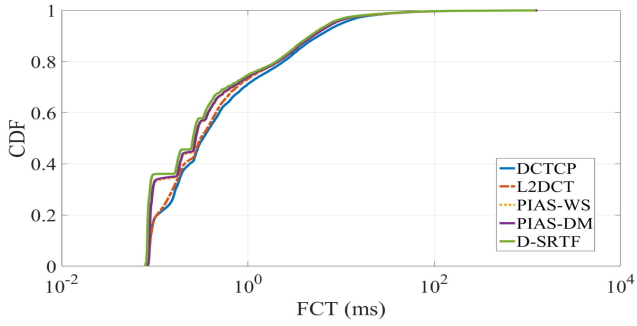
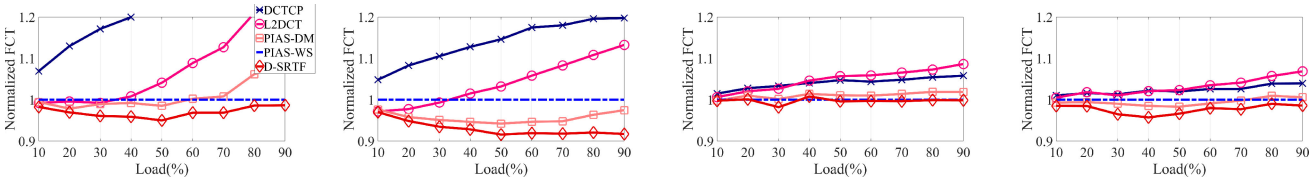


Fig. 9. Comparison of CDF of FCT in mixed workloads scenario.



(a) Cache (b) Web search (c) Hadoop (d) Data mining

Fig. 10. Comparison of FCT for individual workload in mixed workloads scenario. DCTCP's performance is outside the plotted range of (a).

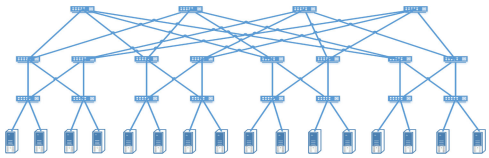


Fig. 11. Fat-Tree topology.

flow size is known and piggybacks this information in packet header, so that switches can implement shortest remaining flow size first scheduling. Since pFabric has significantly superior performance in short flows, we repeat the experiments in Section 5.3 with mixed workloads and compare *D-SRTF* with PIAS-DM, PIAS-WS and pFabric, in short flows, and the results are shown in Fig. 13.

In general, *D-SRTF* performs better than PIAS-DM and PIAS-WS, and is comparable to pFabric. Particularly in data mining workload, *D-SRTF* has only 1.6 percent performance gap to pFabric. This is expected, since data mining workload is more skewed than other workloads, so short flows can finish transmission within the first few windows, leading to less bandwidth contention with other flows, therefore near to optimal performance. Since web search workload is less skewed, so it is more likely that short flows coexist with long flows in the same priority queue. Therefore, short flows have to share link capacity with long flows, so the performance improvement is less significant. pFabric, on the other size, is information aware, i.e., explicitly knows the size of all flows, thus is not affected by such problem.

5.6 *D-SRTF* Deep Dive

Finally, we implement several sets of targeted experiments, to evaluate the properties of *D-SRTF*.

5.6.1 Impact of Number of Queues in Switch Port Buffers

As we leverage the multiple queues in switch port buffers to schedule packets, the number of priorities is restricted by the number of queues. Even though current commodity switches support up to 8 queues per port, in some cases,

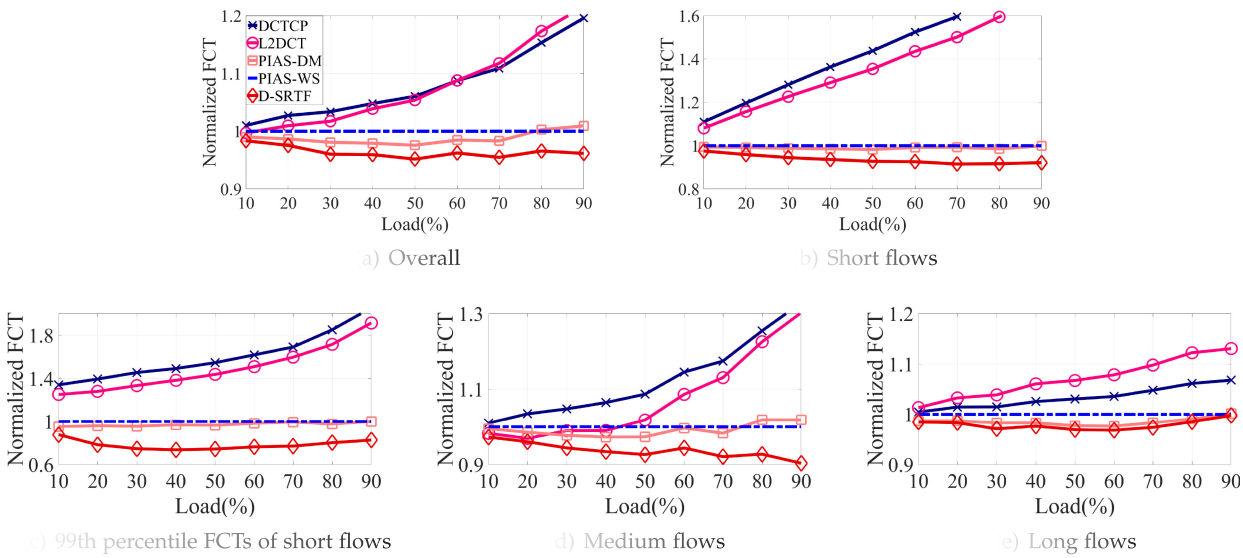
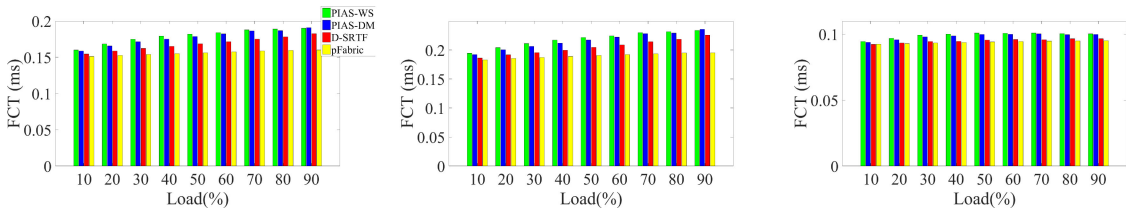


Fig. 12. Comparison of FCT in mixed workloads in Fat-Tree topology.



(a) Overall (b) Web search (c) Data mining

Fig. 13. Comparison with ideal information aware scheme (pFabric) in terms of FCT of short flows.

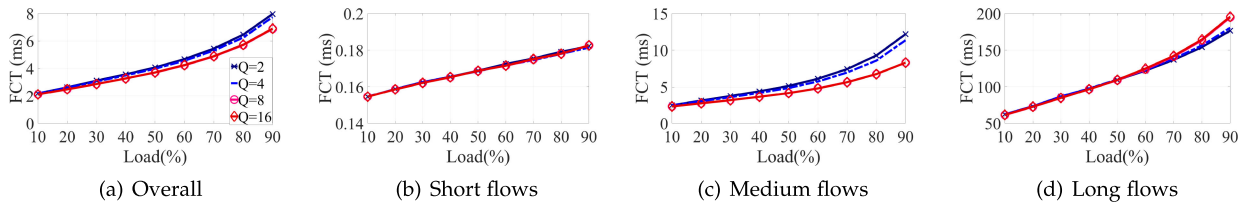


Fig. 14. *D-SRTF*'s sensitivity to the number of queues in switch port.

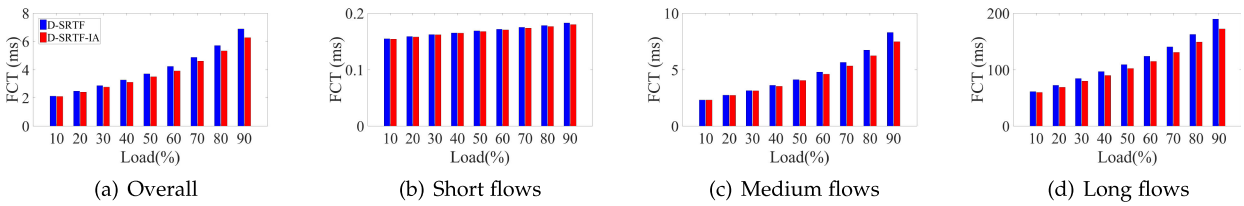


Fig. 15. Effectiveness of remaining flow size estimation.

network operators may reserve some queues for other purposes. For example, in SDN area, some strict high priority queues are utilized to deliver the control traffic [8]. In this case, the number of queues available for flow scheduling is limited. To demonstrate *D-SRTF*'s robustness, we repeat the experiments in Section 5.3 with different numbers of queues, and the results are shown in Fig. 14.

In general, the performances with different numbers of queues are similar, and the more queues we use, the better results we can obtain, as shown in Fig. 14a. This is expected, since the more queues we use, the more priorities we have, therefore the better flows are segregated, thus the better scheduling and overall performance are achieved. Besides, the performance in short flows are almost the same. This is because short flows are likely to be finished in the highest priority queue. The major difference lies in medium flows and long flows. When we use less queues, e.g., 2 queues, long flows are more likely to share the link bandwidth with short flows and medium flows, thus the performance of long flows can be improved, as shown in Fig. 14d.

It is also expected that in the future, switch port buffers can support more queues. To evaluate *D-SRTF*'s performance in more queues, we add another set of experiments with 16 queues in each switch port. Thus, we have 16 priorities for packets for flow scheduling, and the results are also shown in Fig. 14. As we can see, *D-SRTF*'s performance with 16 queues is almost the same as that with 8 queues, which indicates that *D-SRTF* is robust to different numbers of queues.

5.6.2 Effectiveness of Remaining Flow Size Estimation

With the heavy-tailed distribution, *D-SRTF* estimates the remaining flow size via the bytes sent. Here, we evaluate the effectiveness of this remaining flow size estimation, by adding a new scheme, *D-SRTF-IA*, which is a revised version of *D-SRTF* with **I**nformation **A**ware. That is, in *D-SRTF-IA*, we assume that the flow size is known at the flow start, and TCP senders can record the bytes sent, thus the remaining flow size is the flow size minus the bytes sent. All other mechanisms in *D-SRTF-IA* are the same with those in *D-SRTF*. We compare the performance of these two schemes, and the results are shown in Fig. 15. As we can see, with remaining flow size known, *D-SRTF-IA* performs better than *D-SRTF*, but the gap is within 8 percent, which indicates that our estimation is effective. Particularly, we find that *D-SRTF* performs almost the same with *D-SRTF-IA* in short flows. This is because the length of short flows is small, thus the gap between the estimated remaining flow size (using bytes sent) and the real one is small, therefore our estimation functions well for short flows. Considering that short flows dominate the DCN traffic, our estimation is useful in practice.

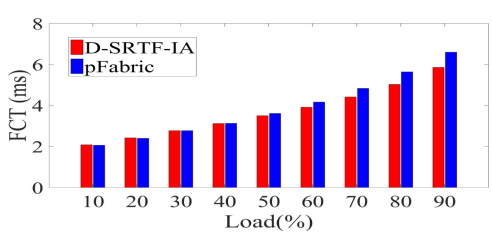


Fig. 16. Effectiveness of bandwidth estimation (Comparison of overall FCT).

5.6.3 Effectiveness of Bandwidth Estimation

To evaluate the effectiveness of bandwidth estimation, we compare *D-SRTF-IA* with pFabric. Note that, both *D-SRTF-IA* and pFabric assume that the remaining flow size is known, and the difference is that pFabric only considers the remaining flow size for scheduling, while *D-SRTF-IA* considers both the remaining flow size and the available bandwidth for scheduling. We repeat the experiment with mixed workloads, and the comparison results are shown in Fig. 16. As we can see, *D-SRTF-IA* performs better than pFabric, which indicates that our light-weight estimation for bandwidth is effective.

6 Conclusion

Minimizing FCT is always one of the objectives to improve DCN performance, and many existing works approximate SRTF scheduling to minimize FCT. However, they fail to consider the available bandwidth for flow scheduling, leading to sub-optimal solution in the dynamic DCN. Moreover, without flow size information and with the difficulty to measure the available bandwidth, it is hard to estimate the remaining time for each flow, to perform SRTF scheduling. Moreover, centralized scheduler could incur large overhead

and delay the scheduling decision. To solve these problems, we propose *D-SRTF*, a *light-weight yet effective* DCN scheme to implement SRTF scheduling in a distributed manner. *D-SRTF* uses the bytes sent to estimate the remaining size of each flow, and leverages the congestion indicator to calculate the available bandwidth. Thus the remaining time of each flow is calculated as the estimated remaining size over the available bandwidth. *D-SRTF* realizes SRTF scheduling in a distributed manner, in that, *D-SRTF* determines the priority for the packets of each flow at senders, based on the remaining time and a set of thresholds distributed to all senders. Upon receiving packets, switches enqueue the packets to the queue corresponding to the priority of packets, and adopt strict priority scheduling among all queues in each switch port buffer. Experiments with various workloads demonstrate the superiority of *D-SRTF* over existing DCN schemes. For example, for web search workload, *D-SRTF* improves the FCT for short flows by 8∼12 percent and improves the 99th percentile by 12∼21 percent when compared to PIAS, and in data mining workload, *D-SRTF* has only 1.6 percent performance gap to pFabric, and if information on flow size is known, *D-SRTF* could perform better than pFabric.

Acknowledgments

We thank *Wei Bai* from *SING Group @ HKUST* for sharing the source code for experiments and comparison. We are also grateful to anonymous reviewers for their constructive comments to improve the presentation of the paper. This work is supported by the China National Basic Research Program (973 Program, No. 2015CB352400), NSFC under grant U1401258, Science and Technology Planning Project of Guangdong Province (2015B010129011), Shenzhen Discipline Construction Project for Urban Computing and Data Intelligence, and grant from City University of Hong Kong (Project No. 7004888).

References

[1] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2012, pp. 127–138.

[2] A. Munir, I. Qazi, Z. Uzmi, A. Mushtaq, S. Ismail, M. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2013, pp. 2157–2165.

[3] M. Alizadeh, S. Yang, and M. Sharif, "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 435–446.

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

[5] 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. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, pp. 1–14.

[6] 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. ACM Conf. SIGCOMM*, 2014, pp. 491–502.

[7] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 455–468.

[8] W. Bai, K. Chen, L. Chen, C. Kim, and H. Wu, "Enabling ECN over generic packet scheduling," in *Proc. 12th Int. Conf. Emerging Netw. EXperiments Technol.*, 2016, pp. 191–204.

[9] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *Proc. 13th USENIX Conf. Netw. Syst. Des. Implementation*, 2016, pp. 537–549.

[10] C. Gao and V. C. S. Lee, "DEME: Decouple packet marking from enqueuing for multiple services in data center networks," in *Proc. IEEE 24th Int. Conf. Netw. Protocols*, 2016, pp. 1–2.

[11] C. Gao, V. C. S. Lee, and K. Li, "DemePro: DEcouple packet Marking from Enqueuing for multiple services with PROactive congestion control," *IEEE Trans. Cloud Comput.*, vol. PP, no. 99, pp. 1–14, 2017.

[12] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "PIAS: Practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 1954–1967, Aug. 2017.

[13] G. Judd, "Attaining the promise and avoiding the pitfalls of TCP in the datacenter," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 145–157.

[14] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, et al., "Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network," *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 183–197.

[15] 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. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 51–62.

[16] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. Snoeren, "Inside the social network's (Datacenter) network," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 123–137.

[17] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2012, pp. 139–150.

[18] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 174–187.

[19] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized zero-queue datacenter network," in *Proc. ACM Conf. SIGCOMM*, 2015, pp. 307–318.

[20] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[21] R. Mittal, R. Agarwal, S. Ratnasamy, and S. Shenker, "Universal packet scheduling," in *Proc. 13th USENIX Conf. Netw. Syst. Des. Implementation*, 2016, pp. 501–521.

[22] T. Wang, H. Xu, and F. Liu, "Aemon: Information-agnostic mix-flow scheduling in data center networks," in *Proc. 1st Asia-Pacific Workshop Netw.*, 2017, pp. 106–112.

[23] F. Liu, J. Guo, X. Huang, and J. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.

[24] J. Guo, F. Liu, J. Lui, and H. Jin, "Fair network bandwidth allocation in IaaS datacenters via a cooperative game approach," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 873–886, Apr. 2016.

[25] H. Yuan, J. Bi, M. Zhou, and A. C. Ammari, "Time-aware multi-application task scheduling with guaranteed delay constraints in green data center," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 3, pp. 1138–1151, Jul. 2018.

[26] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 393–406, 2015.

[27] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 443–454, 2014.

[28] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. Lau, "Efficient online coflow routing and scheduling," in *Proc. 17th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2016, pp. 161–170.

[29] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3366–3380, Nov. 2016.

[30] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward automatically identifying and scheduling coflows in the dark," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 160–173.

[31] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max-min fairness," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[32] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
[33] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw.*, 2009, pp. 73–82.
[34] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "RFC 2616: Hypertext Transfer Protocol–HTTP/1.1," Jun. 1999. [Online]. Available: https://www.ietf.org/rfc/rfc2616.txt
[35] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2008, pp. 63–74.
[36] B. Kalyanasundaram and K. R. Pruhs, "Minimizing flow time nonclairvoyantly," *J. ACM*, vol. 50, no. 4, pp. 551–567, 2003.
[37] I. A. Rai, G. Urvoy-Keller, M. K. Vernon, and E. W. Biersack, "Performance analysis of LAS-based scheduling disciplines in a packet switched network," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 1, pp. 106–117, 2004.
[38] 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. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[39] J. Nair, A. Wierman, and B. Zwart, "The fundamentals of heavy-tails: Properties, emergence, and identification," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, pp. 387–388, 2013.
[40] K. Ramakrishnan, S. Floyd, and D. Black, "RFC 3168: The addition of explicit congestion notification (ECN) to IP," 2001. [Online]. Available: https://tools.ietf.org/html/rfc3168
[41] S. Schaible and J. Shi, "Fractional programming: The sum-of-ratios case," *Optimization Methods Softw.*, vol. 18, no. 2, pp. 219–229, 2003.
[42] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 267–280.
[43] The Network Simulator-NS-2, [Online]. Available: http://www.isi.edu/nsnam/ns/
[44] C. Hopps, "Analysis of an equal-cost multi-path algorithm," 2000. [Online]. Available: https://tools.ietf.org/html/rfc2992

Chengxi Gao received the BS and MS degrees from the Department of Computer Science, Northeastern University, China, and the PhD degree from the Department of Computer Science, City University of Hong Kong. He is an assistant professor with the Institute of Advanced Computing and Digital Engineering, Shenzhen Institutes of Advanced Technology (SIAT), Chinese Academy of Sciences (CAS). Before joining CAS, he was a research associate in City University of Hong Kong. His research interests include data center networking and cloud computing.

Victor C.S. Lee received the PhD degree in computer science from the City University of Hong Kong, in 1997. He is currently an assistant professor with the Department of Computer Science, City University of Hong Kong. His research interests include real-time databases, data management in mobile and wireless computing and performance evaluation. He is a member of the ACM, IEEE and IEEE Computer Society. He has been the chairman of the IEEE, Hong Kong Section, Computer Chapter in 2006-2007.

Keqin Li is a SUNY distinguished professor of computer science with the State University of New York. His current research interests include parallel computing and highperformance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published more than 590 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently serving or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is an IEEE fellow.