S3: Size-aware Sequential Scheduling to Meet Deadlines in Data Center Networks

Chang Ruan[1], Jianxin Wang[1], Jiawei Huang[1], Yi Pan[2], Naixue Xiong[3]

[1] School of Information Science and Engineering, Central South University, China
[2] Department of Computer Science, Georgia State University, USA
[3] School of Computer Science, Colorado Technical University, USA
{ruanchang, jxwang, jiaweihuang}@csu.edu.cn, yipan@gsu.edu, nxiong@coloradotech.edu

Abstract

In modern Data Center Network (DCN), there exist various on-line interactive application services, such as web search, social networking and online transaction processing. An urgent demand for these on-line interactive applications is low deadline miss ratio. Though the flows with small or tiny size are overwhelming for most data center applications, previous deadline-aware transport protocols have no consideration for protection of these flows, with the inevitable result that few large flows occupy too much network service time and many small flows could not meet their deadlines. Therefore, the deadline miss ratio may be improved further if the flow sizes are considered by transport protocols. In this paper, we first analyze the distribution of the flow size according to the real DCN traffic trace. Then, based on the analysis, we propose a preemptive distribution flow scheduling protocol, Size-aware Sequential Scheduling (S^3), which schedules flows such that they can send data at their maximal sending rates and finish as quickly as possible. More importantly, different from the previous transport protocol studies, S^3 sequentially adjusts the flow scheduling order with the consideration of both the flow size and the flow deadline, which can help more small flows meet their deadlines. The at-scale simulations and a real flow distribution based test results show that S^3 significantly reduces the deadline miss ratio and obtains lower flow completion time compared to the recent protocols PDQ and D^3. In addition, S^3 is resilient to packet loss, and schedules flows with only the same time complexity as PDQ.

Keywords: Transport control protocol, Data center network, Deadline, Flow completion time

1 Introduction

Nowadays, many data centers are constructed and deployed to provide various on-line interactive application services, such as web search, social networking and online transaction processing [1]. These application services have soft-real-time nature and needs to respond in a timely fashion. The high response latency of these services degrades the user experience and consequently reduces the revenue of operators of data centers. For example, Amazon found that every extra 100ms latency is equivalent to a 1% loss in business revenue [2]. Hence it is important to guarantee low latency for the on-line interactive application services.

Unfortunately, the underlying transport protocol deployed in many today's data centers is traditional Transport Control Protocol (TCP), which aims to maximize throughput over the unreliable networks like Internet, inclining to enlarge the latency by filling up the switch buffer. Consequently, TCP performs poorly in data center networks (DCNs). For instance, in the popular application services such as MapReduce [3], the many-to-one communication pattern causes TCP flows to suffer from timeout or even the Incast throughput collapse problem [4], which greatly degrades the latency performances of application services.

Recent research focuses on designing novel transport control protocols to address the problem of latency performance with TCP [5-14]. Especially, the proposed protocols operate under soft-real-time constraints (e.g., 300ms latency) which imply deadlines for network communi- cation within the on-line applications. D^3 [5] and D^2TCP [6] exploit the explicit deadline information to allocate bandwidth for each flow in their protocol designs. They all try to help more flows to meet their deadlines since the less flows miss their deadlines, the better the response results of online application services are. To help more flows meet their deadlines, some other deadline-aware protocols are proposed to mimic various scheduling principles [9-11]. For example, PDQ [10] uses Earliest Deadline First (EDF) discipline to meet the deadline requirements. The smaller deadline a flow has, the earlier it sends data. When flows have the same deadline, PDQ uses Shortest Job First (SJF) discipline,

*Corresponding Author: Jianxin Wang; E-mail: jxwang@csu.edu.cn
DOI: 10.3966/160792642018121907001

that is, a flow with the smallest size will send data firstly.

These approaches could alleviate the impact of latency problem greatly and still suffer from their respective drawbacks. In particular, when the aggregate rate demand of the deadline-aware flows exceeds the link capacity, PDQ may not help the maximum number of flows to meet their respective deadlines. Other deadline-aware transport control protocols still use Fair Sharing or First In First Out (FIFO) scheduling discipline, which could degrade the latency performance of large number of small flows. As a result, the number of flows meeting deadlines is reduced much.

In this paper, we propose a flow scheduling protocol called Size-aware Sequential Scheduling (S^3) to meet deadlines for as many flows as possible. S^3 adopts preemptive distributed flow scheduling to finish flows quickly. The key point is that by considering both flow sizes and flow deadlines, small flows can preempt large flows. Therefore, S^3 prevents large flows from occupying too much service time of small flows. The results of performance evaluation show that S^3 outperforms D^3, TCP and PDQ in terms of deadline miss ratio.

2 Related Work

Many transport protocols have been proposed to reduce the latency and meet deadlines for flows in DCNs. In this section, we boardly classify those protocols into three categories. They approximate different, scheduling disciplines, which are summerized in Table 1. In the following, the details of these protocols are illustrated.

Table 1. Transport protocols for reducing the latency in DCNs

Category	Examples	Scheduling discipline
Fair sharing protocols	DCTCP [12]	PS
Deadline-aware protocols	D^3 [5], D^2TCP [6], MCP [16]	FIFO
	L^2DCT [11]	LAS
	PDQ [10]	EDF
Size-aware protocols	pFabric [9]	SRPT
	PASE [18]	SJF

2.1 Fair Sharing Protocols

As a pioneer of enhanced TCP protocol in DCNs, Data Center TCP (DCTCP) [12] uses Explicit Congestion Notification (ECN) marks for measuring the extent of congestion, and adjusts its sending rate to alleviate congestion. With the accurate congestion information according to ECN marks, DCTCP can control the queue around a small value well and reduce the flow completion time. However, DCTCP is a deadline-unaware protocol and approximates to Fair Sharing or Processor Sharing (PS) [15] discipline, which is known to be far from maximizing the number of deadline-meeting flows. When flows have their respective deadline requirements, DCTCP has no corresponding mechanisms to change the sending rates of these flows according to their deadlines.

2.2 Deadline-aware Protocols

D^3 [5] is the first deadline- aware transport control protocol, which employs explicit rate control with the aid of switches. Although D^3 significantly enhances the deadline-meeting ratio in comparison with DCTCP, it does not employ any flow scheduling strategy but allocates rates to flows by their arriving orders. This means that the scheduling discipline of D^3 is FIFO. Consequently, the flows with lax deadlines coming earlier occupy the bottleneck bandwidth, while some urgent flows arriving later may miss their deadlines.

Based on DCTCP, Balajee Vamanan et al. propose D^2TCP [6], which elegantly regulates the congestion window size according to the extent of congestion and the flow deadlines. When the congestion occurs, the far-deadline flows release some bandwidth to the near-deadline ones. Unfortunately, D^2TCP only modifies the congestion control algorithm at the sender side without any enhancement of the flow scheduling scheme, thus faces the same problem of D^3 as illustrated above.

The recent proposed transport protocol L^2DCT [11] mimics the Least Attained Service (LAS) scheduling at senders in a distributed way. According to the bytes a flow has sent, L^2DCT distinguishes the large and small flows. When flows suffer the congestion, the rate of small flows are decreased less than large flows. In this way, more small flows can meet their deadlines. Nevertheless, the large and small flows still compete for the bottleneck link, meaning that the number of deadline-meeting flows with L^2DCT can be improved further.

MCP (Minimal-impact Congestion control Protocol) [16] derives optimal source rates by solving a stochastic packet delay minimization with constraints on completing within deadlines. Apart from finishing the deadline flows, MCP minimizes the flow completion time of non-deadline flows. Although the design of MCP has the solid theory basis, the scheduling discipline of MCP is same to D^3.

Unlike the previous protocols, PDQ [10] is a preemp- tive flow scheduling protocol which sends flows based on flow priorities. The flow priority is determined on switches according to the EDF discipline, that is, a flow with smaller deadline has higher priority. Unless flows have the same deadline, the flow with the smaller size has higher priority (SJF). Flows with high priorities send data with the link rate and can preempt flows with lower priorities. Besides,

when PDQ anticipates that a flow cannot finish its sending before its deadline, it terminates the flow to save the bottleneck link bandwidth for other flows. However, when the bottleneck link are overloaded, EDF may degrade the deadline miss ratio [17]. Furthermore, for the flows with different deadlines, it is very possible that large flows are scheduled earlier due to their smaller deadlines. In this case, even a few of large flows may block a great quantity of small flows, which can result in high deadline miss ratio.

2.3 Size-aware Protocols

pFabric [9] decouples flow scheduling from rate control. The switch determines the flow scheduling order according to the remaining flow size. The flow with the least remaining size is scheduled first (Shortest Remaining Processing Time, SRPT). Congestion control is simplified at sources which send flows at line rate at the beginning and throttle rate only under persistent packet loss. pFabric minimizes the average flow completion time, but its aggressive congestion control manner results in lots of packet loss.

PASE [18] synthesizes existing transport protocols strategies to reduce the average flow completion time and meet the flow deadlines. In its arbitration strategy, PASE determine the flow priorities according to the flow size (SJF), while our work explores the method to schedule flows using both the flow sizes and deadlines.

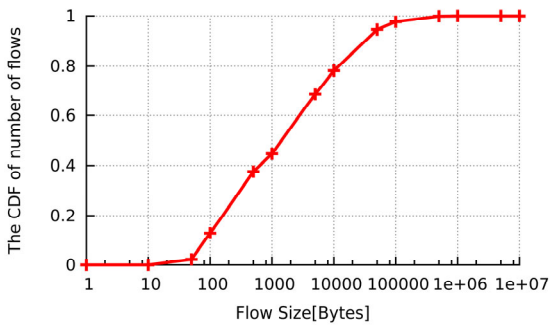
3 Background and Motivation

3.1 Flow Information

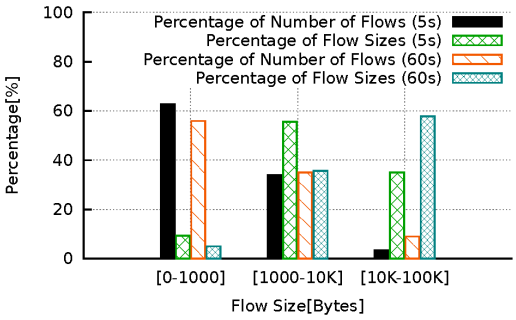
Flow deadline. In most enterprises, the latency requirements of online interactive application services have been specified by SLAs (Service Level Agreements) [19-20]. To prevent the SLA from being violated, working processes in these applications are assigned deadlines, usually on the order of 10-100ms. For these online interactive application services, the number of flows meeting their deadlines is a key performance metric since it relates with the response quality. For example, in the web search application using the partition-aggregate structure, more indexing results can improve the accuracy of query responses [6].

Flow size. For many on-line interactive application services, the flows size initiated by working processes can be known in advance. For example, in web search, the size of query flows is fixed (1.6KB), and the size of the corresponding response flows is 1.6-2KB [12]. In cloud storage system, each Server Request Unit (SRU) stripped on a server is constant, e.g., 64KB or 128KB [21]. The same holds for application services like key-value store and data processing [10].

Next, we give a comprehensive analysis of traffic trace from a real DCN to illustrate the characteristic of the flow size distribution. The data trace is collected from a real university campus data center [22]. The 2.1G trace records traffic data of more than 300,000 flows for at least 10 days. Figure 1(a) presents the cumulative distribution function (CDF) for different flow sizes. The probability of the flow size less than 10KB approximates 0.8, while the probability of that between 10KB and 100KB is about 0.2. In Figure 1(b), we list the percentage results in 5 seconds and 60 seconds drawn randomly from the trace. The percentages of the number of flows below 10K are about 96.7% in 5 seconds and about 90.9% in 60 seconds. As shown in Figure 1(b), the percentages of flow size is the ratio of the sum of the flow sizes in corresponding interval to the sum of all flow sizes, We can find that the prercentages of flow sizes below 10K are only 65% and 40.7% in 5 seconds and 60 seconds, respectively. It is clear that the number of small flows are in the majority, but their total sizes are small. It should be noted that the similar result is also given from the typical applications in DCN. For example, the Web Searching traffic involves at least 80% flows with the data size below 10KB, while this proportion is higher than 96% in the Data Mining traffic [23].



(a) The CDF of the number of flows



(b) The percentage of flow sizes and number of flows in 5s and 60s

Figure 1. Flow size and the number of flows distribution in a real data center

3.2 Size-aware Scheduling

Based on the flow size distribution, there comes a problem that, if a few large flows occupy too much network resources, large number of small flows may be

blocked and can not finish their sending before deadlines. Unfortunately, the problem is ignored by most of scheduling disciplines. Next, by a simple example, we compare three disciplines which are PS scheduling discipline approximated by TCP or DCTCP, FIFO used by D^3 and EDF adopted by PDQ. We use application throughput, the percentage of flows meeting their deadlines, as the performance metric.

Consider the scenario that there are three flows A, B and C through the same bottleneck link. For simplicity, we assume that one unit size needs one unit time to transmit. As shown in Table 2, the flow sizes of three flows are 3, 4, and 6, and the corresponding deadlines are 7, 8 and 6, respectively. When a flow can not finish sending before its deadline, it will be terminated to free its occupied network resources.

Table 2. Flow sizes and deadlines of A, B, C

Flow	Size	Deadline
A	3	7
B	4	8
C	6	6

As shown in Figure 2(a), if PS scheduling discipline is adopted, all three flows share the bottleneck link fairly. After 8 units time, flow C only sends 2 units, flow A sends 2.5 units and flow B sends 3.5 units. Obviously, all flows miss their deadlines. In Figure 2(b), the flows are scheduled in FIFO discipline. We assume that flows arrive in the order C→B→A. In this case, only C accomplishes its transmission before its deadline. Figure 2(c) demonstrates the situation of EDF discipline. According to EDF discipline, C sends earlier than A or B since its deadline is smaller than A or B. Obviously, both A and B miss their deadline requirements. For PS, FIFO and EDF, the application throughputs are 0, 1/3 and 1/3, respectively. The reason of low efficiency is that the large flow C sends firstly and occupies too much service time. Consequently, the small flow A and B are not able to meet their deadlines. However, as shown in Figure 2(d), if A preempts C, while B is scheduled at time 3, only C will miss its deadline. In this case, the application throughput, i.e., the ratio of the number of flows meeting their deadlines to the total number of flows, is increased to 2/3. Based on the observation of the above scheduling example, it is obvious that the application throughput could be improved if the large flow is prevented from occupying too much service time.

Our observation of the empirical data and the example leads us to conclude that (i) the number of small flows is much larger than that of larger flows in most of applications, and (ii) the number of flows meeting deadlines can be improved if we schedule the small flows first rather than larger flows. These conclusions motivate us to investigate a novel scheduling algorithm using both the flow size and flow deadline information.

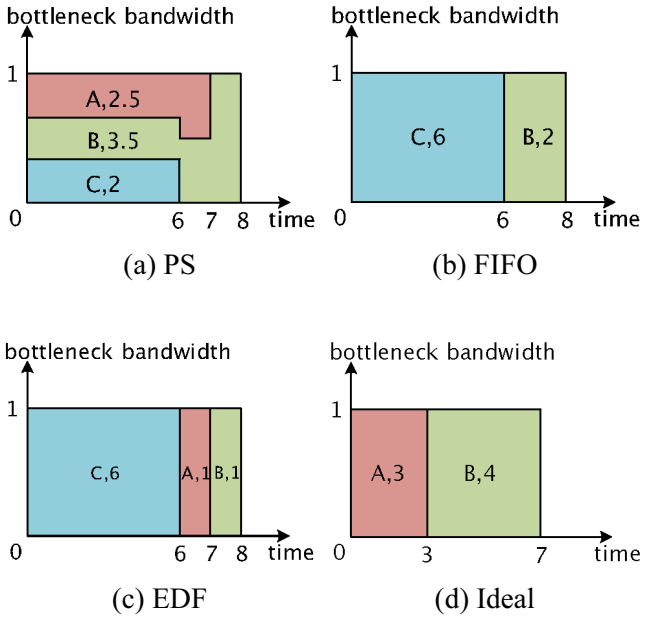


Figure 2. Flow scheduling example

4 Protocol Design

4.1 S^3 Overview

S^3 is a flow scheduling protocol, which aims to maximize the number of flows meeting their deadlines. S^3 works in the distribute manner that can achieve scalability and avoid the single point failure occurring in the centralized manner.

Figure 3 provides the overview of S^3. First, the application exposes the flow size and flow deadline to S3 transport layer. Then the sender adds a scheduling header carrying the flow information to the SYN packet. According to the flow information in the SYN packet, the switch determines the sending order of each flow. The flow will be paused by switch if it is not scheduled to send data. After receiving the SYN packet, the receiver copies all information in the scheduling header into the ACK packet. Finally, when the ACK packet arrives, the feedback information is used to determine whether the sender is paused. If it is not paused, the sender will send data packets with the link rate. Otherwise, it will send probe packets until it is not paused by switches or misses its deadline.

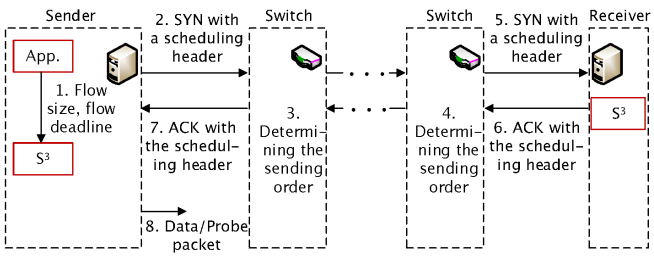


Figure 3. Overview of S^3

4.2 S³ Sender and Receiver

Similar to PDQ, S³ is deployed on the switch, sender and receiver sides. The details are as follows.

Sender. For S³ sender, it sends a SYN packet for flow initialization. Whenever a packet departs, the sender attaches a scheduling header to the packet, containing its current sending rate R_i, the ID of the switch (if any) S_i, the flow deadline D_i, the inter-probing time P_i, and the measured Round Trip Time (RTT) RTT_i. The current sending rate R_i is initialized to the link rate, and S_i, P_i and RTT_i are zero. When an ACK packet arrives, the sender updates these variables based on the feedback. If S_i is not zero, it means that the sender is paused by a switch in the network. Then, the sender will send a probe packet every P_i to obtain the sending information from switches.

For flows which can not finish before their deadlines even if they are not paused by switches, S³ sender stops the flows to prevent them from consuming any network resource. Sometimes, if a packet suffers from a timeout, S³ will resend the packet immediately. In our implementation, the timeout time interval is twice RTTs.

Receiver. The operation of S³ receiver is simple. The receiver only copies the scheduling header from each data packet to its corresponding ACK.

4.3 S³ Switch

In the S³ switch, flows are assigned a rate according to our scheduling algorithm running at the switch. If a flow can be scheduled, it obtains the rate contained in the scheduling header. On the contrary, if a flow can not be scheduled now, the switch assigns zero rate to the flow and records the switch ID into S_i in the scheduling header of the packet (i.e., pauses the flow).

Next, we illustrate the scheduling algorithm, which is the key operation of S³ switch. The basic idea of S³ is that it adopts both the flow sizes and flow deadlines to determine the sending order of flows. The scheduling algorithm includes two operations, called FILTER and SLACK. In FILTER operation, flows are sorted by deadline with an enhanced EDF discipline. Then, S³ adjusts scheduling orders to send small flows as many as possible to achieve lower deadline-miss ratio in SLACK operation.

FILTER operation. It is known that, EDF sorts the flow scheduling order only by the flow deadline. That is, the flow with the earliest deadline will have the highest priority. Even if the flow can not finish its transmission before deadline, it still transmits by the deadline order, which is obviously useless and may harm flows with larger deadlines. Therefore, FILTER operation checks if a flow is schedulable with the consideration of flow execution time, which is determined by flow size.

For n flows, we sort them by their ascending deadline order to get the ordered set $F=\{f_1, f_2 \ldots, f_n\}$. The corresponding deadlines are denoted as $D=\{d_1, d_2, \ldots, d_n\}$ and the sizes are $S=\{s_1, s_2, \ldots, s_n\}$. We assume one unit size needs one unit time to finish transmission. Then, if flow f_i is schedulable, it should satisfy

$$d_i \geq s_i + \sum_{j=1}^{i-1} s_j , \quad (1)$$

where s_j is the size of a schedulable flow, and d_i is the deadline of flow f_i. The formula shows that flow f_i is schedulable if its deadline is not less than its size plus the sum of flow size of all schedulable flows whose positions (i.e., scheduling order) are before flow f_i. The detail of the FILTER operation is presented in Algorithm 1.

Algorithm 1. FILTER operation

Initial state:

At the beginning, there are n flows that need to be determined on the scheduling order at a switch. *sum* refers to the sum of all scheduleable flow sizes $f_i.deadline$ refers to the deadline of flow i, $f_i.size$ refers to the size of flow i, and $f_i.mark$ refers to a mark bit of flow i, taking 1 for the schedulable flow, or else 0;

1: Sort the flows by deadlines;
2: $sum = 0$;
3: for $i = 0 \ldots n$ **do**
4: **if** $f_i.deadline > sum + f_i.size$ **then**
5: $f_i.mark = 1$;
6: $sum = sum + f_i.size$;
7: **else**
8: $f_i.mark = 0$;
9: **end if**
10: end for

S³ firstly sorts the flow scheduling order with EDF discipline in Step 1. Then, S³ checks whether the scheduling result of EDF is schedulable or not in Step 4. If a flow is schedulable according to Eq. (1), the flow is marked with 1, otherwise it is marked with 0.

SLACK operation. After the FILTER operation, small flows may be blocked and can not finish their sending before deadlines. If we substitute some small unschedul- able flows for the large schedulable flow, the total number of schedulable flows may be increased. In our algorithm, the flow substitution is called SLACK.

The SLACK operation is given in Algorithm 2. After the FILTER operation, all flows have been sorted by the deadline. The schedulable and unschedulable flows are marked 1 and 0, respectively. If flow f_i is marked 1, Algorithm 2 inserts flow f_i into a binary search tree structure T in Step 5. The structure is used to find the largest removable flow from F_f with $O(logn)$ time complexity. If flow f_i is marked 0, Algorithm 2 firstly checks whether fi can be added into F_f. If f_i satisfies the Eq. (1) in Step 8, it is marked 1. Or else, in

Step 12, Algorithm 2 finds out the the largest flow f_{max} from T. If the size of fi is not larger than that of f_{max} and f_i's deadline is greater than the total size of schedulable flows before it, Algorithm 2 substitutes f_i for f_{max} and updates T and sum.

Algorithm 2. SLACK operation

Initial state:

There are n flows after performing the FILTER operation. sum refers total execution time, f_{max} refers to the flow with the largest size, and T refers to a binary search tree;

1: Sort the flows by deadlines;
2: $sum = 0$;
3: for $i = 0...n$ **do**
4: **if** $f_i.mark == 1$ **then**
5: insert fi into the binary search tree T;
6: $sum = sum + f_i.size$;
7: **else**
8: **if** $f_i.deadline \geq sum + f_i.size$ **then**
9: insert fi into a tree T and set $f_i.mark$ to be 1;
10: $sum = sum + f_i.size$;
11: ***else***
12: find flow f_{max} with the largest size from T;
13: **if** $f_i.size \leq f_{max}.size$ and $f_i.deadline \geq sum - f_{max}.size + f_i.size$ **then**
14: update T;
15: update $f_i.mark$ and $f_{max}.mark$;
16: $sum = sum - f_{max}.size + f_i.size$;
17: **end if**
18: **end if**
19: **end if**
20: end for

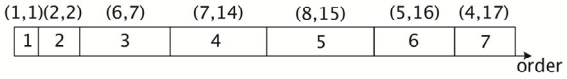
For a flow that cannot be scheduled first, its inter-probing time P_i is calculated by:

$$P_i = 0.2 \cdot RTT_{avg} \cdot I_i, \quad \textbf{(2)}$$

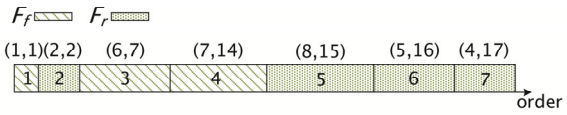
where RTT_{avg} is the average RTT of all flows passing through the current switch and I_i is the index of flow i in the set of schedulable flows determined by the scheduling algorithm.

4.4 Analysis of S^3 Scheduling Discipline

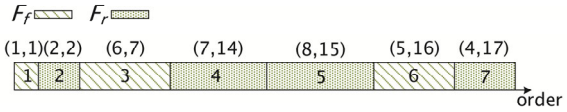
We give an illustrative example to validate the algorithm of S^3. In Figure 4, each flow is denoted as a rectangle. At the top of each rectangle, the notation (s, d) represents that the flow has size s and deadline d. For example, the size and the deadline of f_3 are 6 and 7, respectively. Figure 4(a) shows the result of EDF, in which all flows are sorted in ascending order of deadline. The result set after FILTER operation is shown in Figure 4(b). Here, we use F_f for denoting the schedulable flow set, and F_r for the unschedulable flow set. In Figure 4(b), F_f={ f_1, f_3, f_4 }, F_r={ f_2, f_5, f_6, f_7 }, and F=$F_f \cup F_r$. Figure 4(c) shows that f_6 substitutes the largest one in the three schedulable flows f_1, f_3 and f_4. Figure 4(d) shows the final result after the SLACK operation. Clearly, the total number of schedulable flows increases from 3 to 4.



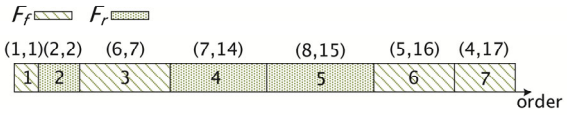
(a) The flow set sorted by EDF



(b) The flow set after FILTER operation



(c) The flow set after f_4 is substituted for f_6 in SLACK operation



(d) The flow set after f_7 is added into F_f in SLACK operation

Figure 4. An illustrative scheduling example

The illustrative example shows that our alogrithm outperforms EDF when there are no other flows arriving. However, in a dynamic setting, the problem that minimiz- ing the number of flows missing their deadlines is an NP-complete problem [10]. The idea of the scheduling alogithm S^3 is designed based on the principle of the least slack time (i.e., the difference of the flow deadline and the flow size), which has been proven that it has better performance than EDF when flows can not be scheduled to satisfy their deadlines by any algorithm [17]. Our contribution here is that we apply the scheduling principle to a preemptive flow scheduling protocol S^3 for meeting flow deadlines in DCNs.

One concern on the preemptive flow scheduling protocol S^3 is that the time complexity of the scheduling algorithm. In DCN, a switch may handle several thousand active flows in a one second [22]. This makes it the critical requirement for low time complexity in a scheduling algorithm. For our algorithm implemen- tation, we use quick sort in FILTER operation with $O(nlogn)$ time complexity. In the SLACK operation, since the binary search tree

structure in *T* is used to find the largest flow, it will only take *O*(*nlogn*) to insert or delete a flow. Therefore, the total time complexity of S³ algorithm is *O*(*nlogn*), which is as same as that of PDQ.

5 Performance Evaluation

5.1 Evaluation Setting

In the performance evaluations, we compare the following three schemes with S³ in DCN. The reason that we choose these schemes is because they are typical protocols using different scheduling discipline for meeting flow deadlines. For example, PDQ schedules flows mainly using EDF, D³ adopting FIFO and TCP approximating PS. While S³ schedules flows by considering both deadlines and sizes. The parameter settings of the schemes are described as follows.

PDQ [10]: As a typical flow scheduling scheme in DCN, PDQ mainly uses EDF as the scheduling algorithms. The Early Start, Early Termination and Suppressed Probing described in [10] are also used to enhance its performance in the evaluations. The threshold *K* in Early Start is set to 1.5 to get better performance.

D³ [5]: We implement D³ with the rate allocation and flow quenching algorithm. In the rate allocation, the parameters $\alpha = 0.1$ and $\beta = 1$. Since D³ allocates the flow rate depending on the flow arriving order, we assume that the order is not previously determined. The lucky flow will obtain its desired rate.

TCP: We use the TCP NewReno as the default transport protocol in DCN. In order to alleviate the Incast problem, we set a small value of 10ms to RTO_{min} as suggested in [24].

For the deadline-constrained flows, as the same as [10], the default flow size is set uniformly from the interval [2KByte, 198KByte], and the default flow deadline is drawn from an exponential distribution with mean 20ms. We set a low bound of deadline to 3ms, therefore each flow can finish before its deadline if there does not exist any other flows.

5.2 Query Aggregation

The Query Aggregation is the most common communication pattern in data center networks. Requests from application layers are partitioned into pieces and forwarded to the workers in low layers. Then the results returned by the workers are aggregated to produce a final result. The Query Aggregation appears in popular applications such as MapReduce [4]. In this subsection, we evaluate the application throughput with different number of flows, flow sizes and flow deadlines.

In Section 5.2 and Section 5.3, we use the single-rooted tree topology as shown in Figure 5. Each switch has 1Gbps uplink and a buffer of 100Kbytes for each port. The end-to-end round trip latency is 160 μs.

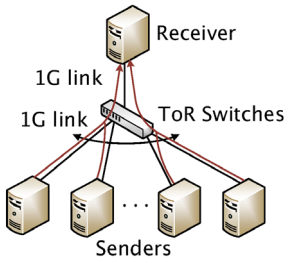


Figure 5. Many-to-one single-rooted tree

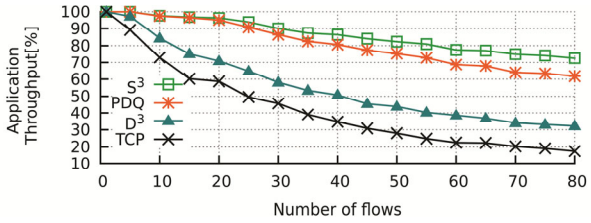
5.2.1 Comparison of Application Throughput

Impact of number of flows. Figure 6(a) shows that, with the increasing of the number of concurrently sending flows, the application throughput of D³ and TCP decrease a lot while that of PDQ and S³ degrade gracefully. When the number of flows becomes larger, S³ has more opportunities to execute the SLACK operation. Therefore, S³ obtains higher performance improvement with the number of flows increasing. The application throughput of S³ is about 10% higher than that of PDQ when the number of flows is larger than 55.

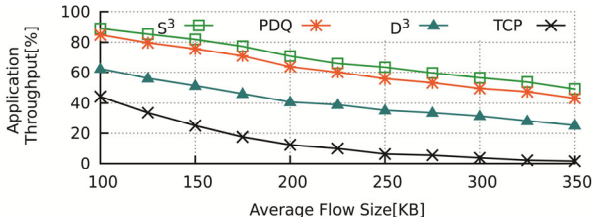
Impact of flow size. We study the application throughput of 30 concurrent flows in this test. Figure 6(b) demonstrates that, when the flow size becomes larger, the application throughputs of all protocols are degraded. As a deadline-agnostic protocol, TCP gets the worst performance. PDQ uses EDF to schedule flows and thus gets high application throughput. By preventing larger flows occupy too much service time of smaller flows, S³ still achieves better performance than that of PDQ with different flow sizes.

Impact of flow deadline. The maximum percent of flows that can meet their deadlines is the focus of the operations of data center. In this comparison, we change the mean flow deadline to test the maximal number of flows that can ensure 90% application throughput. Intuitively, the larger mean deadline, the more flows can be guaranteed to ensure the 90% application throughput.

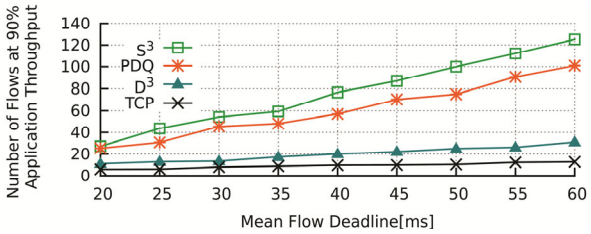
Figure 6(c) shows that, for all protocols, the number of flows at 90% application throughput becomes larger when the deadline is increasing. D³ and TCP can support only less than 40 concurrent flows to ensure 90% application throughput with the mean deadlines from 20ms to 60ms. The performances of S³ and PDQ are much better. Moreover, we find that S³ obtains about 10% improvement over PDQ, more than 3 times improvement over D³ when the mean deadline is 60ms.



(a) Varying the number of flow



(b) Varying the mean flow size



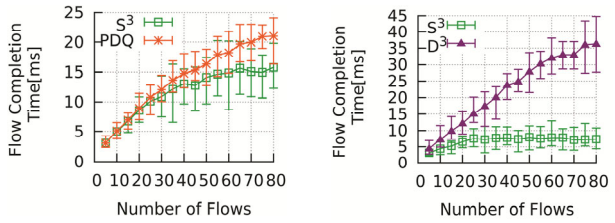
(c) Varying the mean flow deadline

Figure 6. Application throughput in query aggregation

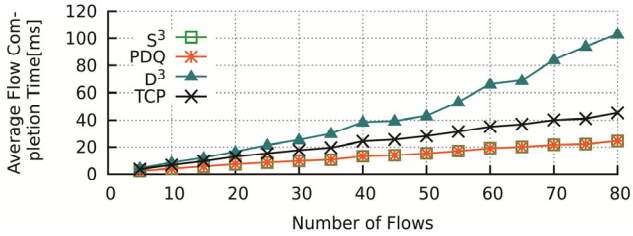
5.2.2 Comparison of Flow Completion Time

For deadline-constrained flows, we compare the flow completion time (FCT) for varying number of flows in Figure 7(a) and Figure 7(b). Since S^3 has higher application throughput than PDQ or D^3 as shown in Figure 6(a), to make accurate and fair comparison, we measure the FCT under the same application throughput. For example, if PDQ or D^3 only finishes 30 flows, we also choose the first 30 flows finished by S^3 to calculate the FCT. The maximum, average and minimum of FCT are drawn in Figure 7(a) and Figure 7(b). We find that S^3 outperforms both PDQ and D^3 with the increasing of the number of flows. When the number of flows is 80, the average FCT of S^3 is about 20% lower than PDQ, and 7 times lower than D^3.

In DCN, there also exits flows having no deadline, such as flows for updating data across servers, but it is desirable that they can finish earlier. For these deadline-unconstrained flows, we use the average flow completion time (AFCT) to compare the performance in different protocols. We measure the completion time of 100 flows in query scenario. In Figure 7(c), both S^3 and PDQ have the lowest AFCT for the reason that both scheduling algorithms degenerate into the SJF discipline in the absence of flow deadlines.



(a) FCT of S^3 and PDQ (b) FCT of S^3 and D^3



(c) Deadline-unconstrained flows

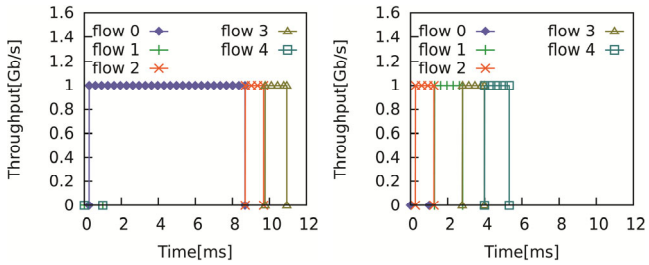
Figure 7. FCT in query aggregation

5.3 Impact of Dynamic Flows

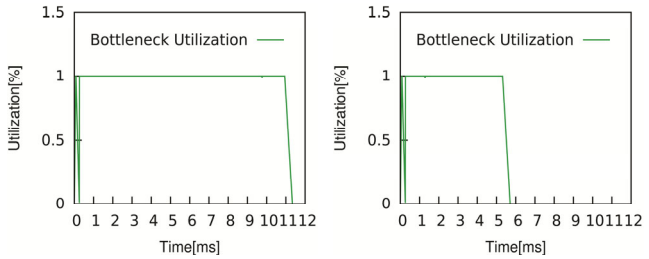
Compared with EDF discipline in PDQ, S^3 interchanges the scheduling order of large and small flows to add new flows into the scheduling order. In this test, we compare the dynamic scheduling of S^3 and PDQ. We start five flows at time 0. One flow, which is the largest one, has the size of 1MB and the deadline of 10ms. The sizes of other four small flows are 120KB, 140KB, 160KB and 180KB respectively, and their deadlines are randomly selected in the range of [10ms, 12ms].

Figure 8 shows the test results of PDQ and S^3 with dynamic flows. In Figure 8(a), since PDQ schedules flows according to the deadline, the first flow grabs the whole link bandwidth until it finishes sending all data packets. Then, the other four flows begin their transmission based on their deadlines. At last, only 3 flows are completed before their deadlines. While in Figure 8(b) with S^3 algorithm, flow 1~4 are transmitted earlier due to their smaller size and larger deadline compared with flow 0. Finally, 4 small flows all meet their deadlines. Compared to PDQ, the application throughput is improved by 20%. In Figure 8(c) and Figure 8(d), both S^3 and PDQ can achieve 100% link utilization at flow scheduling time, which means that S^3 and PDQ can converge to the equilibrium rate quickly. The degradation of link utilization at the beginning is due to sending the SYN packet in the initial RTT. Figure 8(e) and Figure 8(f) show the instantaneous queue lengths of S^3 and PDQ, which are both well controlled. The reason is that, with the flow scheduling algorithm of S^3 and PDQ, there is usually only one flow running in the bottleneck link and thus the oversubscription problem is prevented. We observe that the queue length is not larger than 5, which is mainly caused by probe packets. Nonetheless, from

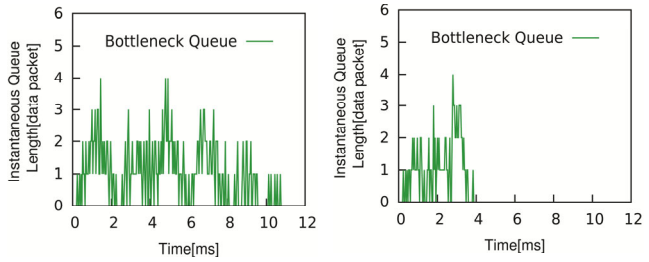
Figure 8, the FCT of S³ is much shorter than that of PDQ. For example, PDQ takes about 11ms for finishing three flows, while S³ takes roughly 5.7ms for finishing four flows.



(a) Flow throughput of PDQ (b) Flow throughput of S³



(c) Link utilization of PDQ (d) Link utilization of S³



(e) Queue length of PDQ (f) Queue length of S³

Figure 8. Dynamic flows of PDQ and S³

5.4 Impact of Traffic Workload

In this subsection, we study the impact of different traffic workloads, which are generated by corresponding sending patterns: (1) Aggregation: the many-to-one sending pattern which has been stated in Section 5.2. (2) Stride(z): a server with index γwill send flows to the server with index (γ+z) *mod* N, where N is the total number of servers and z is a fixed integer number. (3) Random Permutation: a server sends flows to another randomly selected server, and each server receives flows from only another server. (4) Prob(p): a server sends flows to another server in the same rack with probability p whereas to the other server in a different rack with probability 1-p. We use the two-level 12-server single-rooted tree topology as shown in Figure 9. The link rate, switch buffer and link delay are set as same as that in Section 5.3.

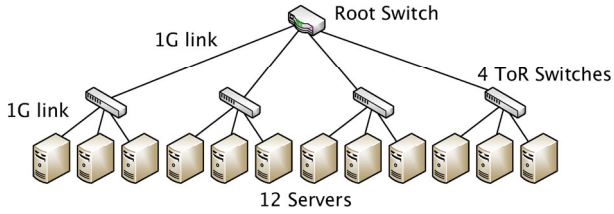
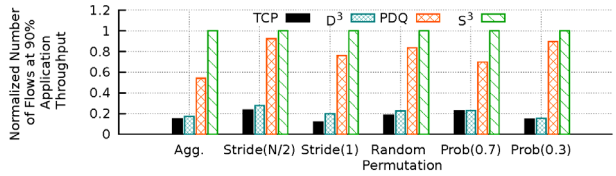
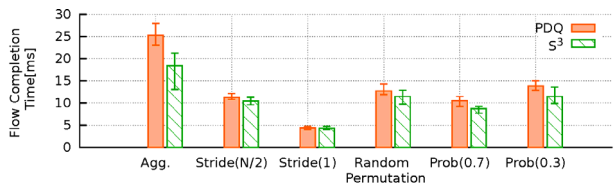


Figure 9. Two-level 12-server single-rooted tree

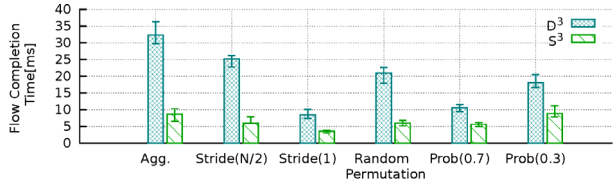
Figure 10(a) shows that S³ outperforms the other three protocols. We normalize the finished flow number based on that of S³. Thus, the value of S³ is always 1 for each sending pattern. Compared to PDQ, S³ gets the best performance improvement in the many-to-one pattern. In Prob(0.7) pattern, flows are confined in the same rack with a large probability. These local flows have smaller RTTs than those across different racks. As the local flows and non-local flows have the same distribution of the flow sizes, this means that the local flows have more lax deadlines. Similar to the result in Section 5.2, the Prob(0.7) pattern gets more performance benefit than Prob(0.3). Note that in the Stride(1) pattern, since each rack has 3 servers, there are only about 2/3≈0.67 of flows running in the same rack. Therefore, the performance of Stride(1) pattern is worse than that of Prob(0.7). In Figure 10(b) and Figure 10(c), we compare the FCT in different patterns. The FCT is calculated under the same application throughput with the same comparison method in Section 5.3. The best performance improvement is still in the many-to-one pattern, and S³ has lower FCT than other protocols.



(a) 90% Application throughput



(b) FCT of S³ and PDQ



(c) FCT of S³ and D³

Figure 10. Performance comparison under aggregation, stride, random permutation and probability patterns

5.5 Impact of Network Scale

In the architecture design of large scale DCN, recent trends favor the network fabric based on multi-rooted tree topologies, such as Fattree [25] and VL^2 [26]. To investigate the impact of the network scale, we conduct the simulation using the fat-tree topology. Here, we just present the fat-tree topology with 4 pods in Figure 11. In this test, we provide the test results when the number of pods ranges from 4 to 8, with the corresponding number of servers varying from 16 to 128. The other network parameters like link rate, switch buffer and link delay are as same as that in former settings.

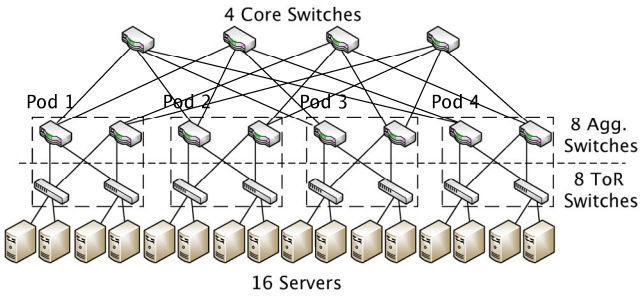
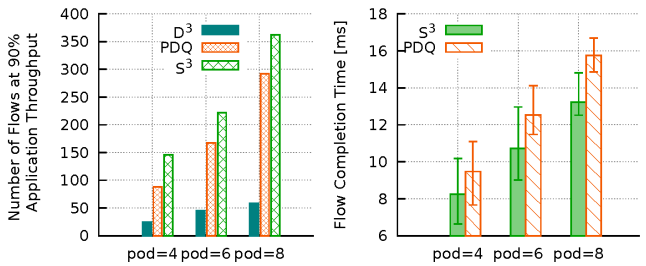


Figure 11. A simple fat-tree topology with pod=4

Figure 12(a) shows the number of flows at 90% application throughput of S^3, PDQ, D^3 and TCP. We find that, when the network scale becomes larger, there are more network resources like servers, switch buffers and links. In this case, more flows could meet their deadlines. Therefore, the test results of all protocols are getting better with larger *m*. Meanwhile, compared with D^3 and TCP, both S^3 and PDQ obtain higher improvements. Since S^3 avoids the deficit of PDQ, the performance of S^3 is higher than that of PDQ. Figure 12(b) also presents the FCT with the number of deadline-constrained flows varying from 100, 250 to 600 for different pods, respectively. The number of flows we test is about 5 times of the number of servers. As the FCT of TCP and D^3 are very large, we only present the results of S^3 and PDQ for clarity. Although the FCT increases with the increasing pods, S^3 still obtains the lower AFCT than PDQ consistently.



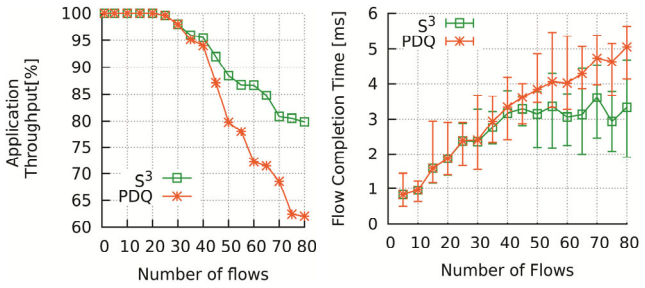
(a) Application throughput (b) FCT

Figure 12. Performance under different pods

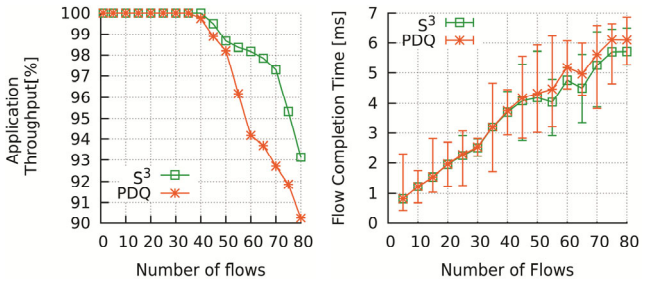
5.6 Real Flow Distribution Based Test

To test our protocol according to a real flow distribution, we generate flows from the previous measured results in Section 3.1. The probability of the flow size between 1KB and 10KB is 0.8, while that between 10KB and 100KB is 0.2. The distribution of the flow deadline is drawn from exponential distribution with mean 5ms or 10ms. Also we set a 2ms lower bound for all flows.

In Figure 13(a) and Figure 13(b), the mean of the flow deadline is 5ms. When the number of flows is less than 40, the performances of S^3 and PDQ are close. After 40, S^3 has higher application throughput than PDQ and lower flow completion time apparently. In Figure 13(c) and Figure 13(d), the mean of the flow deadline is 10ms. Compared with the results with 5ms flow deadline, both S^3 and PDQ have higher application throughput when the number of flows becomes larger. The reason is larger deadline means more flows can finish before their deadlines. Besides, the number of flows that S^3 outperforms PDQ a lot begins from 50 (the value is 40 with 5ms flow deadline), which is increased due to the larger deadline. However, the difference of FCT between S^3 and PDQ varies less than that with 5ms flow deadline since more flows can meet their deadlines with 10ms flow deadline.



(a) Application throughput in 5ms (b) FCT in 5ms



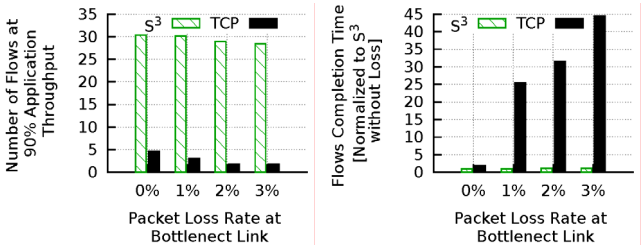
(c) Application throughput in 10ms (d) FCT in 10ms

Figure 13. Performance testing with a real distribution

5.7 Resilience to Packet Loss

Packet loss frequently happens in DCN due to

congestion caused by bursts of flows or flows sending data at line rate simultaneously. Figure 14(a) shows the 90% application throughput when the packet loss rate changes from 0 to 3%. Since S^3 sends probe packets continuously and always sends data at the maximal rate, the performance damage caused by packet loss can be amended. Thus, packet loss does not significantly affect the performance of S^3. When the packet loss rate becomes large, the AFCT of S^3 increases, as shown in Figure 14(b). TCP suffers from large AFCT since the timeout time is very large. In our evaluation, the performance of TCP is 40 times worse than S^3 at 3% packet loss rate.



(a) Application throughput (b) FCT

Figure 14. Performance under different packet loss rate

6 Conclusion and Future Work

For the latency intensive applications in DCN, reducing the number of deadline-missing flows can improve the service quality. We proposed a size-aware flow scheduling protocol S^3 that prevents large flows from occupying too much service time of small flows and sends flows as quickly as possible. The key idea of S^3 is that it schedules flows with the consideration of the flow size and the flow deadline. With only $O(nlogn)$ time complexity, S^3 uses the FILTER and SLACK operations to determine the flow scheduling order and obtains lower deadline-missing ratio. The extensive simulations show that, S^3 achieves higher application throughput and lower flow completion time than PDQ and D^3.

Several improvements will be investigated in the future. First of all, since S^3 modifies the switches, it is hard to test the performance of S^3 in a real large network system, while it is necessary to further valid its performance in a real small network system. The scheduling algorithm of S^3 is intentionally designed with more flow information such that it could outperfom EDF discipline. However, a detailed theoretical disscussion on the scheduling algorithm in the dynamic DCN environment will be more helpful for understanding the algorithm. Additionally, we will investigate how to implement the multipath variant of S^3 to raise the application throughput further since today's DCN has plenty of paths between servers.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant nos. 61502539, 61572530, and 61202494).

References

[1] H. Xu, Y. Sun, User Preference Mining and Privacy Policy Recommdation for Social Networks, *Journal of Internet Technology*, Vol. 16, No. 6, pp. 1145 -1155, Novemver, 2015.

[2] Latency Is Everywhere and It Costs You Sales-how to Crush It, http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it.

[3] J. Dean, S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, Vol. 51, No. 1, pp. 107-113, January, 2008.

[4] J. Zhang, F. Ren, C. Lin, Modeling and Understanding TCP Incast in Data Center Networks, *Proc. IEEE INFOCOM*, Shanghai, China, 2011, pp. 1377-1385.

[5] C. Wilson, H. Ballani, T. Karagiannis, A. Rowtron, Better Never than Late: Meeting Deadlines in Datacenter Networks, *Proc. ACM SIGCOMM*, Toronto, Canada, 2011, pp. 50-61.

[6] B. Vamanan, J. Hasan, T. N. Vijaykumar, Deadline-aware Datacenter TCP (D^2TCP), *Proc. ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Helsinki, Finland, 2012, pp. 115-126.

[7] X. Sun, J. Wang, N. Xiong, P. Dong, An Effective Experimental Platform for Multipath Transmission Protocol Algorithms and Performance Analysis, *Journal of Internet Technology*, Vol. 19, No. 5, pp. 1315 -1325, September, 2018.

[8] T. Zhang, J. Wang, J. Huang, Y. Huang, J. Chen, Y. Pan, Adaptive-Acceleration Data Center TCP, *IEEE Transactions on Computers*, Vol. 64, No. 6, pp. 1522-1533, June, 2015.

[9] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, pFabric: Minimal Near-optimal Datacenter Transport, *Proc. ACM SIGCOMM*, Hong Kong, China, 2013, pp. 435-446.

[10] C. Hong, M. Caesar, P. B. Godfrey, Finishing Flows Quickly with Preemptive Scheduling, *Proc. ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Helsinki, Finland, 2012, pp. 127-138.

[11] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, B. Khan, Minimizing Flow Completion Times in Data Centers, *Proc. IEEE INFOCOM*, Turin, Italy, 2013, pp. 2157-2165.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data Center TCP (DCTCP), *Proc. ACM SIGCOMM*, New Delhi, India, 2010, pp. 63-74.

[13] T. Zhang, J. Wang, J. Huang, Y. Huang, J. Chen, Y. Pan, Adaptive Marking Threshold Method for Delay-sensitive TCP in Data Center Network, *Journal of Network and Computer Applications*, Vol. 61, pp. 222-234, February, 2016.

[14] C. Ruan, J. Wang, J. Huang, W. Jiang, Analysis on Buffer Occupancy of Quantized Congestion Notification in Data Center Networks, *IEICE Transactions on Communications*, Vol. E99-B, No. 11, pp. 2361-2372, November, 2016.

[15] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown, Processor Sharing Flows in the Internet, *Proc. 13th International Conference on Quality of Service*, Passau, Germany, 2005, pp. 267-281.

[16] L. Chen, K. Chen, W. Bai, M. Alizadeh, Scheduling Mix-flows in Commodity Datacenters with Karuna, *Proc. ACM SIGCOMM*, Florianopolis, Brazil, 2016, pp. 174-187.

[17] C. D. Locke, *Best-effort Decision Making for Real-time Scheduling*, Ph. D. Thesis, Carnegie Mellon University, Pittsburgh, PA, 1986.

[18] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, F. R. Dogar, Friends, not Foes: Synthesizing Existing Transport Strategies for Data Center Networks, *Proc. ACM SIGCOMM*, Chicago, IL, 2014, pp. 491-502.

[19] Performance and Scalability, https://www.allthings distributed.com/2006/04/performance_and_scalability.html

[20] S. Yang, Using SLA Strategy to Design an SOC Platform in Data Center on the Cloud Computing, *Journal of Internet Technology*, Vol. 14, No. 5, pp. 751-758, September, 2013.

[21] H. Wu, Z. Feng, C. Guo, Y. Zhang, ICTCP: Incast Congestion Control for TCP in Data Center Networks, *IEEE/ACM Transactions on Networking*, Vol. 21, No. 2, pp. 345-358, April, 2013.

[22] T. Benson, A. Akella, D. A. Maltz, Network Traffic Characteristics of Data Centers in the Wild, *Proc. ACM SIGCOMM Conference on Internet Measurement (IMC)*, Melbourne, Australia, 2010, pp. 267-280.

[23] H. Xu, B. Li, Repflow: Minimizing Flow Completion Times with Replicated Flows in Data Centers, *Proc. IEEE INFOCOM-IEEE Conference on Computer Communications*, Toronto, Canada, 2014, pp. 1581-1589.

[24] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, B. Mueller, Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication, *Proc. ACM SIGCOMM 2009 Conference on Data Communication*, Barcelona, Spain, 2009, pp. 303-314.

[25] H. Hsiao, Y. Chao, C. Chu, Irregular Network Topologies for Data Centers, *Journal of Internet Technology*, Vol. 16, No. 3, pp. 517-523, May, 2015.

[26] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, S. Sengupta, VL^2: A scalable and Flexible Data Center Network, *Proc. ACM SIGCOMM 2009 Conference on Data Communication*, Barcelona, Spain, 2009, pp. 51-62.

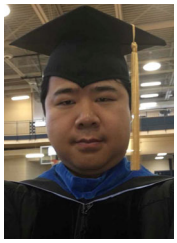
Biographies

Chang Ruan is currently a student pursuing his Ph.D. degree at School of Information Science and Engineering from Central South University in China, where he received his Master (2011) and Bachelor (2007) degrees. His current research interest includes data center networks.

Jianxin Wang received the Ph.D. degree in computer science from Central South University, China, in 2001. Currently, he is a professor at School of Information Science and Engineering, Central South University, China. His current research interests include algorithm optimization, computer network and bioinformatics.

Jiawei Huang obtained his Ph.D. degrees in Computer Science and Technology from Central South University, China. He now works in the School of Information Science and Engineering at Central South University. His research interests include wireless networks and data center networks.

Yi Pan received the BEng and MEng degrees in computer engineering from Tsinghua University, China, in 1982 and 1984, respectively, and the Ph.D. degree in computer science from the University of Pittsburgh, Pennsylvania, in 1991. His research interests include parallel and distributed computing, networks, and bioinformatics.



Naixue Xiong is current a faculty at the Department of Business and Computer Science, Southwestern Oklahoma State University (SWOSU), OK, USA. He received his both Ph.D. degrees in Wuhan University, and Japan Advanced Institute of Science and Technology, respectively. His research interests include Cloud Computing, Business Networks, and Optimization Theory.