More Load, More Differentiation - a Design Principle for Deadline-Aware Congestion Control

Han Zhang *‡, Xingang Shi†‡, Xia Yin*‡,Fengyuan Ren*‡ and Zhiliang Wang†‡
*Department of Computer Science and Technology, Tsinghua University
†Institute for Network Sciences and Cyberspace, Tsinghua University
‡Tsinghua National Laboratory for Information Science and Technology (TNLIST)
Email:{zhanghan}@csnet1.cs.tsinghua.edu.cn, {shixg, wzl}@cernet.edu.cn, {yxia, renfy}@tsinghua.edu.cn

***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. Our results show that LPD nearly always outperforms D^{2}TCP, a latest deadline-aware rate control scheme, and often reduces the number of flows missing their deadlines by more than 50%. We also give some other applications of this principle, for example, in reducing flow completion time.**

I. Introduction

Nowadays, more and more services and applications such as web search and social networking are hosted in data centers. 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 hardware. In data center networks (DCNs), applications typically require low latency for short flows, high burst tolerance, and high throughput for large flows [1]. 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) [6], [11] applications are popular in data centers. Such applications must serve users in a timely manner since even one more millisecond of latency may considerably affect the revenue [5], and they typically need to query and aggregate results from a large number of nodes. To meet hard and strict response deadlines, applications may send out incomplete responses 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 [1] do not work well for these applications, and deadline has been explicitly taken into consideration in modern data center flow rate control [11], [12] or flow scheduling [4] algorithms.

In this paper, we focus on the design of deadline-aware flow rate control mechanisms for DCNs. We start our work based on D^{2}TCP [11], which requires no changes to switch hardware and can coexist with legacy TCP (i.e., characteristics that are just missed by D^3 [12] and PDQ [4]). By careful investigations on D^{2}TCP's performance under different traffic load scenarios, we notice that under high traffic load, it degenerates to DCTCP and becomes almost 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 design 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. We have implemented LPD in both ns2 and Linux kernel 3.2.61. 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 [1], D^{2}TCP [11], and L2DCT [7]. LPD nearly always outperforms them in enabling more flows to meet their deadlines. In typical scenarios, it can reduce the number of flows missing their deadlines by more than 25%, compared with D^{2}TCP, the best existing deadline-aware schemes. 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.

The key contributions of this paper are:

978-1-4799-8381-0/15/$31.00 ©2015 IEEE

- 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 design, analyze and evaluate the LPD algorithm as an application of our principle, and compare it with latest deadline-aware rate control mechanisms, to demonstrate its effectiveness.
- 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 II. Then in Section III-A, we define the more load, more differentiation principle, which is motivated by careful investigations on D^{2}TCP. The LPD algorithm is presented and analyzed in Section IV, and is evaluated in Section V both by simulation and in real testbed. Some further discussions are given in Section VI, and finally, Section VII concludes the paper.

II. 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 [3], [8]. 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 [6] 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 [1] 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 α, 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 show 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 [11].

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 [4], [12], or adjusted implicitly by varying the congestion window size on end points [11].

D^3 [12] 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 [11]. On the other hand, D^3 requires switch hardware change, and cannot coexist with TCP, due to its request-reply mechanism for bandwidth allocation.

D^{2}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 α and the deadline imminence factor d are used in the multiplicative decrease of D^{2}TCP's congestion window as $w = w \times (1 - \alpha^d/2)$, where α^d is the well-known gamma-correction function [9], and is used as a penality function here. In this way, D^{2}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 [4] uses preemptive flow scheduling at switches to assist bandwidth allocation. Unlike D^3, PDQ works in a distributed fashion, and its preemptive scheduling can deal with the race conditions that threaten D^3. PDQ can emulate different scheduling policies like Earliest Deadline First (EDF) or Shortest Job First (SJF). However, policy like SJF was not purposely designed for scheduling flows with deadlines, and only flow completion time is used as the evaluation metric in [4], while their performance in completing flows before hard deadlines has not been investigated throughly. Besides, PDQ also requires switch hardware change and cannot coexist with legacy TCP.

Another class of related algorithms are specifically optimized to reduce flow completion time (FCT) [2], [7] or completion time "tail" [13]. L2DCT [7] adjusts both the additive increase and multiplicative decrease of an endpoint's congestion window size, where the gamma-correction function is also used, similar to that in D^{2}TCP but with a different imminence factor. On the other hand, pFabric [2] schedules flows according to their remaining sizes. The proposed distributed algorithm approximates the centralized Shortest Remaining Processing Time (SRPT) police, 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. Another work [13] 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.

For our object to let more flows finish before their deadlines, we praise the idea that rate control and flow scheduling should be decoupled [2], and focus on the first part in this paper. Our work is in particular relevant to D^{2}TCP and

L2DCT, both of which adjust their congestion window sizes to implicitly regulate the bandwidth occupied by each flow.

III. More Load, More Differentiation

Our work borrows ideas heavily from earlier rate control schemes in DCNs, such as DCTCP, D^{2}TCP and L2DCT. In particular, we have exactly the same object as D^{2}TCP, so some further analysis of D^{2}TCP is worth it. Through combining a deadline-aware penalty function with DCTCP, D^{2}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 III-A, we start with a simple scenario where D^{2}TCP does not perform as well as expected. Then in Section III-B, we analyze the root cause of this "bad case". We propose the deadline-aware rate control principle in Section III-C, where we also derive the sufficient and necessary conditions for satisfying it.

A. 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 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 I, 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^{2}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^{2}TCP is only as good as DCTCP, if perceived by applications.

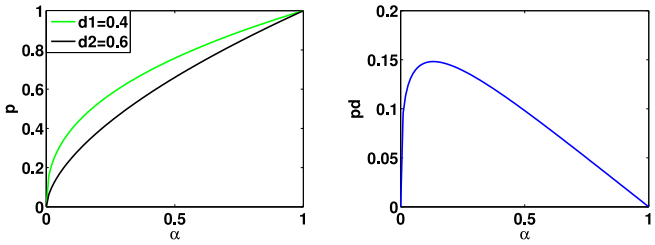
Fig. 1 provides more details on the internal status of D^{2}TCP. As we know, D^{2}TCP uses $w = w \times (1 - \alpha^d/2)$ to adjust its congestion window size w, where α is the estimated extent of congestion, and d is an deadline imminence factor. In Fig. 1(a), the bandwidth occupied by each flow is 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 α, the extent of congestion, in Fig. 1(d), which shows flow 1 and flow 4 achieve a comparable estimation on α.

B. Analysis of the Gamma-Correction Function

Now we take a closer look at D^{2}TCP's rate control mechanism, trying to dig out why it performs badly in the scenario above. The central idea of D^{2}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 α 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 α increases, and then decreases after α 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^{2}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.

TABLE I: Untransmitted data after deadline

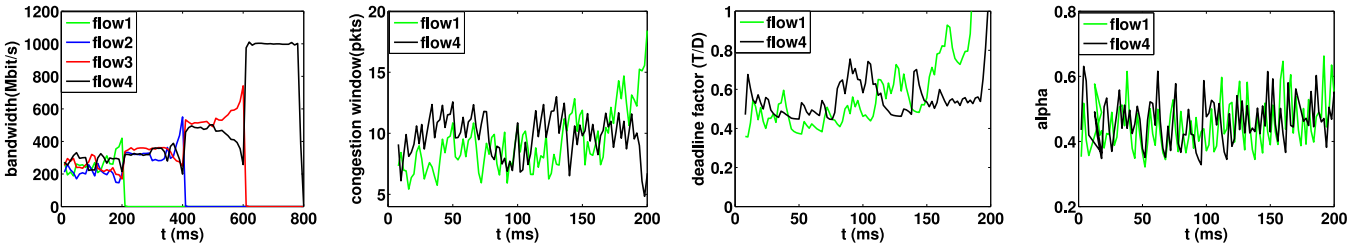
	D^{2}TCP	DCTCP
flow 1 (8 MB, 200 ms)	1.53 MB	2.07 MB
flow 2 (16 MB, 400 ms)	1.79 MB	2.14 MB
flow 3 (32 MB, 600 ms)	3.64 MB	4.08 MB
flow 4 (40 MB, 800 ms)	0 MB	0 MB



(a) $p = \alpha^d$ (b) $pd = p(\alpha, d_1) - p(\alpha, d_2)$

Fig. 2: D^{2}TCP uses the gamma-correction function to decrease its congestion window

Now we can explain why D^{2}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 α 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 $d_4 = 0.4$, 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 flow 1'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 α increases.



(a) Individual flow bandwidth (b) Congestion window size (c) Deadline imminence factor (d) Extent of congestion

Fig. 1: D^{2}TCP's performance under four concurrent flows

C. 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 D^3 or D^{2}TCP. 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 meet 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 α ($0 < \alpha < 1$), where greater d and α 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 α and d.

Theorem *1:* The sufficient and necessary condition for r to follow the above principle is $\frac{\partial r}{\partial d} > 0$, and $\frac{\partial^2 r}{\partial d \partial \alpha} >= 0$ (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

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

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 equation (1), holds true if and only if $\frac{\partial^2 r}{\partial \alpha \partial d} \geq 0$ but does not constantly equal 0. ■

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 α 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.

IV. Load Proportional Differentiation

A. Basic Algorithm

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

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

where $f = \alpha / d$ is the penalty applied to the congestion window size according to traffic load α and flow deadline imminence d, and is capable to differentiate between flows. The extent of congestion (i.e., the traffic load factor α), is estimated using the same mechanisms as that in DCTCP and

D^{2}TCP [1], [11]. Since f is proportional to α, 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.

B. Properties and Analysis

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. ■

It is also quite easy to understand why LPD respects PI and PD from a geometric view. Since f is proportional to α, a scatter plot of (α, 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 α.

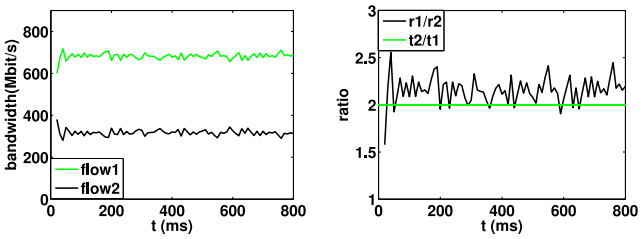
proposition *2:* Suppose LPD-t can lead to a longterm steady state, that is the congestion window of each flow steadily (and synchronously) cycles within a small fixed interval, implying a steady flow rate and congestion extent. If $t \ll t_{max}$, where t and t_{max} are parameters used by LPD-t in equation (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 equation (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 $r_1/r_2 \approx t_2/t_1$. ■



(a) Individual flow bandwidth (b) Bandwidth ratio

Fig. 3: Two flows under LPD-t

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 $t_2/t_1 = 2$.

C. Considering Flow Size

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

$$f = \alpha \times \frac{t}{t_{max}} \times \frac{s}{s_{max}} \quad (4)$$

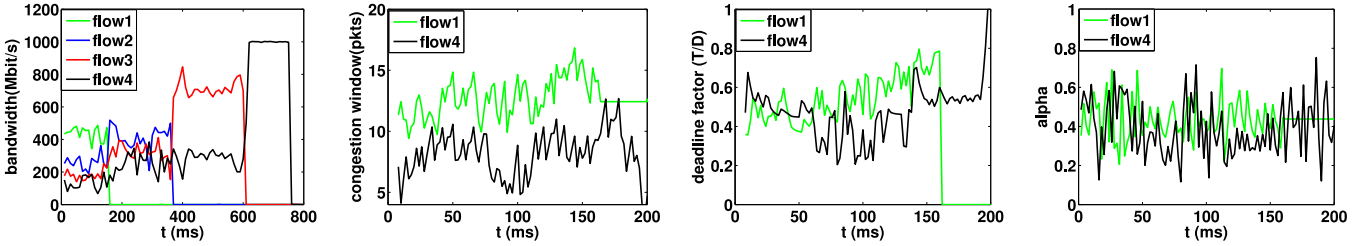
where s_{max} is an upper bound for truncating a large s.

In this way, we hope when there is not enough bandwidth to meet all deadlines, as least those small flows may have a greater chance to meet their deadlines. Equation (4) 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 VI. 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.

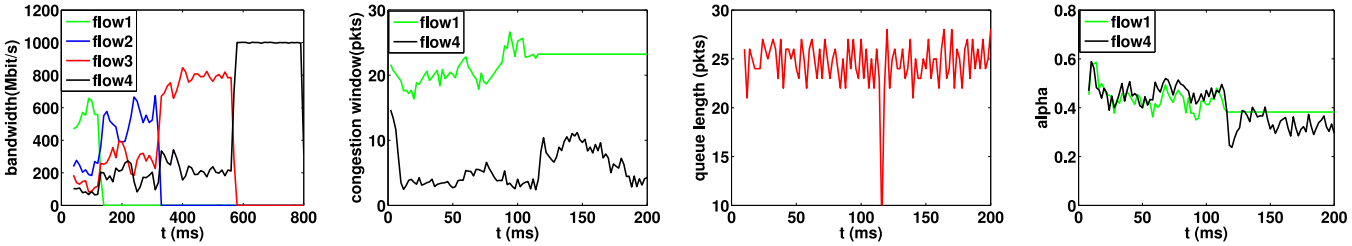
V. Evaluation

In this section, we evaluate our design principle and the LPD algorithm by ns-2 simulation. We implement LPD (including LPD-t, LPD-e) based on the source code of DCTCP released at their website [10], and port the ns-3 code of D^{2}TCP provided by its author to ns-2. In addition, we also implement L^{2}DCT based on its pseudo algorithm.



(a) Individual flow bandwidth (b) Congestion window size (c) Deadline imminence factor (d) Extent of congestion

Fig. 4: LPD's performance under 4 concurrent flows, using the same imminence factor as D^{2}TCP



(a) Individual flow bandwidth (b) Congestion window size (c) Queue length (d) Extent of congestion

Fig. 5: LPD-t's performance under 4 concurrent flows

A. Revisiting the Motivating Example

We first apply LDP to our motivating example presented in Section III-A. 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^{2}TCP. To help a better understanding of why LPD outperforms D^{2}TCP and DCTCP, we present more details on LPD's internal status in Fig. 4. The bandwidth of each individual flow, as well as the congestion window size w, the imminence factor d, and the traffic load α 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. 4(a). Ultimately, less deadline miss is achieved by LPD than by D^{2}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. 5 depicts its internal status. Compared with that of LPD and D^{2}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. 5(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.

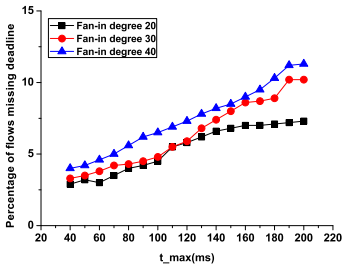


Fig. 6: The effect of varying t_{max}

B. 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 equation (3) that, if t_{max} is larger, the penalty function f will be smaller, and hence 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 to a tight value $t = 20$ ms, and t_{max} is varied from 20 ms to 200 ms. Fig. 6 depicts the number of flows missing 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. 6, we can see that when $t_{max} < 80ms$, 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^{2}TCP (the corresponding values under D^{2}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 140 ms (i.e., seven times of the deadlines), LPD-t's performance get greatly harmed, and becomes worse than D^{2}TCP.

C. At Scale Simulation In a 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. 7. 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 [1] [11] [7], we set the packet marking threshold $k = 20$ at the switches, and set the round trip propagation delay as 100 us. The TCP retransmission timeout (RTO) is set as 10 ms. 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 varis from 20 to 40, representing different levels of traffic load. These parameters are chosen to match those used in D^{2}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 [8] if traditional TCP is used, since a large number of concurrent small flows sharing a bottleneck link often induce heavy congestion.

Fig. 8 shows the percentage of flows missing their deadlines under each setting, where DCTCP, D^{2}TCP, L2DCT 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 L2DCT is also a TCP-like congestion window based rate control scheme, using gamma-correction in a similar way to D^{2}TCP, but with a different imminence factor. The results demonstrate the superiority of LPD-e, which in all experiments causes the smallest number of

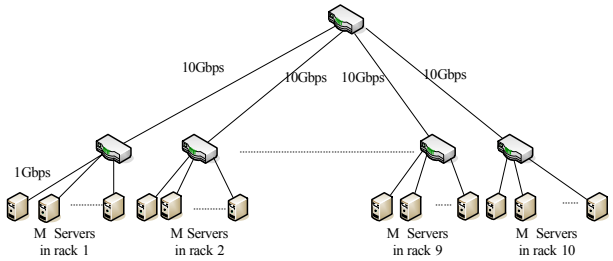
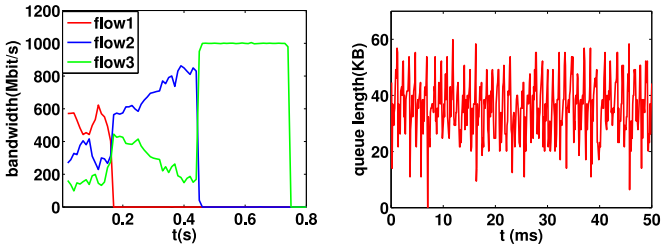


Fig. 7: The Data Center toplogy



(a) Throughput of 3 longlive flows (b) Queue length on the switch

Fig. 9: Flow Differentiation

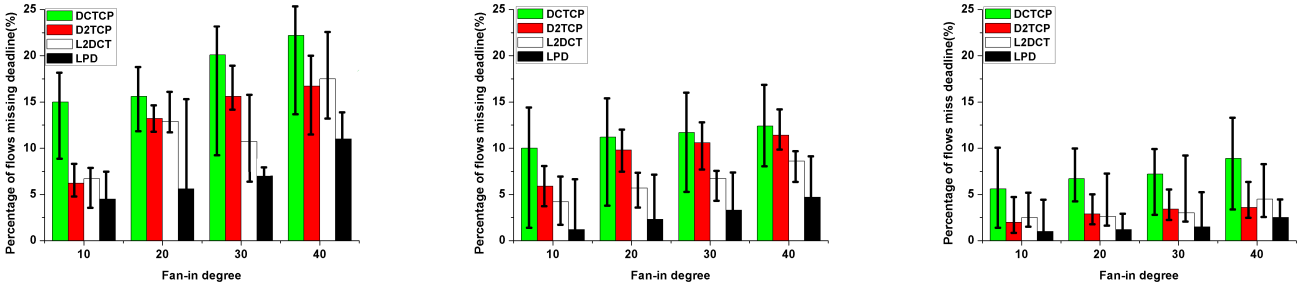
(sometimes zero) deadline misses due to its greater capability to differentiate flows under heavy load. Although D^{2}TCP (as well as L2DCT) improves on DCTCP, it has a clear advantage over DCTCP only for lax deadlines (Fig. 8(c)), but not for tight deadlines (Fig. 8(a) and (b)). This effect was not shown in [11] since their experiments used mixed deadlines.

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. 10. 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. 11, 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, L2DCT often achieves comparable results to D^{2}TCP, and we guess it is due to the gamma-correction function they both use.

D. Small Scale Test Bed Evaluation

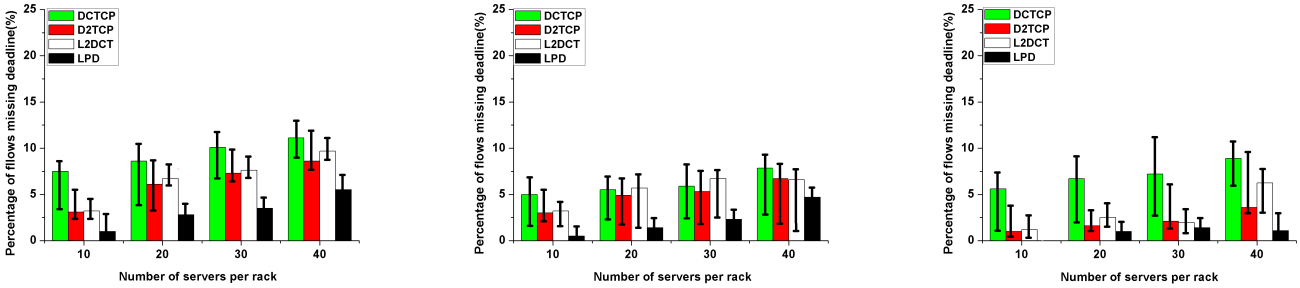
To test the performance of LPD under a real environment, we implement the aforementioned D^{2}TCP and LPD-e in Linux kernel 3.2.61, based on DCTCP's code. We also use NetFPGA cards to implement switches, each with four 1 Gbps ethernet ports, so that packets can be forwarded and marked at line speed.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

We first verify whether LPD-e can differentiate flows with different deadlines in a real environment, using a setting



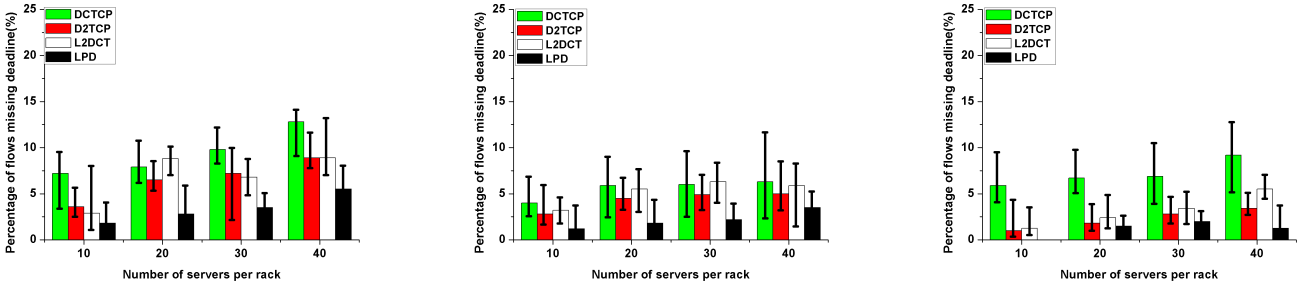
(a) Tight deadline (20 ms) (b) Moderate deadline (30 ms) (c) Lax deadline (40 ms)

Fig. 8: Deadline missing rate in OLDI applications, under DCTCP, D^{2}TCP, L^{2}DCT and LPD-e



(a) Tight deadline (20 ms) (b) Moderate deadline (30 ms) (c) Lax deadline (40 ms)

Fig. 10: Deadline missing rate for random flows (Pareto Distribution), in DCTCP, D^{2}TCP, L^{2}DCT and LPD-e



(a) Tight deadline (20 ms) (b) Moderate deadline (30 ms) (c) Lax deadline (40 ms)

Fig. 11: Deadline missing rate for random deadlines, in DCTCP, D^{2}TCP, L^{2}DCT and LPD-e

similar to the motivating example in Section III-A. The flow sizes are 10, 30 and 50 MB, and the corresponding deadlines are 300, 600 and 900 ms. In Fig. 9, 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).

We then test LPD-e with an OLDI application scenario, where one server receives short flows with deadlines from all the other servers, with long-lived flows as background flows. The setting here is similar to those in the simulated OLDI application test, except that the sizes of small flows are fixed at 50, 100, 150 and 200 KB, and the deadlines of them are fixed at 20 ms. We repeat the experiments 10 times, then the average, minimum, maximum deadline missing rates are depicted in Fig. 12 for DCTCP, D^{2}TCP and LPD-e. We can see that for a small flow size (i.e., 50 KB), no deadline is missed by LPD-e, while D^{2}TCP still misses a small percentage of them. The deadline missing rate of all schemes increase with the flow sizes, due to more traffic load, where LDP-e also shows more advantage.

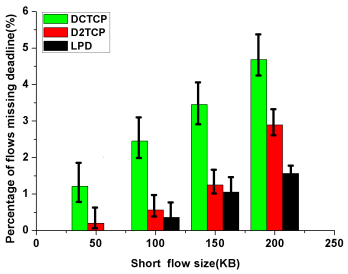
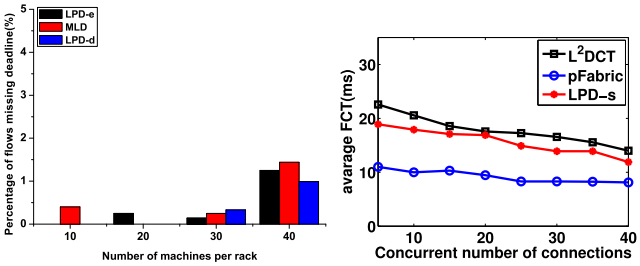


Fig. 12: Deadline miss with increasing flow sizes



(a) Different principles (b) Flow completion time

Fig. 13: Further extensions of the design principle

VI. 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 aforementioned equation (2) for congestion control. The first example is $f = \alpha/d$ where d is the same imminence factor used by D^{2}TCP, 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 LPD-d respects the more load more differentiation principle, and we call it MLD.

Fig. 13(a) depicts the results for LPD-e, MLD and LPD-d, against uniformly distributed deadlines. 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. Our final example is to demonstrate that, 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. It's known that pFabric [2] is a near optimal centralized scheduling algorithm, while L^{2}DCT [7] is a TCP-like congestion control algorithm, both designed for minimizing FCT. We set up concurrent flows sharing a bottleneck link of 1 Gbps, and let each flow sends 1 MB data. In Fig. 13(b), we plot the average FCT achieved by L^{2}DCT, pFabric and LPD-s, respectively, when the number of concurrent flows increases from 1 to 40. We can see LPD-s improves on L^{2}DCT, again because its more load more differentiation principle, while L^{2}DCT uses the gamma-correction function.

VII. 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 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^{2}TCP, and L^{2}DCT, especially under a heavy traffic load.

VIII. Acknowledgement

This research is supported by the National Natural Science Foundation of China (Grant No. 61402253, 61161140454 and 61225011), and project for 2012 Next Generation Internet technology research and development, industrialization, and large scale commercial application of China (No. 2012 1763).

References

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

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

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

[4] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.

[5] Latency. 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, 2009.

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

[7] A. Munir, I. Qazi, Z. Uzmi, A. Mushtaq, S. Ismail, M. Iqbal, and B. Khan. Minimizing flow completion times in data centers. In *INFOCOM, 2013 Proceedings IEEE*, pages 2157–2165, April 2013.

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

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

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

[11] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). SIGCOMM '12, pages 115–126, 2012.

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

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