

# More load, more differentiation – Let more flows finish before deadline in data center networks



Han Zhang<sup>a,c</sup>, Xingang Shi<sup>b,c,\*</sup>, Yingya Guo<sup>a,c</sup>, Zhiliang Wang<sup>b,c</sup>, Xia Yin<sup>a,b</sup>

<sup>a</sup> Department of Computer Science and Technology, Tsinghua University, Beijing, China

<sup>b</sup> Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China

<sup>c</sup> Tsinghua National Laboratory for Information Science and Technology (TNLIST), Beijing, China

## ARTICLE INFO

### Article history:

Received 24 February 2017

Revised 21 August 2017

Accepted 22 August 2017

Available online 30 August 2017

### Keywords:

Data center

Congestion

Rate control

Deadline

Flow completion time

## ABSTRACT

Data center network has become an important facility for hosting various online services and applications, and thus its performance and underlying technologies are attracting more and more interests. In order to achieve better network performance, recent studies have proposed to tailor data center network traffic management in different aspects, devising various routing and transport schemes. In particular, for applications that must serve users in a timely manner, strict deadlines for their internal traffic flows should be met, and are explicitly taken into consideration in some latest flow rate control or scheduling algorithms in data center networks. In this paper, we advocate that when designing such deadline-aware rate control schemes, a simple principle should be followed: flows with different deadlines should be differentiated in their bandwidth allocation/occupation, and the more traffic load, the more differentiation should be made. We derive sufficient and necessary conditions for a flow rate control scheme to follow this principle, and present a simple congestion control algorithm called Load Proportional Differentiation (LPD) as its application. We have evaluated LPD under different topologies and load scenarios, both by simulation and in real testbed. Compared with D<sup>2</sup>TCP, the state-of-art window-based deadline-aware congestion control schema, LPD often reduces the number of flows missing their deadlines by more than 25%. Compared with Karuna, the state-of-art deadline-aware rate control method, LPD only performs about 5% worse on average, but under heavy congestion, LPD performs about 5%–10% better than it. Indeed, the more load more differentiation is a general principle and it can also be used for the optimization of other object. Specifically, we consider minimizing the average flow completion time. Compared with the window-based protocol L<sup>2</sup>DCT, LPD can reduce flow completion time by 30% and compared with the state-of-art scheduling method pFabric, it only underperforms by 20% on average.

© 2017 Elsevier B.V. All rights reserved.

## 1. Introduction

Nowadays, more and more services and applications such as web search and social networking are hosted in data centers [1–5]. Since their performance, especially the network performance such as data transfer latency and throughput, directly affect the service quality experienced by end users, network protocols have to be specially designed to fully recognize the unique characteristics of data centers, and fully utilize the resources provided by the underlying infrastructure. In data center networks (DCNs), applications typically require low latency for short flows, and require high burst tolerance and high throughput for large flows [2,6]. To meet these requirements, recent studies have proposed to tailor

traffic management in different aspects, and devised various transport schemes for DCNs.

In particular, an important class of applications called Online Data-Intensive (OLDI) [7,8] applications are popularly deployed in data centers. Such applications must serve users in a timely manner since even one more millisecond of latency may considerably affect the revenue [9], and they typically need to query and aggregate results from a large number of nodes. To meet strict response deadlines, applications may send out incomplete responses to users even when some internal query results are not ready yet. This further requires applications' internal traffic flows to meet their respective deadlines as much as possible, since fewer missed deadlines means better response quality, and hence more revenue. Old deadline-agnostic transport schemes like TCP or DCTCP [6] do not work well for these applications, and deadline has been explicitly taken into consideration in modern data center flow rate control [8,10] or flow scheduling [11] algorithms.

\* Corresponding author at: Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China.

E-mail address: [shixg@cernet.edu.cn](mailto:shixg@cernet.edu.cn) (X. Shi).

In this paper, we first focus on the design of deadline-aware flow rate control mechanisms for DCNs. We start our work based on D<sup>2</sup>TCP [8]. By careful investigations on D<sup>2</sup>TCP's performance under different traffic load scenarios, we notice that under high traffic load, it degenerates to DCTCP and becomes nearly deadline-agnostic, which is troublesome since traffic bursts in the aggregation stage of OLDI applications often cause link congestion. With this in mind, we argue that when the network load is heavier, relatively more network bandwidth should be allocated to flows with tighter deadlines, and propose a simple deadline-aware flow rate control principle: *flows with different imminence should be differentiated in their bandwidth allocation/occupation, and the more traffic load, the more differentiation should be made.*

To demonstrate the effectiveness of this principle, we have also designed a simple congestion window based rate control algorithm called Load Proportional Differentiation (LPD) as its application. The basic idea of LPD is that, when using deadline to regulate the congestion window, traffic load is introduced as a multiplicative factor, so that the extent of regulation, as well as the difference of that extent, is proportional to traffic load. Using typical data center topologies and various traffic load scenarios, we evaluate LPD's performance and compare it with those achieved by some latest data center flow rate control algorithms, including DCTCP [6], D<sup>2</sup>TCP [8], and L<sup>2</sup>DCT [12], Karuna [2]. LPD nearly always outperforms them in enabling more flows to meet their deadlines. In typical scenarios, compared with D<sup>2</sup>TCP, it can reduce the number of flows missing their deadlines by more than 25%. Also, compared with the best known deadline-aware method Karuna, LPD performs about 5% worse than it on average. But for some heavy congestion cases, because of the principle of more load, more differentiation, LPD performs about 10% better than Karuna. Besides, we also show our principle is quite general, since its realizations in forms other than LPD can achieve comparable results, and it can be easily adapted for other tasks, such as reducing flow completion time. We modify LPD to let it fit to reduce flow completion time and then compare its performance with the state-of-art window-based protocol L<sup>2</sup>DCT and find it can reduce flow completion time by about 30%. Even compared with state-of-art schedule method pFabric [13], LPD performs about 20% worse than it on average. We have modeled LPD and implemented LPD in both ns2 [14,15] and Linux kernel 3.2.61 [16].

The key contributions of this paper are:

- We uncover scenarios where modern deadline-aware flow rate control mechanisms in DCNs become less effective.
- We propose a simple principle for designing deadline-aware flow rate control mechanism in DCNs, and mathematically derive the sufficient and necessary conditions for a scheme to follow this principle.
- We propose the LPD algorithm as an application of our principle and build a fluid model to analyze its performance with different parameters.
- We implement LPD in both ns-2 and linux kernel 3.2.61 and thoroughly evaluate its performance against latest deadline-aware as well as flowsize-aware rate control mechanisms. Simulation code of LPD can be downloaded at [15] and linux kernel source code of LPD can be found at [16].
- We also show the generality of our principle by other forms of rate control algorithms and applications.

The rest of our paper is organized as follows. We first introduce some necessary background and related work in Section 2. Then in Section 3, we describe the more load, more differentiation principle, which is motivated by careful investigations on D<sup>2</sup>TCP. The LPD algorithm is presented and analyzed using fluid model in Section 4, and is evaluated in Section 5 both by simulation and

real testbed. Some further discussions are given in Section 6, and finally, Section 7 concludes the paper.

## 2. Background and related work

The ever-increasingly used data center networks, although often have ultra high bandwidth and low latency links, still face frequent network congestions [17,18]. As a result, throughput collapse or high latency caused by congestion may seriously affect the applications which use DCN as their underlying facility. In particular, a class of Online Data-Intensive (OLDI) applications [7] are very sensitive to the service responsiveness, and often require their internal traffic flows to complete before certain hard deadlines. For these applications, network congestion control or scheduling algorithms have to be specifically tailored, so that flows, especially those short and bursty query flows, can finish before their deadlines.

Among many TCP-like transport schemes in DCNs, DCTCP [6] proposes to mark packets on a switch when its instantaneous queue length exceeds a certain threshold  $k$ . The endpoints then estimate the extent of congestion by the marked packets, and throttle flow rates in proportion to that extent. For a congestion level estimation  $\alpha$ , the multiplicative decrease algorithm for the congestion window  $w$  now becomes  $w = w \times (1 - \alpha/2)$  instead of being halved in TCP. By setting an appropriate  $k$ , DCTCP elegantly reduces the queue length and its variability on switches, and can reduce the queueing delay and network congestion. Extensive experiments show that DCTCP effectively provides high burst tolerance and low latency for short flows. However, its deadline-agnostic fair sharing of bandwidth among flows with different deadlines make it less effective for the deadline-sensitive OLDI applications, such as web query and advertisement [8].

To this end, deadline-aware protocols have been proposed to explicitly take flow deadline into consideration when allocating bandwidth, which is either computed explicitly by switches [10,11], or adjusted implicitly by varying the congestion window size on endpoints [8].

D<sup>3</sup> [10] computes the required bandwidth on the switches in a centralized fashion, and grants a sender's request for bandwidth accordingly. Although it is the first known deadline-aware transport protocol and improves upon DCTCP, it is shown to work bad in some race conditions where far-deadline requests arrive slightly ahead of near-deadline requests due to its greedy and first-come-first-service policy [8]. On the other hand, D<sup>3</sup> requires switch hardware change, and cannot coexist with TCP, due to its request-reply mechanism for bandwidth allocation.

D<sup>2</sup>TCP improves on DCTCP by adjusting an endpoint's congestion window more intelligently. It defines a deadline imminence factor  $d = T/D$ , where  $T$  is the time needed for a flow to complete its transmitting under a deadline-agnostic manner, while  $D$  is the time remaining until its deadline expires. Both the network congestion extent  $\alpha$  and the deadline imminence factor  $d$  are used in the multiplicative decrease of D<sup>2</sup>TCP's congestion window as  $w = w \times (1 - \alpha^d/2)$ , where  $\alpha^d$  is the well-known gamma-correction function [19], and is used as a penalty function here. In this way, D<sup>2</sup>TCP not only possesses DCTCP's nice property of keeping queue length steady and small, but also favors flows with tighter deadlines. It can co-exist with TCP, and is easy to implement in real data centers.

PDQ [11] uses preemptive flow scheduling at switches to assist bandwidth allocation. Unlike D<sup>3</sup>, PDQ works in a distributed fashion, and its preemptive scheduling can deal with the race conditions that threaten D<sup>3</sup>. PDQ can emulate different scheduling policies like Earliest Deadline First (EDF) or Shortest Job First (SJF). However, SJF is the just optimal solution to minimize flow completion time over the single link, for complex link it is far

optimal. Flow completion time is used as the evaluation metric in [11], while their performance in completing flows before hard deadlines has not been investigated thoroughly. Besides, PDQ also requires switch hardware change and cannot coexist with legacy TCP.

TIMELY [20] is RTT-based protocol. Instead of using packets dropping or packets marking, TIMELY uses RTT as the congestion signal. However, RTT in data center network is too short to measure accurately and it can not co-exist with packets-drop scheme which makes it hard to deploy widely.

Another class of related algorithms are specifically optimized to reduce flow completion time (FCT) [12,13] or completion time “tail” [21]. L<sup>2</sup>DCT [12] adjusts both the additive increase and multiplicative decrease of an endpoint’s congestion window size, where the gamma-correction function is also used ( $cwnd = cwnd * (1 - \alpha^{w_c})$ ), similar to that in D<sup>2</sup>TCP but with a different imminence factor. On the other hand, pFabric [13] schedules flows according to their remaining sizes. The proposed distributed algorithm approximates the Shortest Job First (SJF) policy, and is proved to be near optimal (i.e., within a constant factor) for minimizing the average FCT. Compared with PDQ, pFabric decouples flow scheduling from rate control, so it is much simpler to implement, and can co-exist with TCP-like schemes. Other work such as [22] claims protocols (DCTCP, PDQ, pFabric, etc) are not competitors, they are complementary. PASE does not change network elements and its performance is better than the those state-of-art protocols [22]. Another work [21] studies cross-layer techniques, including flow prioritization and efficient load balancing, to reduce the long tail of flow completion times. Although all of these schemes are effective in reducing FCT, thus being able to accelerate service response and improve user experience, their performance in OLDI applications with hard deadlines may not be the same.

FDRC [3] and Karuna [2] are systematic solutions for scheduling mix-flows. They advocate that deadline flows should meet their deadlines while minimally impacting the FCT of non-deadline flows. Indeed, both methods can reduce the percentage of flows missing deadline as well as cutting down the average flow completion time. FDRC emphasizes on reducing the latency of short flows. However, in data center network, some long live flows may also have time constraint and using FDRC may make these flows miss deadline. Karuna has to do change both at switch and end nodes. This makes it hard to deploy in practice.

For the objective to let more flows finish before their deadlines, [3,23] praised the idea that rate control and flow scheduling should be decoupled [13], and focus on the first part.

### 3. More load, more differentiation

Our work borrows ideas heavily from earlier rate control schemes in DCNs, such as DCTCP, D<sup>2</sup>TCP and L<sup>2</sup>DCT. In particular, we have exactly the same object as D<sup>2</sup>TCP, so some further analysis of D<sup>2</sup>TCP is worth it. Through combining a deadline-aware penalty function with DCTCP, D<sup>2</sup>TCP essentially allocates bandwidth unfairly to flows with different deadlines, thus achieves better performance in reducing deadline misses. We believe that it is a significant guideline, and also argue that it is insufficient. In Section 3.1, we start with a simple scenario where D<sup>2</sup>TCP does not perform as well as expected. Then in Section 3.2, we analyze the root cause of this “bad case”. We propose the deadline-aware rate control principle in Section 3.3, where we also derive the sufficient and necessary conditions for satisfying it.

#### 3.1. A motivating example

We use ns-2 to conduct a simulation as follows. Assume four flows share a bottleneck link of 1 Gbps. They start at the same

**Table 1**

Untransmitted data after deadline.

|                        | D <sup>2</sup> TCP | DCTCP   |
|------------------------|--------------------|---------|
| flow 1 (8 MB, 200 ms)  | 1.53 MB            | 2.07 MB |
| flow 2 (16 MB, 400 ms) | 1.79 MB            | 2.14 MB |
| flow 3 (32 MB, 600 ms) | 3.64 MB            | 4.08 MB |
| flow 4 (40 MB, 800 ms) | 0 MB               | 0 MB    |

time, with flow sizes of 8, 16, 32 and 40 MB, and deadlines of 200, 400, 600 and 800 ms, respectively, with a RTT of 100 us. These parameters are chosen such that they have different kinds of urgency, but can at least finish before the last deadline expires, if standard TCP is used.

The flow transmission results are listed in the forth column of Table 1, where each number represents how much data is left untransmitted when the corresponding flow deadline expires (so a number greater than 0 means that a flow misses its deadline, after which that flow is stopped by force). For comparison, the transmission results under the deadline-agnostic DCTCP are also listed in the last column. We can see that, although D<sup>2</sup>TCP transmits more data before flow deadlines than DCTCP does, it cannot effectively reduce the number of flows missing their deadlines. In both cases, only flow 4 gets finished before its deadline, and in this sense, D<sup>2</sup>TCP is only as good as DCTCP, if perceived by applications.

Fig. 1 provides more details on the internal status of D<sup>2</sup>TCP. As we know, D<sup>2</sup>TCP uses  $w = w \times (1 - \alpha^d/2)$  to adjust its congestion window size  $w$ , where  $\alpha$  is the estimated extent of congestion, and  $d$  is a deadline imminence factor. In Fig. 1(a), the bandwidth occupied by each flow is depicted, where we see no much difference between different flows in most of the time. Furthermore, the congestion window sizes of flow 1 and flow 4 during the first 200 ms are plotted in Fig. 1(b), which shows little difference either, except in the last 10 ms before flow 1 misses its deadline. However, the imminence factors of the two flows, as shown in Fig. 1(c), indeed differ much greater, especially during the time from 100 ms to 200 ms. For completeness, we also plot  $\alpha$ , the extent of congestion, in Fig. 1(d), which shows flow 1 and flow 4 achieve a comparable estimation on  $\alpha$ .

#### 3.2. Analysis of the gamma-correction function

Now we take a closer look at D<sup>2</sup>TCP’s rate control mechanism, trying to dig out why it performs badly in the scenario above. The central idea of D<sup>2</sup>TCP is to use the gamma-correction function in its penalty function as  $p(\alpha, d) = \alpha^d$ , where  $0 < \alpha < 1$ . We plot the curves of  $p$  for  $d = 0.4$  and  $d = 0.6$  in Fig. 2(a). When  $\alpha$  increases,  $p$  also increases, resulting in a smaller congestion window due to heavier load. And when  $d$  increases,  $p$  decreases, resulting in a larger congestion window due to greater urgency. These properties are indeed reasonable and necessary for deadline-aware rate control. However, if we consider the difference between the penalties,  $pd(\alpha, d_1, d_2) = p(\alpha, d_1) - p(\alpha, d_2) = \alpha^{d_1} - \alpha^{d_2}$  where  $d_1 < d_2$ , we can find it first increases as  $\alpha$  increases, and then decreases after  $\alpha$  passes a certain point (around 0.2 in Fig. 2(b)). For example,  $pd(0.4, 0.4, 0.6) = 0.116$ ,  $pd(0.6, 0.4, 0.6) = 0.079$ , and  $pd(0.9, 0.4, 0.6) = 0.019$ . This means, at a relatively heavier load, the ability that D<sup>2</sup>TCP throttles a more urgent flow less greatly than it throttles a less urgent flow tends to diminish, and becomes a near fair-sharing scheme. This behavior, as we believe, is not appropriate for deadline-aware rate control.

Now we can explain why D<sup>2</sup>TCP does not perform well for the flows in the example above. In particularly, let’s examine the interval from 150 ms to 160 ms, during which  $\alpha$  increases approximately from 0.4 to 0.6. In the same interval, the imminence factors of flow 1 and flow 4 are around  $d_1 = 0.6$  and  $d_4 = 0.4$ ,

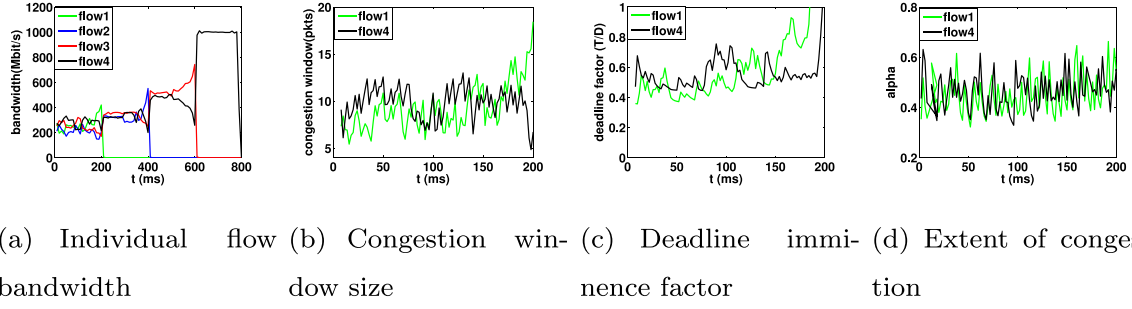


Fig. 1. D²TCP's performance under four concurrent flows.

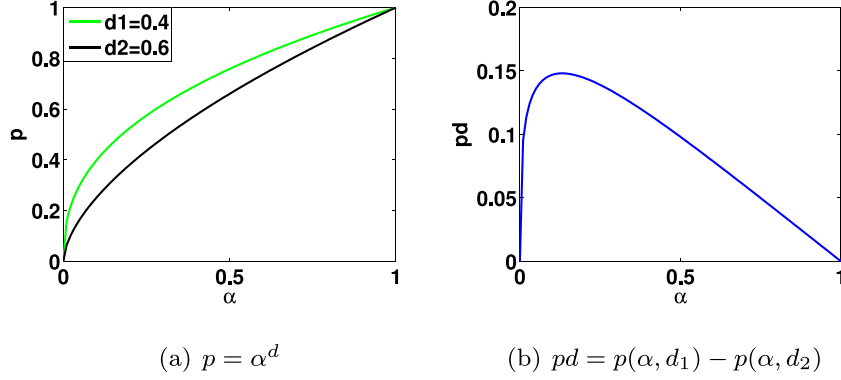


Fig. 2. D²TCP uses the gamma-correction function to decrease its congestion window.

respectively, which means flow 1 is more urgent than flow 4. However,  $pd(0.4, d_4, d_1) = 0.116$  while  $pd(0.6, d_4, d_1) = 0.079$ . Thus, when the load becomes heavier, as shown in Fig. 1(b), the extent that flow1's window gets throttled becomes closer to that of flow 4, resulting a similar window size (notice that  $pd(0.9, d_4, d_1) = 0.019$ ). Thus, flow 1 cannot grab enough bandwidth to finish before its deadline. What we would have expected here is that, flow 1 reduces its window at a even smaller pace than flow 4 does, as  $\alpha$  increases.

### 3.3. Principle for deadline-aware flow rate control

For OLDI applications, the traffic load can often be high due to bursts of concurrent query flows. With this in mind, we advocate that, when designing a deadline-aware flow rate control scheme, the following principle should be respected:

- Principle of Imminence (PI): the rate of a more imminent flow should be higher than that of a less imminent flow.
- Principle of Differentiation (PD): the difference between the rates of two flows with different levels of imminence should be increased when the traffic load becomes heavier.

The first part of this principle (i.e., PI) says *flow rates should be differentiated by their imminence*, and can be intuitively explained as follows. Consider two flows of the same size. The flow with a tight deadline should use more bandwidth than the flow with a lax deadline, so that their deadlines can be met with a greater probability. Apparently, this part is already implemented in the modern deadline-aware TCP schemes, such as D³ or D²TCP. However, as explained above, they miss the relationship between congestion and imminence.

The second part of our principle (i.e., PD) says *the more heavier the load is, the greater differentiation between flow rates should be made*, and can be intuitively explained in the following way. Compared to light loads, heavier loads may be regarded, in a plausible way, as evenly reducing the bandwidth each flow can get.

So deadlines are more difficult to met even though flows are already differentiated as in a light load. Then to meet the tight deadlines as much as possible, the corresponding flows should seize some bandwidth from the other flows. In other words, the relative bandwidth differentiation between the two flows should be greater than that in a light load. In this way, it is more probable that both the tight and the lax deadlines can be met. Modern deadline-aware rate control schemes overlook this point, and result in less effectiveness.

Formally, suppose a rate of  $r(\alpha, d)$  is allocated to or occupied by a flow with a deadline imminence factor  $d$  ( $d > 0$ ) under a traffic load factor  $\alpha$  ( $0 < \alpha < 1$ ), where greater  $d$  and  $\alpha$  represent greater imminence and load, respectively. Then Theorem 1 states how to apply the above principle to regulate  $r$ , if we only consider  $r$  of a good shape, i.e., always differentiable with respect to  $\alpha$  and  $d$ .

**Theorem 1.** The sufficient and necessary condition for  $r$  to follow the above principle is  $\frac{\partial r}{\partial d} > 0$ , and  $\frac{\partial^2 r}{\partial d \partial \alpha} \geq 0$  (but does not constantly equal 0).

**Proof.** The first equation corresponds to PI, and is just the basic property of a monotonically increasing function. It suffices to prove the second equation, corresponding to PD.

Define  $dr(\alpha, d_1, d_2) = r(\alpha, d_2) - r(\alpha, d_1)$ . Then we have

$$\begin{aligned} \frac{\partial dr(\alpha, d_1, d_2)}{\partial \alpha} &= \frac{\partial [r(\alpha, d_2) - r(\alpha, d_1)]}{\partial \alpha} \\ &= \frac{\partial r(\alpha, d_2)}{\partial \alpha} - \frac{\partial r(\alpha, d_1)}{\partial \alpha} \\ &= \int_{d_1}^{d_2} \frac{\partial^2 r}{\partial \alpha \partial d} dd. \end{aligned} \quad (1)$$

On the other hand, PD is equivalent to say  $dr(\alpha_2, d_1, d_2) > dr(\alpha_1, d_1, d_2)$  whenever  $\alpha_1 < \alpha_2$  and  $d_1 < d_2$ , which is true if and only if  $\frac{\partial dr(\alpha, d_1, d_2)}{\partial \alpha} > 0$  whenever  $d_1 < d_2$ . This, according to Eq. (1), holds true if and only if  $\frac{\partial^2 r}{\partial \alpha \partial d} \geq 0$  but does not constantly equal 0.  $\square$

We believe the proposed principle is simple and general, while how to quantitatively measure the traffic load and deadline imminence, and how to implement the rate regulation function with  $\alpha$  and  $d$  factors, can be determined depending on specific applications. Besides, the relative rate change, instead of the absolute rate, can also be guided in a similar way. In the next section, we will develop a concrete deadline-aware flow control scheme following this principle.

#### 4. Load proportional differentiation

##### 4.1. Basic algorithm

As an application of the *more load, more differentiation* principle, we propose the following deadline-aware congestion control algorithm:

$$w = \begin{cases} w + (1 - f) & \text{without congestion;} \\ w \times (1 - f) & \text{with congestion} \end{cases} \quad (2)$$

where  $f = \alpha/d$  is the penalty applied to the congestion window size according to traffic load  $\alpha$  and flow deadline imminence  $d$ , and is capable to differentiate between flows. The extent of congestion (i.e., the traffic load factor  $\alpha$ ), is estimated using the same mechanisms as that in DCTCP and D<sup>2</sup>TCP [6,8]. Since  $f$  is proportional to  $\alpha$ , we call this algorithm Load Proportional Differentiation.

We start with a simple imminence factor  $d = t_{\max}/t$ , where  $t$  is the duration between a flow's start time and its deadline, and  $t_{\max}$  is a tunable upper bound of  $t$ , which will be discussed in more detail later. For flows without deadlines,  $t$  is simply set to  $t_{\max}$ . So a smaller  $t$ , thus a larger  $d$ , means greater imminence. Now we get our first algorithm named LPD-t:

$$w = \begin{cases} w + (1 - \alpha \times \frac{t}{t_{\max}}) & \text{without congestion;} \\ w \times (1 - \alpha \times \frac{t}{t_{\max}}) & \text{with congestion} \end{cases} \quad (3)$$

The algorithm is rather simple. It computes  $f = \alpha \times \frac{t}{t_{\max}}$  and uses it to adjust the congestion window size  $w$ . If there is no congestion in the last window of data,  $w$  is increased by  $1 - f$ , while if any congestion occurs,  $w$  is decreased by a fraction of  $f$ .

##### 4.2. Analysis of LPD

###### 4.2.1. Simple property of LPD

In TCP-like schemes, flow rate is implicitly adjusted by regulating congestion window. Thus we can roughly regard the change of congestion window size as some kind of reflection of flow rate. In this sense, we have the following proposition.

**Proposition 1.** LPD respects PI and PD.

**Proof.**  $\frac{\partial(1-f)}{\partial d} = \frac{\partial(1-\alpha/d)}{\partial d} = \frac{\alpha}{d^2} > 0$ , and  $\frac{\partial^2(1-f)}{\partial d \partial \alpha} = \frac{1}{d^2} > 0$ , so the regulation on congestion windows size enforced by LPD satisfies the conditions in Theorem 1, and thus respects PI and PD.  $\square$

It is also quite easy to understand why LPD respects PI and PD from a geometric view. Since  $f$  is proportional to  $\alpha$ , a scatter plot of  $(\alpha, f)$  will be a straight line across the origin. The distance between two such lines, which correspond to different  $f$  under different  $d$ , just increases with  $\alpha$ .

LPD-t has another property under some wild assumptions.

**Proposition 2.** Suppose LPD-t can lead to a longterm steady state, that is the congestion window of each flow steadily (and synchronously) cycles within a small fixed interval, implying a steady flow rate and congestion extent. If  $t \ll t_{\max}$ , where  $t$  and  $t_{\max}$  are parameters used by LPD-t in Eq. (3), then approximately, the steady rate a flow achieves is inversely proportional to its corresponding  $t$ .

**Proof.** In a steady state, the congestion window size of a flow behaves like a periodic sawtooth with a fixed period  $N$ . Using  $W_{\min}$  and  $W_{\max}$  to represent the minimum and maximum window size, then according to Eq. (3), we have  $W_{\max} = W_{\min} + N \times (1 - \alpha \times \frac{t}{t_{\max}})$ , and  $W_{\min} = W_{\max} \times (1 - \alpha \times \frac{t}{t_{\max}})$ , from which we get  $W_{\max} = N \times (1 - \alpha \times \frac{t}{t_{\max}}) / (\alpha \times \frac{t}{t_{\max}})$ , and the average window size  $W_{av} = (1 - \alpha \times \frac{t}{2t_{\max}}) \times W_{\max}$ .

For two flows  $f_1$  and  $f_2$  with time duration  $t_1$  and  $t_2$  between their start points and deadlines, suppose their maximum window sizes are  $W_1$  and  $W_2$ , and their steady flow rates are  $r_1$  and  $r_2$ , respectively. Then the ratio between  $r_1$  and  $r_2$  is

$$\begin{aligned} \frac{r_1}{r_2} &= \frac{(1 - \alpha \times \frac{t_1}{2t_{\max}}) \times W_1}{(1 - \alpha \times \frac{t_2}{2t_{\max}}) \times W_2} \\ &= \frac{t_2}{t_1} \times \frac{2t_{\max} - \alpha \times t_1}{2t_{\max} - \alpha \times t_2} \times \frac{t_{\max} - \alpha \times t_1}{t_{\max} - \alpha \times t_2} \end{aligned}$$

If  $t \ll t_{\max}$ , then  $r_1/r_2 \approx t_2/t_1$ .  $\square$

Although the long-term stability of LPD-t has not been proved, and in reality,  $t$  cannot be much smaller than  $t_{\max}$ , our simulation results demonstrate that the calculations are fairly accurate. We start two long flows with LPD-t, both of 100 MB, but with different deadlines ( $t_1 = 5$  s,  $t_2 = 10$  s, while  $t_{\max} = 50$  s). Fig. 3(a) depicts the bandwidth evolution of the flows at the first 800 ms, while Fig. 3(b) depicts the actual bandwidth ratio  $r_1/r_2$ , which is close to  $t_2/t_1 = 2$ .

###### 4.2.2. A simple fluid model of LPD

A simple fluid model for LPD can be built following the procedure similar to those in [24] and [25]. Assume  $N$  simultaneous flows pass to the same receiver node through the bottleneck link whose capacity is  $C$  pkts/sec. Then the fluid model can be described as:

$$\frac{dw_i}{dt} = \frac{1 - \alpha_i(t) \frac{t_i}{t_{\max}}}{R(t)} - \frac{w_i(t) \alpha_i(t) \frac{t_i}{t_{\max}}}{R(t)} p(t - R^*) \quad (4)$$

$$\frac{d\alpha_i}{dt} = \frac{g}{R(t)} (p(t - R^*) - \alpha_i(t)) \quad (5)$$

$$\frac{dq}{dt} = \sum_{i=1}^N \frac{w_i(t)}{R(t)} - C \quad (6)$$

$$\hat{p}(t) = 1_{\hat{q}(t) > 1} \quad (7)$$

Where,  $w_i$  denotes flow <sub>$i$</sub> 's congestion window.  $t_i$  is the deadline of flow <sub>$i$</sub>  and  $\alpha_i$  presents the congestion level of flow <sub>$i$</sub> .  $q$  is the length of switch queue size and  $R$  presents RTT. Eqs. (5) and (7) describe the ECN marking process at the switch. Eq. (4) models the change of the congestion window and Eq. (6) describes queue.

With the model, we can compute the window size and queue length. To test the model, we use a simple scenario of two long flows ( $t_1=10$  s,  $t_2=20$  s,  $t_{\max}=40$  s) and check their window size, the queue length and the congestion level under both ns2 and the fluid model with same parameters. The simulation results are compared with the fluid model results in Fig. 4(a), Fig. 4(b) and Fig. 4(c) respectively, where we can see they match quite well.

We can see congestion window,  $\alpha$  and queue length match quite well. To further test the accuracy of the model, we start 2 flows ( $t_1=0.5$  s,  $t_2=1$  s) sharing a bottleneck of 10 Gbps. The packet size is 1500 byte and RTT is 100 us,  $K$  is 64. Fig. 5 demonstrates the results. From Fig. 5(a), we can see that when  $t_{\max} < 100$ , both fluid model and ns-2 simulation have the value of  $w_1/w_2 \approx 2$ . However, when  $t_{\max} > 100$ , ns-2 simulation's  $w_1 \approx w_2$  and fluid model's  $w_1/w_2 \approx 2$ . Fig. 5(b) demonstrates when  $t_{\max} > 100$ , fluid model

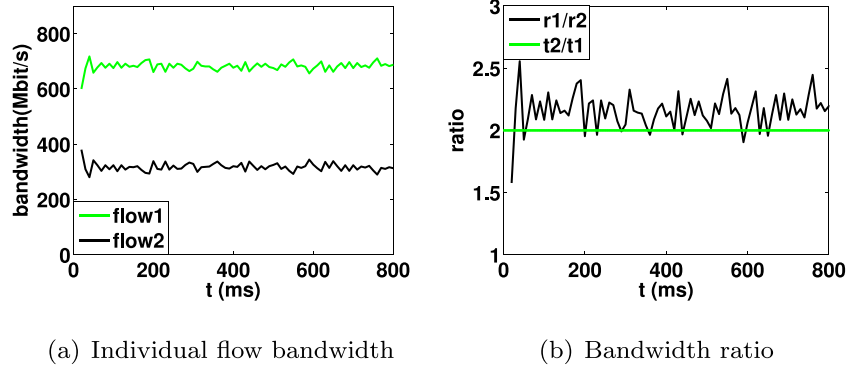


Fig. 3. Two flows under LPD-t.

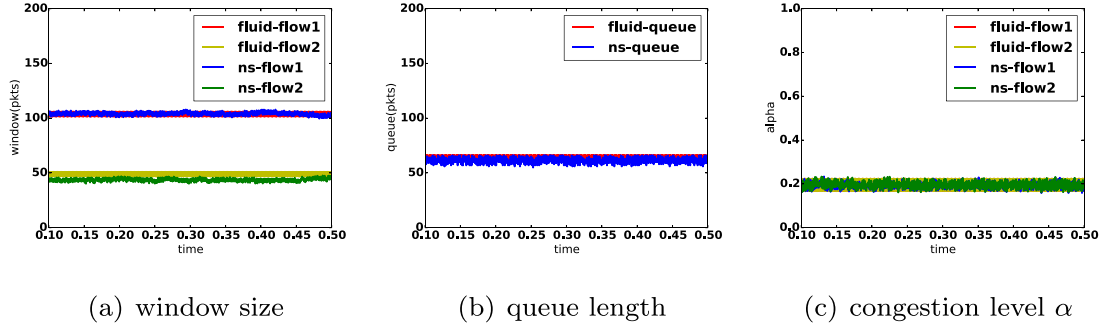
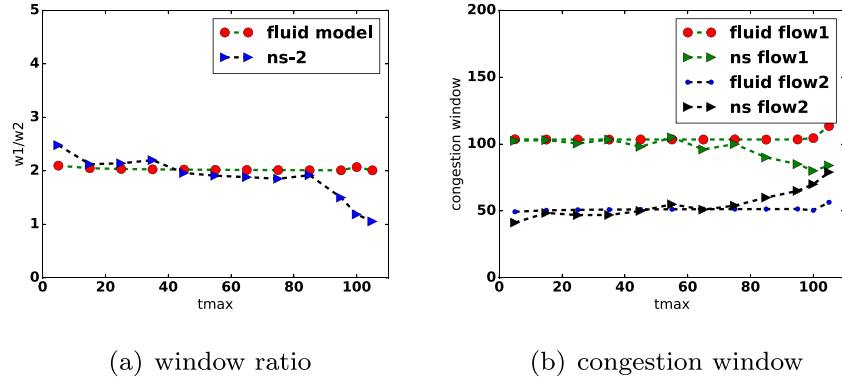


Fig. 4. Details comparison between fluid model and ns-2 simulation.

Fig. 5. Window comparison with different  $t_{\max}$ .

has a large value of congestion window. This result suggests that proper value of  $t_{\max}$  should be set in practice, which will be discussed in the next section.

Note that, in the fluid model, a flow has a fixed point  $(w_i, \alpha_i) = (\frac{t_{\max} - t_i}{t_i}, 1)$ , which means when  $\alpha=1$  (i.e., all packets are marked under heavy congestion), all flows have a fixed congestion window size.

#### 4.2.3. Choosing $t_{\max}$

In LPD,  $t_{\max}$  should be at chosen in a reasonable range. In this part, we analyze the range of  $t_{\max}$ . Assume there are  $N$  synchronized flows ( $t_{\max} > t_1 > t_2 > \dots > t_n$ ) starting simultaneously, queue length at time  $t$  denotes as  $q(t)$ , for the fixed point time, then we have:

$$\frac{t_{\max} - t_1}{t_1} + \frac{t_{\max} - t_2}{t_2} + \dots + \frac{t_{\max} - t_n}{t_n} < q(t) + C \times RTT \quad (8)$$

Assume marking threshold is  $K$ . For the  $N$  flows, as every RTT, window increment is smaller than 1, so we have  $q(t) < K + N$ . Be-

cause  $t_{\max} > t_1 > t_2 > \dots > t_n$ , so have:

$$t_{\max} < \frac{t_1 \times (K + C \times RTT + 2N)}{N} \quad (9)$$

From (9) we know  $t_{\max}$  can be set according to the maximum value of flow deadline, maximum concurrent flow number, RTT and link bandwidth. Assume, for the common modern data center network,  $C$  is 10 Gbps, packet size is 1500 B, RTT is 100 us, concurrent number of connection is less than 100 for most cases [6], Setting  $K$  to 60, we have:

$$t_1 < t_{\max} < 3.5 \times t_1 \quad (10)$$

$t_{\max}$  can be computed according to (9) when deploying LPD in practice. Default setting of  $t_{\max}$  can be set according to (10). Indeed, this result is consistent with Fig. 5(b). According to (10), when  $N = 2$ ,  $t_{\max} \leq 76 \times t_1$ . In this case,  $t_1 = 1$  s. We can see from ns-2 simulation, when  $t_{\max} > 80$ . Flows' congestion windows are no longer inversely proportional to their corresponding  $t_i$ . For more detail of  $t_{\max}$ , we will discuss later at the evaluation part.

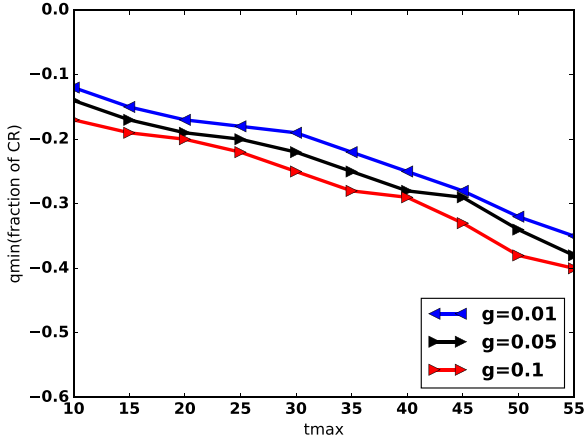


Fig. 6. Queue undershoot with different  $t_{\max}$ .

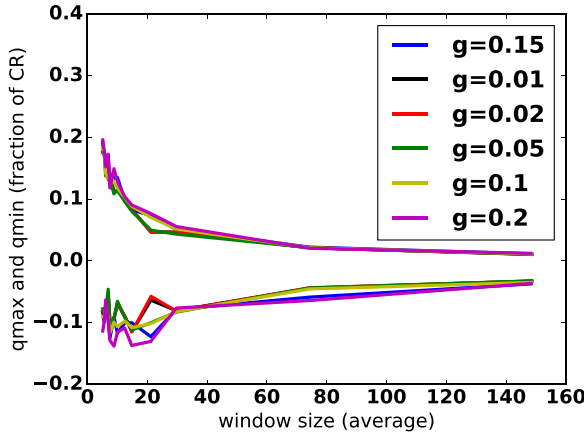


Fig. 7. Queue undershoot and overshoot.

#### 4.2.4. Choosing an appropriate marking threshold $k$ to achieve 100% throughput

One may ask: how much buffering is required for LPD to achieve 100% throughput? Since queue underflow must be avoided for 100% throughput, we need to determine how large the queue size oscillations are about the marking threshold  $K$  [24].

Let  $q_{\min}$ ,  $q_{\max}$  denote minimal and maximal size of queue during the period. The marking threshold  $K$  should be larger than the undershoot oscillations (else queue size will be zero and link utility will not be 100%). We use our fluid model and set flows' deadlines within 10, then we vary  $t_{\max}$  from 10 to 55 and vary the link capacity from 1 Gbps to 10 Gbps. Finally, we write down the minimal queue size undershoot for each group of  $t_{\max}$ . Fig. 6 shows the result. We can see for small value of  $g$ , even the worst case ( $t_{\max} = 55$ ,  $t_{\max} \simeq 5t_1$ ), we have:

$$K \gtrsim 0.4CR \quad (11)$$

As for RTT, we have  $R = d + K/C$ , so for the worst case, minimal queue size can be suggested as  $0.67Cd$ . This means, for the link capacity  $C = 10$  Gbps, link propagation is 100  $\mu s$ , packet size is 1500 B,  $K$  should be at least be set as 56.

For a common case in data center, where link propagation delay is 100  $\mu s$  and link capacity to 10 Gbps.  $t_{\max} = 10$ , and flow deadline randomly generated within 1 to 10. Use fluid model, Fig. 7 shows  $q_{\min}$  and  $q_{\max}$ 's against average window size of flow<sub>1</sub> for some values of  $g$ . We can see  $K \gtrsim 0.12CR$  for small value of  $g$ . To avoid queue underflow, we require  $K > |q_{\min}|C \cdot R \simeq 0.12CR$ . Experiment methods are same as [24]. As  $R = d + K/C$ , we have:

$$K \simeq 0.14Cd \quad (12)$$

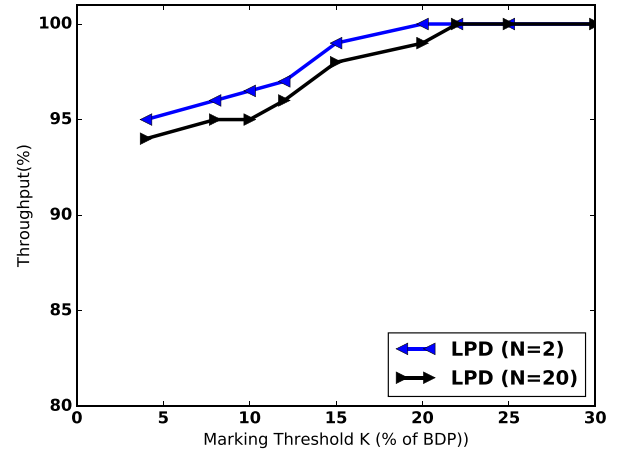


Fig. 8. Throughput vs marking threshold  $K$  in ns2 simulation;  $K$  is varied from 4 to 30 packets (2.4–18% of the BDP). Note that as predicted by the analysis, the throughput remains higher than 95% even for  $K$  as small as 3% of the BDP.

In words, about 14% of the bandwidth-delay product of buffering is needed for 100% throughput; any more available buffer can be used as headroom to absorb bursts.

To test this, Fig. 8 shows throughput vs marking threshold  $K$  in ns2 simulation; In this experiment, we varied from 4 to 30 packets (2.4–18% of the BDP). Result shows that the throughput remains higher than 95% even for  $K$  as small as 3% of the BDP. ns-2 simulation's result is similar to LPD fluid model.

#### 4.3. Taking flow size into consideration

Deadline-aware OLDI applications often set deadlines for their small query flows, so we extend LPD-t by taking flow size  $s$  into consideration, and propose to use a penalty function

$$f = \alpha \times \frac{t}{t_{\max}} \times \frac{s}{s_{\max}} \quad (13)$$

where  $s_{\max}$  is an upper bound for truncating a large  $s$ .

In this way, we hope when there is not enough bandwidth to meet all deadlines, at least those small flows may have a greater chance to meet their deadlines. Eq. (13) still preserves the property of load proportional differentiation, and we call the corresponding algorithm LPD-e. In our evaluations, LPD-e will be used, if not explicitly stated otherwise.

We note that, the “more load, more differentiation” principle is quite general, and can be implemented in different forms. We choose to present LPD just because its simplicity, and will discuss some other forms of functions in Section 6. On the other hand, LPD itself can also be further extended by incorporating other factors such as flow remaining size. We leave that for future studies.

## 5. Evaluation

In this section, we evaluate our design principle and the LPD algorithm by both ns-2 [14] simulation and testbed. We implement LPD (including LPD-t, LPD-e) based on the source code of DCTCP released at their website [26], and port the ns-3 code of D<sup>2</sup>TCP provided by its author to ns-2. In addition, we implement L<sup>2</sup>DCT, Karuna based on their pseudo algorithm and use the pFrabric code provided by the authors. All the code of our simulation can be downloaded at [15]. Meanwhile, we also implement LPD, D<sup>2</sup>TCP, L<sup>2</sup>DCT in Linux kernel 3.2.61 and evaluate them in a real testbed, and the corresponding source code can be downloaded at [16].

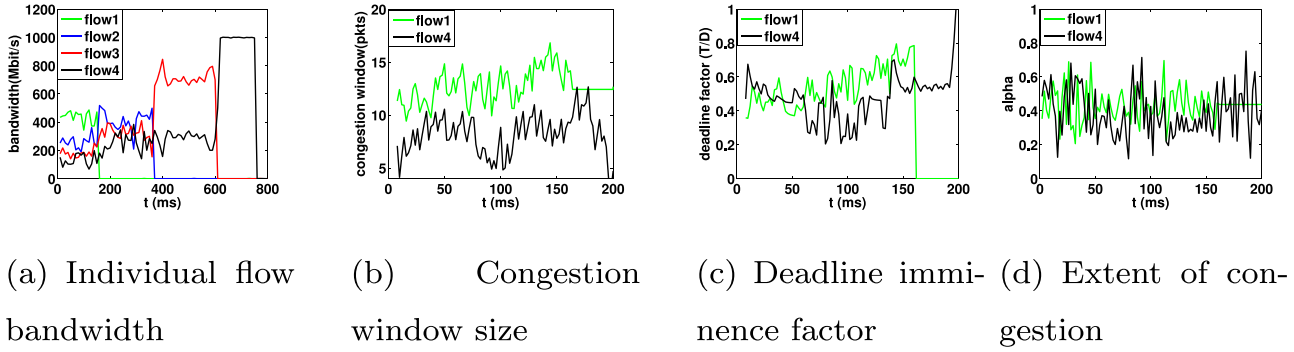


Fig. 9. LPD's performance under 4 concurrent flows, using the same imminence factor as D<sup>2</sup>TCP.

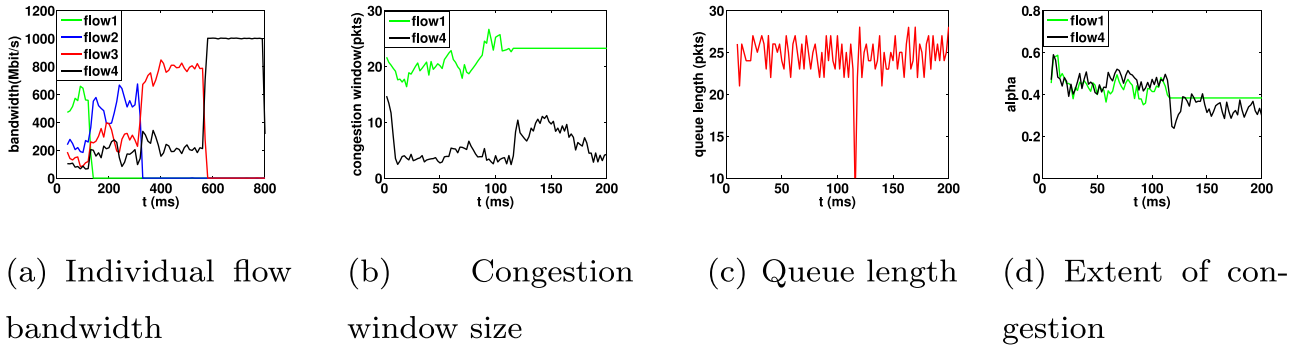


Fig. 10. LPD-t's performance under 4 concurrent flows.

### 5.1. Revisiting the motivating example

We first apply LDP to our motivating example presented in Section 3.1. Although that example is not a typical OLDI scenario, we use it to check whether LPD actually works for helping more flow meet their deadlines, especially under a heavy load. With the same setting, only flow 2 misses its deadline under LPD, which uses the same imminence factor  $d = T/D$  as that in D<sup>2</sup>TCP. To help a better understanding of why LPD outperforms D<sup>2</sup>TCP and DCTCP, we present more details on LPD's internal status in Fig. 9. The bandwidth of each individual flow, as well as the congestion window size  $w$ , the imminence factor  $d$ , and the traffic load  $\alpha$  of flow 1 and flow 4, are plotted.

Compare with the corresponding results in Fig. 1, we can see that the difference between the congestion windows of individual flows becomes much greater, while the imminence factor and traffic load do not change much. This effect is exactly caused by our choice of load proportional differentiation regulation, and results in more differentiated bandwidth allocation, as illustrated in Fig. 9(a). Ultimately, less deadline miss is achieved by LPD than by D<sup>2</sup>TCP.

For comparison, we also apply LPD-t to the same example using  $t_{\max} = 800$  ms. With LPD-t, all flows complete their transmission before their deadlines expire. This is because LPD-t favors flows with small  $t$ , and Fig. 10 depicts its internal status. Compared with that of LPD and D<sup>2</sup>TCP, although the traffic load does not change much, the difference between the bandwidth occupied flows with different levels of imminence is even greater. For example, during the first 200 ms, flow 1 gets about 55% bandwidth, while flow 4 only gets 10%.

Fig. 10(c) depicts the queue length at the switch during the first 200 ms, exhibiting a small fluctuation around the packet marking threshold  $k = 25$ . The sudden drop at 128 ms is caused by flow 1's finishing its transmission at that time, and then the queue is quickly filled up by the other flows.

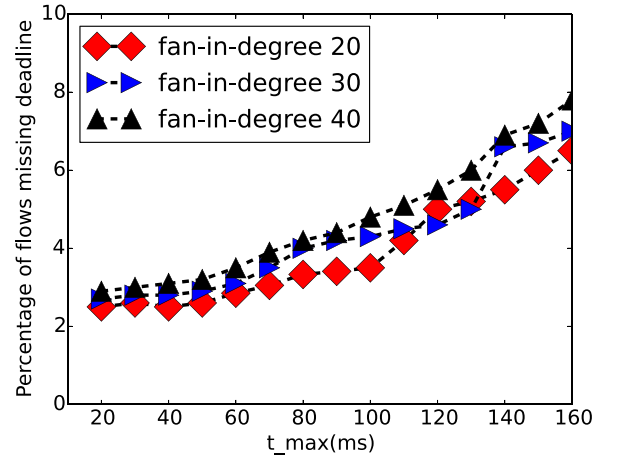


Fig. 11. The effect of varying  $t_{\max}$ .

### 5.2. Parameter selection

LPD-t and LPD-e have to set appropriate upper bounds  $t_{\max}$  and  $s_{\max}$ . We only discuss  $t_{\max}$  here since  $s_{\max}$  has a similar role. It can be seen from LPD-t's Eq. (3) that, if  $t_{\max}$  is larger, the penalty function  $f$  will be smaller. From (9) we know for  $C=1$  Gbps,  $K=25$ ,  $RTT = 1$  ms, max packet size = 1500 B and when concurrent number of flows is 20, 30, 40, the corresponding  $t_{\max}$  should be smaller than  $6 \cdot t_1$ ,  $4 \cdot t_1$ ,  $3 \cdot t_1$ . If  $t_{\max}$  is too large the ability to differentiate between flows with different  $t$  also becomes weaker. We use an OLDI application to check how different choices of  $t_{\max}$  may affect LPD-t's performance, where the deadline of flows are all set within a tight value  $t = 20$  ms, and  $t_{\max}$  is varied from 20 ms to 160 ms ( $t_1 < t_{\max} < 8 \cdot t_1$ ). Fig. 11 depicts the number of flows missing

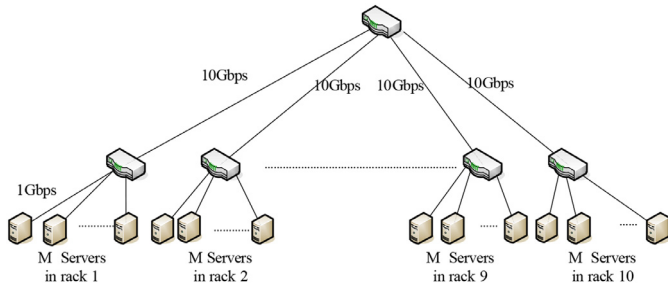


Fig. 12. The simple data center topology.

their deadlines under different  $t_{\max}$ , while the fan-in degree (i.e., the number of nodes that have to send query results back to a root node) is 20, 30 and 40, respectively.

Form Fig. 11, we can see that when  $t_{\max} < 80$  ms, which is no more than four times of the deadlines, LPD-t works quite well. Less than 5% flows may miss their deadlines, which is always better than D<sup>2</sup>TCP (the corresponding values under D<sup>2</sup>TCP at fan-in degrees 20, 30 and 40 are 3.45%, 4.96% and 5.45%, respectively), and there's not much difference for different  $t_{\max}$  in this range. However, when  $t_{\max}$  is too large, for example, when it is greater than 120 ms (i.e., six times of the deadlines), LPD-t's performance get greatly harmed, and becomes worse than D<sup>2</sup>TCP.

### 5.3. At scale simulation in simple data center topology

To demonstrate how LPD-e works for OLDI applications in data centers, we construct a three-level tree topology as shown in Fig. 12. We simulate ten racks, each of which consists of  $M = 40$  servers, connecting to a Top-of-Rack (TOR) switch by 1 Gbps links. All TOR switches then are connected to a root switch with 10 Gbps links. Using this topology, we evaluate the performance of LPD-e in a medium scale data center.

Following the suggestions [6,8,12], we set the packet marking threshold  $k = 20$  at the switches, and set the round trip propagation delay as 100  $\mu$ s. The TCP retransmission timeout (RTO) is set as 10 ms and initial cwnd size is 12. We consider small flows of sizes within 200 KB to be deadline-aware, and set  $s_{\max} = 200$  KB. The flow sizes follow a Pareto distribution with a Pareto shape parameter of 1.2, and the mean flow size is 50 KB. We set flow deadlines as 20, 30 and 40 ms, and set  $t_{\max} = 80$  ms. The fan-in degree of OLDI applications varies from 20 to 40, representing different levels of traffic load. These parameters are chosen to match those used in D<sup>2</sup>TCP. Besides, we also let ten servers generating long-lived background traffic. The start time of the small deadline-aware flows are nearly synchronized, as query results are sent back to the root node of the OLDI applications. Such a scenario may cause the Incast problem [17] if traditional TCP is used, since a large number of concurrent small flows sharing a bottleneck link often induce heavy congestion.

Fig. 13 shows the percentage of flows missing their deadlines under each setting, where DCTCP, D<sup>2</sup>TCP, L<sup>2</sup>DCT, Karuna and LPD-e are used as the transport layer scheme. The presented results include the average deadline miss percentage, and also the minimum and maximum ones for 100 random experiments with different seeds. We note that L<sup>2</sup>DCT is also a TCP-like congestion window based rate control scheme, using gamma-correction in a similar way to D<sup>2</sup>TCP, but with a different imminence factor. The results demonstrate the superiority of LPD-e, which in all experiments causes the smallest number of (sometimes zero) deadline misses due to its greater capability to differentiate flows under heavy load. Although D<sup>2</sup>TCP (as well as L<sup>2</sup>DCT) improves on DCTCP, it has a clear advantage over DCTCP only for lax deadlines (Fig. 13(c)), but not for tight deadlines (Fig. 13(a) and (b)). This effect was not

shown in [8] since their experiments used mixed deadlines. For the state-of-art deadline-aware flow schedule method Krna [2], it performs about 5% better than LPD-e on average. However, for large concurrent condition (such as fan-in degree is 40), the performance of Krna sometimes performs worse than LPD. The reason for this is that LPD can differentiate flows with different deadlines more effectively under heavier network congestion situation.

Then we test LPD-e under a different scenario, where servers send short flows randomly to each other, and we vary  $M$ , the number of such servers in each rack. We model the inter-arrival of short flows as a poisson process (1000 flows per second per server), while all other settings, including flow size distribution and deadlines, are the same as those in the OLDI scenario. The percentage of flows that miss their deadlines under each protocol is depicted in Fig. 14. The presented results include the average deadline miss percentage, and also the minimum and maximum ones for 100 random experiments with different seeds, where we can see LPD-e outperforms all other protocols in all cases again. In particular, when the load is relatively light (i.e., when 10 servers in each rack send short flows), LPD-e achieves no deadline miss no matter whether the deadline is tight or lax. Compared with the other protocols, LPD-e reduces the deadline missing rate by more than 50% in nearly all cases, except when the traffic load is quite heavy (i.e., 40 servers sending short flows in each rack). In that case, LPD-e still reduces deadline missing rate by at least 30%. A similar test is conducted by generating the flow sizes according to a uniform distribution within 2 KB to 200 KB, and generating deadlines according to an exponential distribution with a mean value of 30 ms. 40 and 50 ms, respectively. The results are shown in Fig. 15, where LPD-e still performs the best, no matter the extent of congestion is light, mild or heavy. We also note that, in all simulations, L<sup>2</sup>DCT often achieves comparable results to D<sup>2</sup>TCP, and we guess it is due to the gamma-correction function they both use. Karuna performs better than LPD-e at most situation, but under heavy network load, sometimes LPD-e can perform better than Karuna, we think this is because of the effectiveness of the principle “more load, more differentiation” in the design process of LPD-e.

### 5.4. At scale simulation in spine-leaf data center topology

To demonstrate how LPD works under complex Data Center topology and its performance under real traffic workload, we build the leaf-spine topology in ns-2 simulator and the topology is shown as Fig. 16. In our experiment, there are 144 servers in total and 9 leaf switches connecting with 4 spine switches. The end to end round trip latency across the spine switch (4 hops) is 14.6  $\mu$ s and the end to end trip latency across the leaf switch (2 hops) is 13.3  $\mu$ s. Among the latency, 10  $\mu$ s is cost at the end server and this simulates the computation time of applications like hadoop. The TCP retransmission timeout (RTO) is set as 10 ms and initial cwnd size is 12. Link capacity between spine and leaf nodes are 40 Gbps/x, where x is oversubscription factor and capacity between leaf and end servers is 10 Gbps. K is 64 and the default value for  $s_{\max}$  is 1 MB and  $t_{\max}$  is 30 ms.

**Benchmark workload.** We use two real data center workloads in the experiments: web search workload and data mining workload [6,13]. In web search workload, 70% flows are smaller than 1 MB and more than 50% of flows are between 100 KB and 1 MB. In data mining workload, more than 80% flows are small flows which are smaller than 10 KB and more than 90% bytes are from the 4% large flows which are larger than 1 MB. In the following simulations, flows arrive across a Poisson Process. Flow's source and destination server is random chosen and it obeys Uniform Distribution. Different load means different rate of flows arrival. As today's datacenter network are often oversubscribed [27], we also vary the oversub-

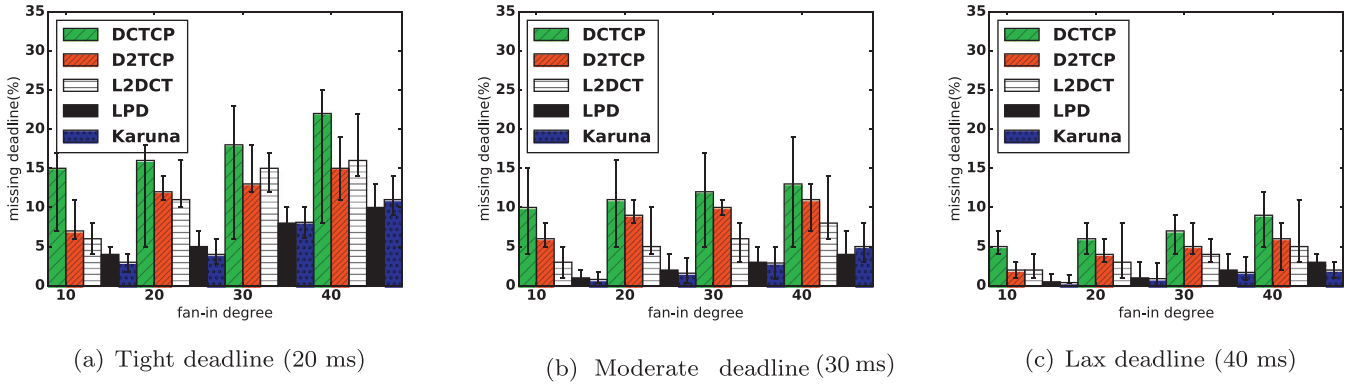


Fig. 13. Deadline missing rate in OLDI applications, under DCTCP, D<sup>2</sup>TCP, L<sup>2</sup>DCT, Karuna and LPD-e.

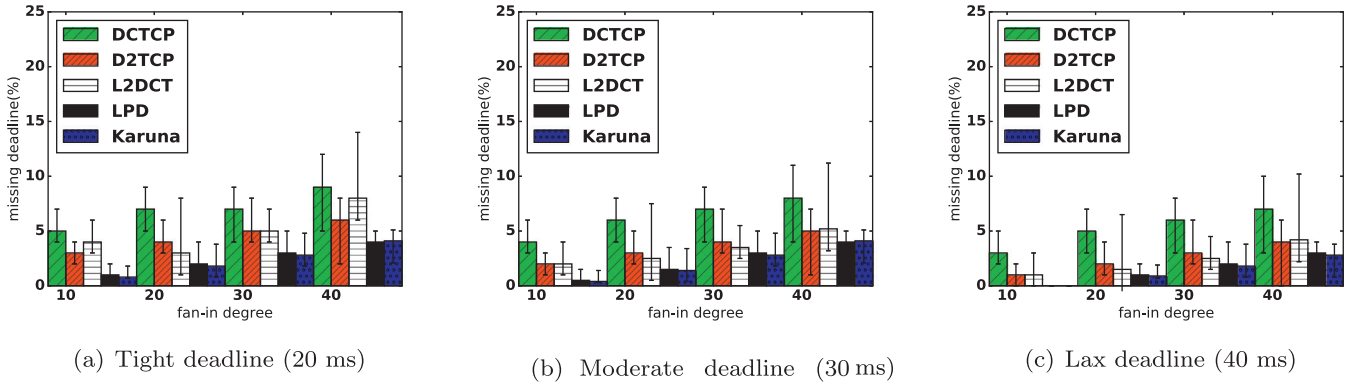


Fig. 14. Deadline missing rate for random flows (Pareto Distribution), in DCTCP, D<sup>2</sup>TCP, L<sup>2</sup>DCT, Karuna and LPD-e.

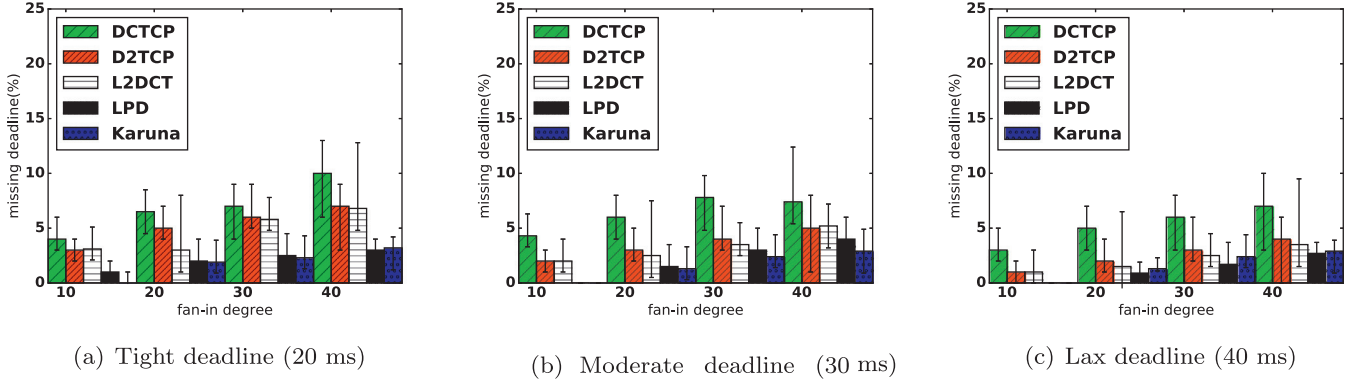


Fig. 15. Deadline missing rate for random deadlines, in DCTCP, D<sup>2</sup>TCP, L<sup>2</sup>DCT, Karuna and LPD-e.

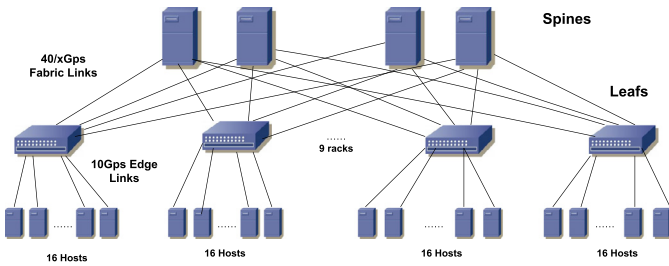


Fig. 16. The spine-leaf data center topology.

scription factor  $x$  of the link that is between the leaf switch and the spine switch to simulate this.

**Performance metrics.** For deadline-constrained flows, we choose the percentage of flows missing deadline as the metrics to

judge the performance of different algorithms. For traffic that are deadline-agnostic, we consider average flow completion time.

#### 5.4.1. Deadline miss comparison

In this section, we test the performance of LPD under delay-sensitive traffic situation. Delay-sensitive business such as web search need to finish transferring the computation results to clients within some milliseconds otherwise it will affect the user experience. In our experiment, deadlines have *uniform* distribution that are between 10 ms to 30 ms. Parameters that are set as the default values. Varying the oversubscription factor  $x$  from 1 to 10, then Fig. 17 shows the result. Note that, we repeat 100 times for each group of experiment and the draw the average, maximum, minimum value of each group in the picture.

From Fig. 17, we can see LPD-e has less than 1% flows and DCTCP, D<sup>2</sup>TCP, L<sup>2</sup>DCT has about 17%, 2.5%, 2.4% flows missing dead-

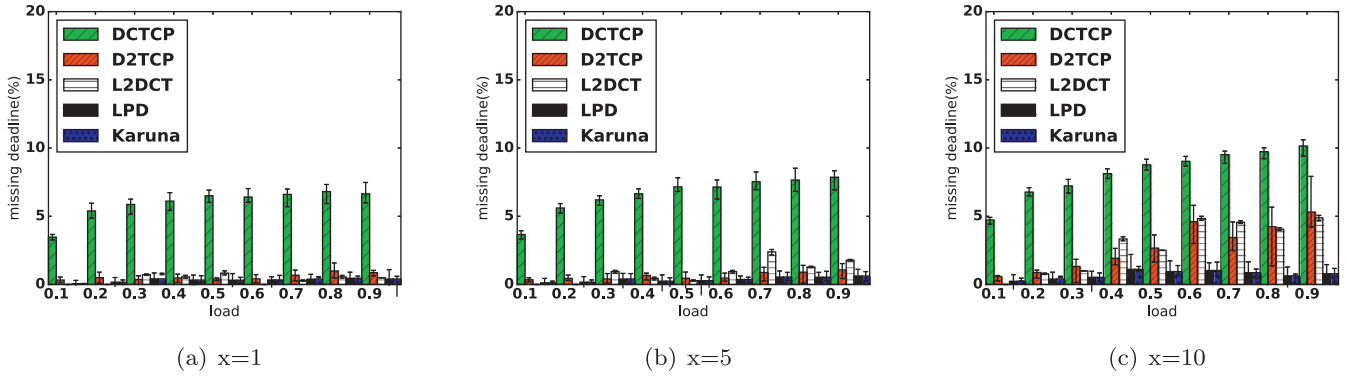


**Fig. 17.** Web search workload of deadline miss comparison between LPD-e, D<sup>2</sup>TCP, L<sup>2</sup>DCT, Karuna and DCTCP. Flows' deadline have uniform distribution between 10 ms and 30 ms.

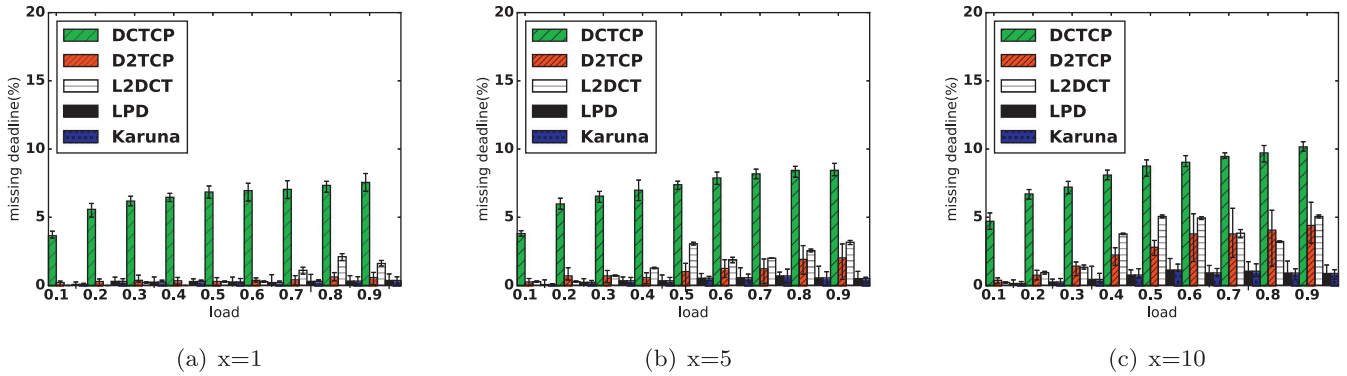


**Fig. 18.** Data mining workload of deadline miss comparison between LPD-e, D<sup>2</sup>TCP, L<sup>2</sup>DCT, Karuna and DCTCP. Flows' deadline have uniform distribution between 10 ms and 30 ms.

line respectively as a whole. Especially, when load and oversubscription factor is small ( $\text{load} = 0.1$ ,  $x = 1$ ), D<sup>2</sup>TCP as well as L<sup>2</sup>DCT has about  $2 \times$  more flows missing than LPD-e, but when oversubscription factor and load are large ( $\text{load} = 0.9$ ,  $x = 10$ ) they have  $5 \times$  more than LPD-e. This is because with the oversubscription factor becomes larger, the load is heavier and LPD's penalty function works better than D<sup>2</sup>TCP and L<sup>2</sup>DCT's gamma correction function, which leads to smaller percentage of flows missing deadline. Karuna, the state-of-art deadline-aware rate control method, also has less than 1% flows missing deadline. However, when the load of network becomes heavier ( $\text{load} = 0.9$ ,  $x = 10$ ), LPD performs better than Karuna. This also denotes that LPD performs better at heavier congestion network condition. The result is similar for data mining workload, which is shown at Fig. 18.

#### 5.4.2. Flow completion comparison

The more load more differentiation principle can also be used for the optimization of other object. Specifically, we consider minimizing the average flow completion time (FCT). We adapt LPD to use  $f = s/s_{\max}$ , and call it LPD-s. In LPD-s,  $s$  can be flow size or remaining flow size. In our experiment we select flow size as the parameter and set  $s_{\max} = 1$  MB. For flows that are larger than 1 MB, we set  $s = 1$  MB. These means for those large flows, LPD's performance is the same as DCTCP. For flows that are smaller than 10 KB, we set  $s = 10$  KB. Assume  $N = 1$  and according to (9), the adjust region of LPD-s is [10 KB, 1 MB]. Setting the oversubscription factor  $x = 5$ , Fig. 19 shows the average flow completion time of web search workload.

From Fig. 19, we can see that overall both LPD-s and L<sup>2</sup>DCT improve much completion time over DCTCP, but has gaps over pFabric. Fig. 19(a) shows that for light traffic load ( $\text{load} < 0.5$ ), LPD-s has an advantage of  $3 \times$  over DCTCP and 10% over L<sup>2</sup>DCT, but when

the traffic load is heavy ( $\text{load} > 0.5$ ), LPD-s performs  $5 \times$  and  $15 \times$  over DCTCP and L<sup>2</sup>DCT. Fig. 19(b) shows the result for flows that are larger than 500 KB. Fig. 19(c) is the result for small flows and note that Fig. 19(c) has different y-axis unit with Fig. 19(b) and (a).

We then change the workload to data mining workload and Fig. 20 shows the result. Specially, from Fig. 20(b), we can see LPD reduces about 40% over DCTCP, while Fig. 19(b) shows for large flows, LPD improves about  $1 \times$ . The reason for this is that in data mining workload, a large percentage of flows are larger than 1 MB and in our experiment  $s_{\max} = 1$  MB, for flows larger than 1 MB will have lowest priority thus can lag the completion time.

To test the performance of LPD under heavy load, we vary the oversubscription factor from 5 to 30. For each group of experiment, we also change the traffic load from 0.1 to 0.9 and then we write down the average FCT and the maximum value and the minimum value of the group. Fig. 21 demonstrates the result. We can observe when the oversubscription factor is small ( $= 5$ ), the average FCT of LPD decreases about 20% comparing to L<sup>2</sup>DCT. However, when the oversubscription factor becomes larger, the advantage is about 50%. This result demonstrates LPD is a good choice specially for heavy load link.

#### 5.5. Small scale test bed evaluation

##### 5.5.1. Simple testbed evaluation

To test the performance of LPD under a real environment, we implement the aforementioned D<sup>2</sup>TCP and LPD-e in Linux kernel 3.2.61, based on DCTCP's code. We also use NetFPGA cards to implement switches, each with four 1 Gbps ethernet ports, so that packets can be forwarded and marked at line speed. Due to our limited hardware, we can only set up a small figure scale topology of two switches and six servers. Main codes of LPD's evaluation of Linux kernel 3.2.61 can be downloaded at [16].

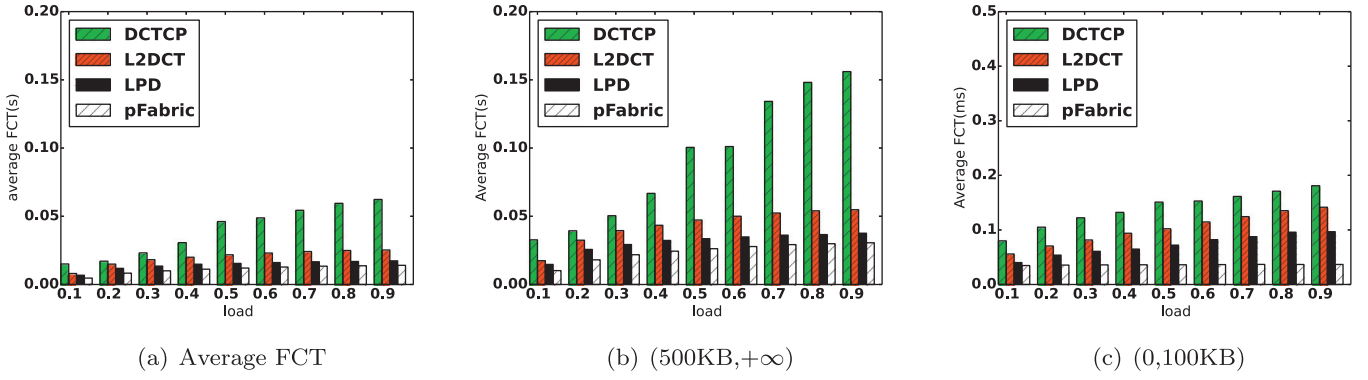


**Fig. 19.** Web search workload, comparison of average flow completion time between LPD-s and DCTCP, L<sup>2</sup>DCT, pFabric. Figure (a) and (b) have different range of y-axis. Figure (c) we show normalized FCT. Note that y-axis in Figure (c) has different unit from Figure (a)'s and (b)'s.

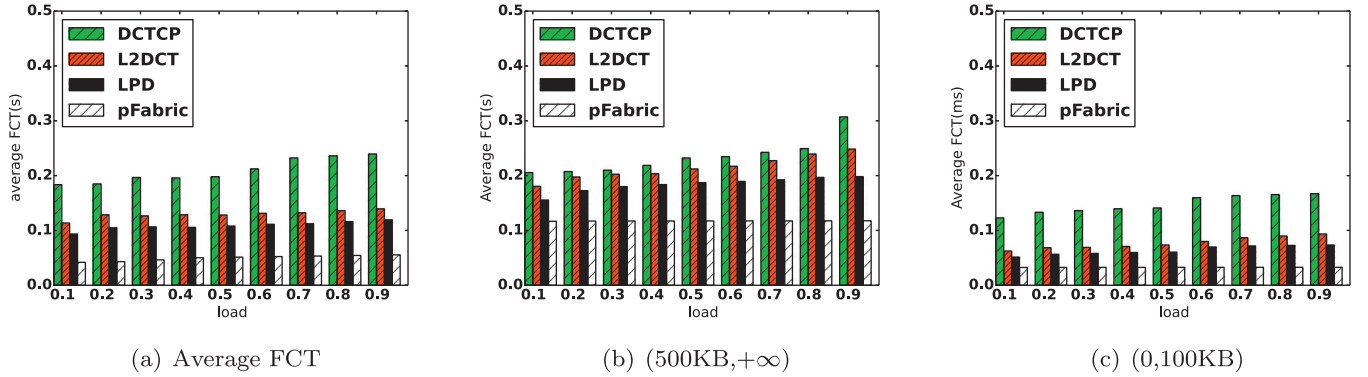


**Fig. 20.** Data Mining workload, overall normalized average FCT for LPD-s, DCTCP, L<sup>2</sup>DCT and pFabric. Note that y-axis in Figure (c) has different unit from Figure (a)'s and (b)'s.

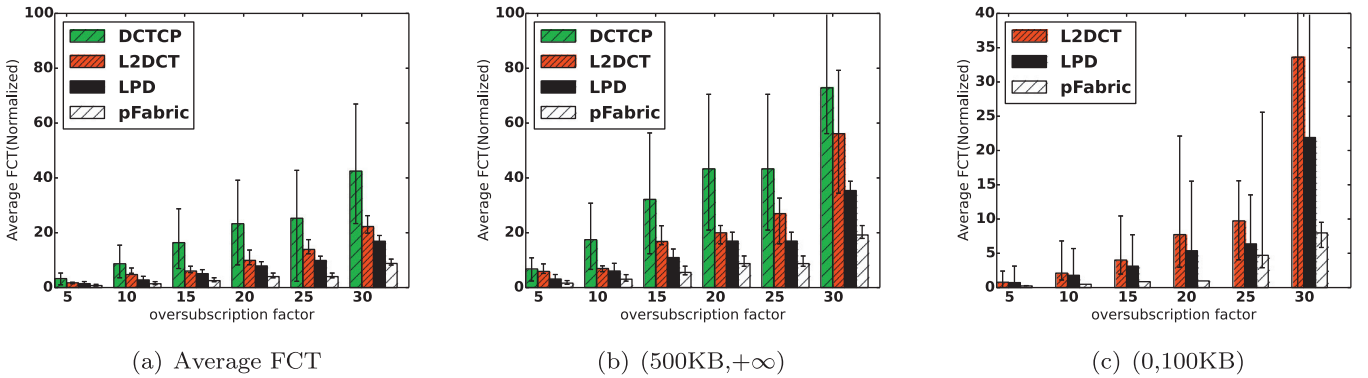


**Fig. 21.** Web Search workload, overall normalized average FCT for LPD-s, DCTCP, L<sup>2</sup>DCT and pFabric when we varying the oversubscription factor from 5 to 30. Note that DCTCP does not appear in Figure (c) as its value is out of the plot range.

We first verify whether LPD-e can differentiate flows with different deadlines in a real environment, using a setting similar to the motivating example in Section 3.1. The flow sizes are 10, 30 and 50 MB, and the corresponding deadlines are 300, 600 and 900 ms. In Fig. 22, we plot the bandwidth evolution of each flow, and also the queue length on the switch. Similar to the simulation result, flows with different deadlines are clearly differentiated, and the queue length stays around the marking threshold of 37.5 KB (equivalent to  $k = 25$  packets of 1.5 KB).

### 5.5.2. Small data center evaluation

We build a small tree-based test environment to test the performance of LPD. Fig. 23 shows the topology. In this topology, two linux servers run Ubuntu 14.04 along with dummynet [28,29] to construct a two-way bottleneck link. We patch dummynet to

mark packets. In our experiment, all interfaces of the topology is 1 Gbps.

LPD's correction function follows the principle of more load, more differentiation and it works well even under heavy traffic load, thus, more flows can meet deadline. In our experiment, each server  $S_i$  first builds up a connection with the random server  $S_j (i \neq j)$ . Then  $S_i$  continuous figure maximum and minimum value of deadline miss percentage and Fig. 24(a) shows the result. We can see LPD performs better than D<sup>2</sup>TCP and DCTCP. Comparing with DCTCP and D<sup>2</sup>TCP, the percentage of missing deadline reduces about 50% and 30% respect. To see the performance of LPD under OLDI application, we let  $S_1 - S_6$  continuous send small flows with random deadline ( $\leq 10$ ms) to root R, Fig. 24(b) shows the result. We observe that LPD performs about 30% better than DCTCP and 20% than D<sup>2</sup>TCP.

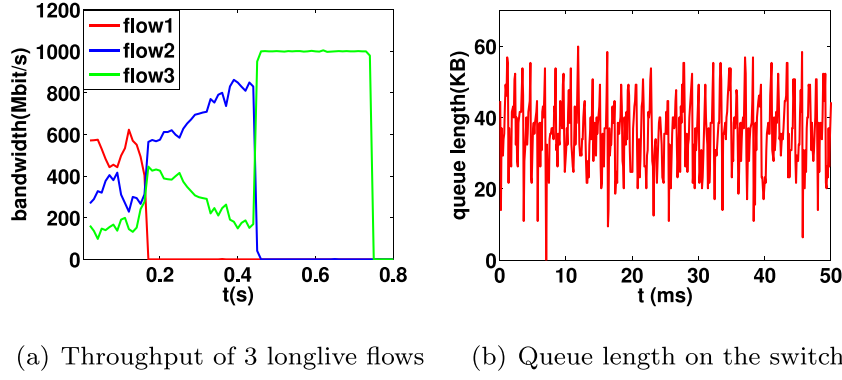


Fig. 22. Flow differentiation.

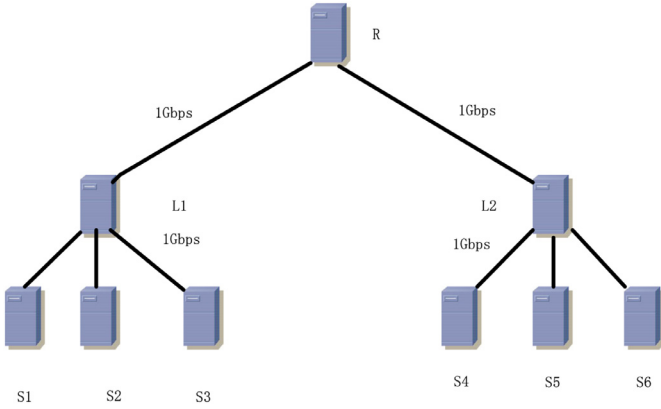


Fig. 23. The small scale data center tested.

To test the performance of LPD on reducing flow completion time. We let each server  $S_i$  send flows to the random destination. The process lasts 2 min and we repeat the process 20 times, then Fig. 25(a) shows the result. We can see that comparing with DCTCP and  $L^2DCT$ , LPD reduces about 20% and 10% on average flow completion time. Then we test LPD's performance on the scenery of Partitions/Aggregate. First we let the root node server R sends query to each end node. After the end node receives the query, it will send response with flow size K, varying K from 50 KB to 200 KB, then Fig. 25(b) shows the result. We can see that comparing with DCTCP and  $L^2DCT$ , LPD's cuts about 20% and 10% on completion time. The reason for LPD's advantage is that LPD's penalty

still works well even heavy load which is caused by the concurrent of flow sending by end nodes.

## 6. Discussion

As we have noted, our more load, more differentiation is quite general, in the sense that different functions of the load and imminence factors, as well as different kinds of imminence factors, can be used. We will use several examples to demonstrate its generality.

Our example still use the forementioned Eq. (2)

$$w = \begin{cases} w + (1 - f) & \text{without congestion;} \\ w \times (1 - f) & \text{with congestion} \end{cases}$$

for congestion control.

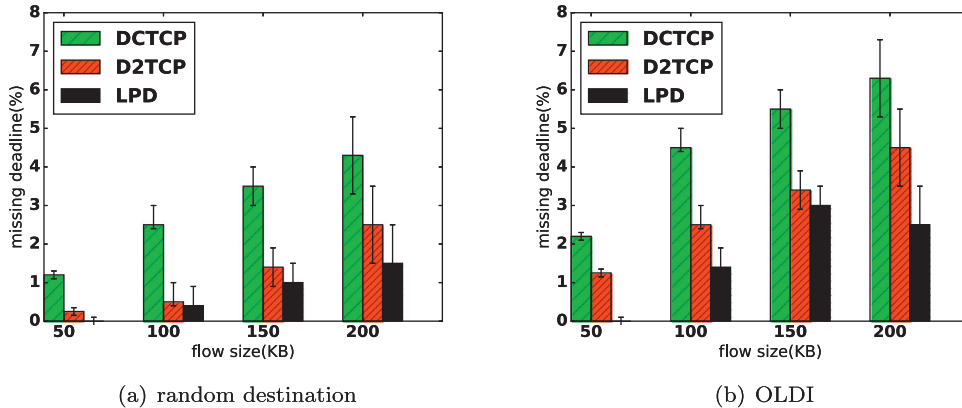
### 6.1. Different penalty function

$f = \alpha/d$  where  $d$  is the same imminence factor used by  $D^2TCP$ . It's just LPD with another imminence factor, and we call the resulting congestion control algorithm LPD-d. It has already been tested against our motivating example. The second example is a more complex function, define as:

$$f = \begin{cases} (2 - d)^\alpha & 0 < d < 1 \\ \alpha & d = 1 \\ \alpha * \frac{2-d}{2d-\alpha} & d > 1 \end{cases}$$

It can be proved that  $\frac{\partial(1-f)}{\partial d} > 0$ , while  $\frac{\partial^2(1-f)}{\partial \alpha \partial d} > 0$ , so it respects the more load more differentiation principle, and we call it MLD.

Fig. 26(a) depicts the results for LPD-e, MLD and LPD-d, against uniformly distributed deadlines. Fig. 26(b) shows the results of

Fig. 24. Deadline miss comparison between LPD and DCTCP,  $D^2TCP$ .

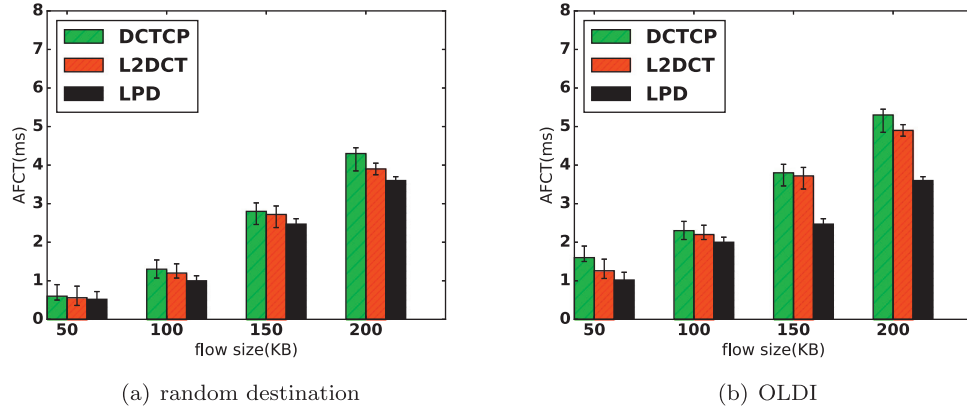


Fig. 25. Flow completion time comparison between LPD and DCTCP, L²DCT.

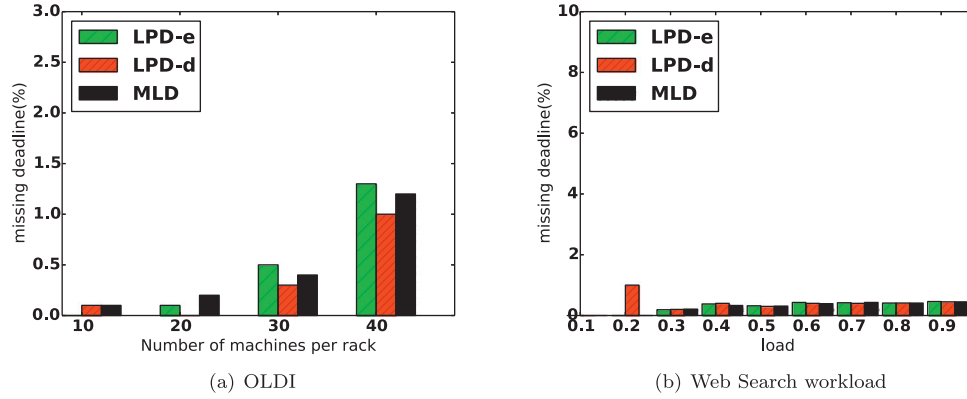


Fig. 26. Further extensions of the design principle.

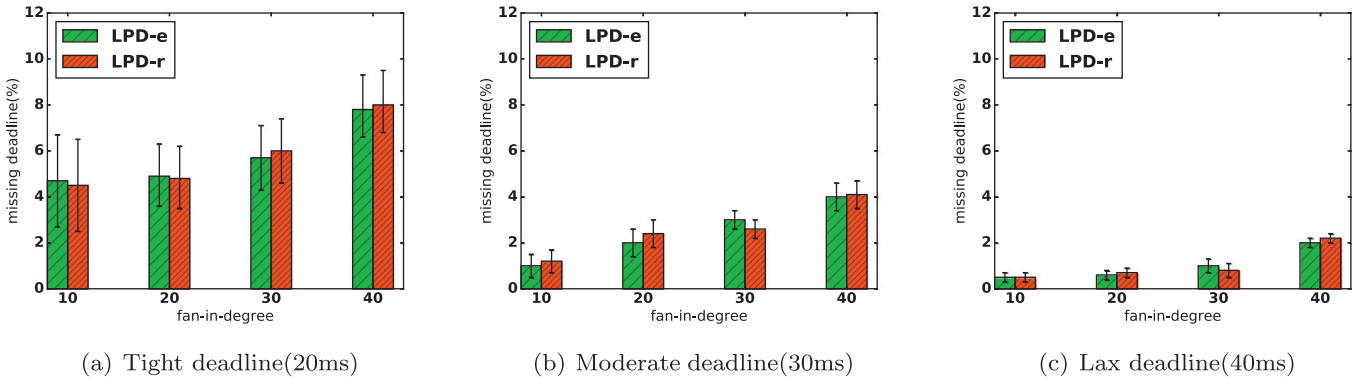


Fig. 27. Deadline missing rate in OLDI applications, under LPD-e and LPD-r.

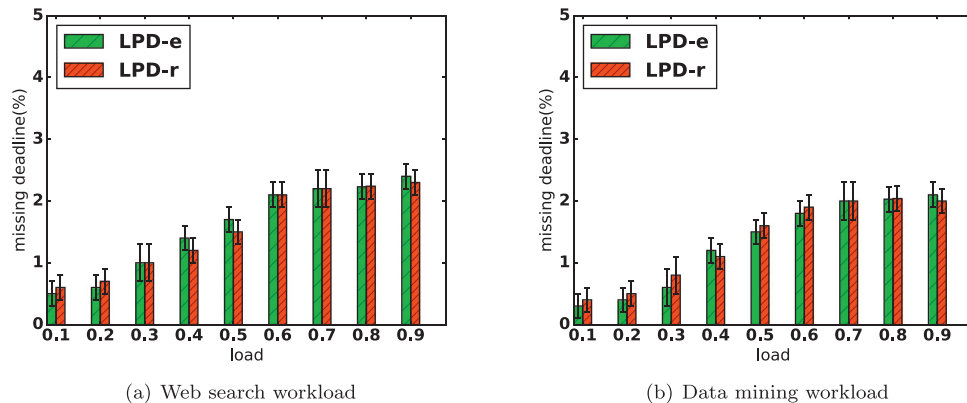


Fig. 28. Web search workload and data mining of deadline miss comparison between LPD-e, LPD-r. Flows' deadline have uniform distribution between 10 ms and 30 ms. The oversubscription factor is 10.

web search workload. It suggests that, different forms of control algorithms do not differ in a great deal in this scenario, while whether there exists any optimal algorithm remains to be investigated.

## 6.2. Use remaining time

In Section 4, we propose LPD-e that uses the duration between a flow's start time and its deadline in the imminence factor. Since, the ideal scheduling policy to reduce deadline missing rate is earliest deadline first. We investigate another algorithm that uses the duration between current time and flow's deadline in the imminence factor. We call this algorithm LPD-r.

We evaluate LPD-r under the same simulation settings used earlier, and compare it with LPD-e. Fig. 27 depicts the flow deadline missing rate under random flows in a simple topology, while Fig. 28 depicts the missing rate under web search and data mining workload in a spine-leaf topology. From these illustrations, we can see LPD-r and LPD-e have comparable performance. This confirms again the generality of our LPD principle, i.e., different functions of the load and imminence factors, as well as different kinds of imminence factors, can be used.

## 7. Conclusion

In this paper, we propose a simple principle to meet the requirement that flows in data centers should be completed before certain deadlines. To prove its usefulness, we further design a rate control scheme called load proportional differentiation, which, under a heavy congestion, differentiates flows in a greater manner when traffic load is heavier. We use different topologies and traffic loads to demonstrate the performance of LPD, which nearly always outperforms other latest rate control protocols such as DCTCP, D<sup>2</sup>TCP, L<sup>2</sup>DCT and Karuna, especially under a heavy traffic load.

Although LPD is specifically designed to meet flow deadlines, we believe the “more load, more differentiation” principle is quite general, and can be applied to other scenarios where flows should be differentiated and given different priorities, with very simple adaption. How to carry out this adaption, as well as more rigorous analysis on LPD, such as its stability and convergence properties, requires further studies.

## Acknowledgments

This work is supported by the National High Technology Research and Development Program of China (863 Program) No. 2015AA016105, the National Natural Science Foundation of China (Grant No. 61402253) and China Scholarship Council (Grant No. 201606210204).

## References

[1] M. Chowdhury, Y. Zhong, I. Stoica, Efficient coflow scheduling with varies, in: ACM SIGCOMM Computer Communication Review, vol. 44, ACM, 2014, pp. 443–454.

[2] L. Chen, K. Chen, W. Bai, M. Alizadeh, Scheduling mix-flows in commodity datacenters with karuna, in: Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference, ACM, 2016, pp. 174–187.

[3] H. Zhang, X. Shi, X. Yin, Z. Wang, Y. Guo, Fdrc-flow duration time based rate control in data center networks, in: Quality of Service (IWQoS), 2016 IEEE/ACM 24th International Symposium on, IEEE, 2016, pp. 1–10.

[4] M. Chowdhury, I. Stoica, Coflow: a networking abstraction for cluster applications, in: Proceedings of the 11th ACM Workshop on Hot Topics in Networks, ACM, 2012, pp. 31–36.

[5] M. Chowdhury, I. Stoica, Efficient coflow scheduling without prior knowledge, in: ACM SIGCOMM Computer Communication Review, vol. 45, ACM, 2015, pp. 393–406.

[6] M. Alizadeh, A. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center tcp (dctcp), in: SIGCOMM '10, 2010, pp. 63–74.

[7] D. Meisner, C.M. Sadler, L.A. Barroso, W.-D. Weber, T.F. Wenisch, Power management of online data-intensive services, in: ISCA '11, 2011, pp. 319–330.

[8] B. Vamanan, J. Hasan, T. Vijaykumar, Deadline-aware datacenter tcp (d2tcp), in: SIGCOMM '12, 2012, pp. 115–126.

[9] Latency, Latency is everywhere and it costs you sales - how to crush it, 2009, (<http://highscalability.com/latency-everywhere-and-it-costs-you-sales-how-crush-it>).

[10] C. Wilson, H. Ballani, T. Karagiannis, A. Rowtron, Better never than late: meeting deadlines in datacenter networks, in: SIGCOMM '11, 2011, pp. 50–61.

[11] C.-Y. Hong, M. Caesar, P. Godfrey, Finishing flows quickly with preemptive scheduling, ACM SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 127–138.

[12] A. Munir, I. Qazi, Z. Uzmi, A. Mushtaq, S. Ismail, M. Iqbal, B. Khan, Minimizing flow completion times in data centers, in: INFOCOM, 2013 Proceedings IEEE, 2013, pp. 2157–2165.

[13] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, pfabric: Minimal near-optimal datacenter transport, in: SIGCOMM '13, 2013, pp. 435–446.

[14] The network simulator — ns-2, (<http://www.isi.edu/nsnam/ns/>).

[15] Simulation code of lpd, (<https://github.com/zhangan1990/LPD-sim/>).

[16] Linux kernel code of lpd, (<https://github.com/zhangan1990/LPD/>).

[17] A. Phanishayee, E. Krevat, V. Vasudevan, D.G. Andersen, G.R. Ganger, G.A. Gibson, S. Seshan, Measurement and analysis of tcp throughput collapse in cluster-based storage systems, in: FAST'08, 2008, pp. 12:1–12:14.

[18] Y. Chen, R. Griffith, J. Liu, R.H. Katz, A.D. Joseph, Understanding tcp incast throughput collapse in datacenter networks, in: WREN '09, 2009, pp. 73–82.

[19] C. Poynton, Digital Video and HDTV Algorithms and Interfaces, 1, Morgan Kaufmann Publishers Inc., 2003.

[20] R. Mittal, N. Dukkkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats, et al., Timely: Rtt-based congestion control for the datacenter, in: ACM SIGCOMM Computer Communication Review, vol. 45, ACM, 2015, pp. 537–550.

[21] D. Zats, T. Das, P. Mohan, D. Borthakur, R. Katz, Detail: Reducing the flow completion time tail in datacenter networks, in: SIGCOMM '12, 2012, pp. 139–150.

[22] 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, in: Proceedings of the 2014 ACM Conference on SIGCOMM, ACM, 2014, pp. 491–502.

[23] H. Zhang, X. Shi, X. Yin, F. Ren, Z. Wang, More load, more differentiation: a design principle for deadline-aware congestion control, in: 2015 IEEE Conference on Computer Communications (INFOCOM), IEEE, 2015, pp. 127–135.

[24] M. Alizadeh, A. Javanmard, B. Prabhakar, Analysis of dctcp: stability, convergence, and fairness, in: Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, ACM, 2011, pp. 73–84.

[25] V. Misra, W.-B. Gong, D. Towsley, Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red, in: ACM SIGCOMM Computer Communication Review, vol. 30, ACM, 2000, pp. 151–160.

[26] H. Scalability, Latency is everywhere and it costs you sales - how to crush it, (<http://simula.stanford.edu/~alizade/Site/DCTCP.html>).

[27] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, P. Sharma, Application-driven bandwidth guarantees in datacenters, in: Proceedings of the 2014 ACM conference on SIGCOMM, ACM, 2014, pp. 467–478.

[28] The dummynet project. [online]. available., (<http://info.iet.unipi.it/luigi/dummynet/>).

[29] Y. Cao, M. Xu, X. Fu, E. Dong, Explicit multipath congestion control for data center networks, in: Proceedings of the ninth ACM conference on Emerging networking experiments and technologies, ACM, 2013, pp. 73–84.



**Han Zhang** received the B.S. degree in Computer Science and Technology from Jilin University, China in 2013. He is now pursuing his Ph.D. in Tsinghua University. His research concerns Data Center network, cloud computing and Transport protocol.



**Xingang Shi** received the B.S. degree from Tsinghua University and the PhD degree from The Chinese University of Hong Kong. He is now working in the Institute for Network Sciences and Cyberspace at Tsinghua University. His research interests include network measurement and routing protocols.



**Yingya Guo** received the B.S degree in computer science from Xiamen University, China, in 2013. She is currently a PhD candidate at the Department of Computer Science and Technology, Tsinghua University. Her research interests include traffic engineering, routing optimization and Software Defined Networking.



**Zhiliang Wang** received the B.E., M.E. and Ph.D. degrees in computer science from Tsinghua University, China in 2001, 2003 and 2006 respectively. Currently he is an Associate Professor in the Institute for Network Sciences and Cyberspace at Tsinghua University. His research interests include formal methods and protocol testing, next generation Internet, network measurement.



**Xia Yin** received the B.E., M.E. and Ph.D. degrees in computer science from Tsinghua University in 1995, 1997 and 2000 respectively. She is a Full professor in Department of Computer Science and Technology at Tsinghua University. Her research interests include future Internet architecture, formal method, protocol testing and large-scale Internet routing.