

FDRC – Flow Duration Time Based Rate Control in Data Center Networks

Han Zhang^{*‡}, Xingang Shi^{†‡}, Xia Yin^{*‡}, Zhiliang Wang^{†‡}, YingYa Guo^{*‡}

^{*}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)
Beijing, P.R. China

Email: {zhanghan, wzl, yxia, guoyingya} @csnet1.cs.tsinghua.edu.cn, {shixg} @cernet.edu.cn

Abstract—Data Center is now becoming an important facility for many applications (e.g, web search and retail). As TCP can't meet applications' demands for latency and throughput, many tcp-based protocols (e.g, DCTCP, D2TCP, L2DCT) have been proposed. Among them, protocols such as D2TCP incorporate explicit deadline into congestion window adjustment procedure to guarantee flows' latency and protocols such as L2DCT consider flow size when computing congestion window adjustment factor to guarantee the throughput of short flows. These two methods work well at some scenery but they have some deficiencies on two aspects. Firstly, we find that they can only reduce the percentage of flows missing deadline or reduce flow completion time, but can not meet both the goals simultaneously. Secondly, most of these methods need the user to know flow information (e.g, deadline, flow size), which may be hard to know exact value beforehand. In this paper, we advocate to use flow duration time into congestion window adjustment procedure. Based on this, we propose FDRC—Flow Duration Time based Rate Control algorithm. We find that without knowing flow information beforehand, FDRC can achieve the goal of reducing the percentage of flows missing deadline and cutting average flow completion time simultaneously. We theoretically analyze FDRC's behavior and implement FDRC into ns-2 as well as linux kernel. Our experiments show that FDRC performs better than D2TCP and L2DCT at nearly all the scenarios. On average, it performs 30% better than the state-of-art deadline-aware congestion control protocol D2TCP and 10% better than the state-of-art flowsize-aware protocol L2DCT.

I. INTRODUCTION

Nowadays, more and more latency-sensitive applications (e.g, web search, retail) are deployed in data center networks. Data center networks (DCNs) require high throughput, low latency and high burst tolerance [8] [16] [17]. Recently, among these characters, latency attracts more attention, as it affects user experience in these applications and thus revenue. Indeed, every additional 100ms can cause 1% loss of revenue [3].

In particular, traffic in data center is mixture of long-live background traffic, delay-sensitive traffic and bandwidth-sensitive traffic, the traditional transport mechanism TCP falls short on two sides. Firstly, TCP can cause buffers overflow and this will affect user experience. Secondly, window will cut by half when con-

gestion happens and this causes link utilization low. Due to the pitfalls of TCP, DCTCP [3] is proposed. DCTCP uses ECN marking mechanism [3] which is supported by most modern switch to give feedback to senders, so that senders can adjust its congestion window according to congestion degree. DCTCP keeps switch's queue shallow and maintain link utilization high. However, DCTCP is deadline-agnostic and it treats all flows fair and many delay-sensitive flows will miss their time constraints [15] [14]. Then two types of methods are proposed to make up the deficiencies:

Set explicit deadline to flows. D3 [15], D2TCP [14] and LPD [16] allow users to set explicit deadline to flows from user space, so that congestion window can adapt to fit the time constraints. As a result, those emergency flows will get relatively larger bandwidth than the lax ones and this can help more flows finish before time constraints.

Set high priority to short flows. In DCNs, time-constraint flows are always short [9], so another class methods such as PDQ [9], L2DCT [13] and pFabric [5] try to set short flows high priority. They try to realize SJF (shortest job first) schema, which proves to be optimal for reducing average flow completion time in single bottleneck link scenery, so that more short urgent flows can finish before time constraints.

We conclude that both methods try to set higher priority to urgent flows to let more flows finish before time constraints. The difference between them is that the first kind of methods set priority according to users' expected latency the second kind set priority according to flow size. Both methods need the users to know flow information beforehand. That means, if we use the first class of protocols, we should know flow time constraints in advance and if we use the second class, flow size is needed beforehand. Besides, traffic in data center is mixture of delay-sensitive flow, bandwidth-sensitive flow and long-live background flow, the first class of methods just meet the requirements of delay-sensitive flow and the second class of methods just tries to reduce average flow completion time. Neither of the methods can meet the demands simultaneously.

Because of the flaws, we advocate flow's priority can be set according to duration time from flow starting. Flows have the highest priority when they begin to transfer and their priority descends as time goes on. We think this a good choice, because on one hand, we do not need to know flow information (flow size or deadline) beforehand; on the other hand, by setting high priority to those short duration ones, the latency-constraint flows and the small flows will get more bandwidth on average, so that the demands of reducing the percentage of flows missing time constraints and reducing average flow completion time can be met. Based on this, we propose FDRC-Flow Duration Time Based Rate Control algorithm. We implement FDRC into ns-2 [2] and linux kernel 3.2.61. Our experiment results demonstrate FDRC improves about 30% over D2TCP and 2X over DCTCP. Thus the contribution we make in this paper:

- A data center transport protocol FDRC that helps more flows to meet deadline and helps to reduce average flow completion time.
- Design and analyze congestion control mechanism— FDRC which can reduce the percentage of flows missing deadline without setting explicit information to flows in advance.
- Extensive evaluation of FDRC in ns-2, then compare FDRC with DCTCP, D2TCP, L2DCT, LPD and pFabric.
- Evaluate FDRC into linux kernel and construct small data center testbed to test the performance of FDRC.

The rest of our paper is organized as follows. We first introduce some necessary background and related work in Section II. In Section IV, we propose FDRC and then we analyze it. Then we evaluate FDRC in Section V both by simulation and in real testbed. At last, we conclude the paper at Section VI.

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. As many applications (i.e. OLDI, web search) call for low latency and the traditional transport TCP can cause a large percentage of flows missing time constraints, so many new schedule and transport protocols are raised to be suited to these small urgent flows.

To keep switch buffer shallow, DCTCP [3] uses ECN marking mechanism. When packets arrive, if queue size is larger than threshold K , packets will be marked. Senders then compute α which reflects congestion degree. Then congestion window will be computed as $w = w * (1 - \alpha/2)$. DCTCP reduces congestion window according to link congestion degree. With the help of ECN marking mechanism, DCTCP has small queue delay. DCTCP is hardware-friendly protocol. However, for some applications like web search, DCTCP's fair sharing

is not a good choice, because flows in these applications call for different time constraints. Fair sharing policy can cause some urgent flows miss their time constraints.

Since different flows call for different latency, D3 [15] tries to set soft deadline constraints to flows. Bandwidth is computed at the sender side according to deadline, so urgent flows can get more bandwidth than lax ones, as a result, more flows will finish before their time constraints. Although D3 improves much over DCTCP, it also has some drawbacks: on one hand, it does changes to switch hardware and can not coexist with TCP; on the other hand, 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 [14].

Based on DCTCP, D2TCP [14] sets deadline constraints to flows, so urgent flows will have smaller congestion window reduction than lax ones. In D2TCP, flows reduce congestion window using $w = w * (1 - \alpha^d)$, where α is the congestion level as DCTCP and $d = TC/D$ is deadline factor. TC can be understood as the time needed for flow to complete its transmitting under the deadline-agnostic manner and D is the remaining time till deadline. D2TCP can coexist with TCP and does not need to change switch hardware. D2TCP's performance drops when load becomes heavy.

LPD [16] finds the defects of D2TCP and advocate congestion control function should follow the principle: more load, more differentiation. That means when load is heavy, it is more necessary to differentiate flows with different deadline request. LPD uses the linear function $f = \alpha * t/t_{max}$. LPD can co-exist with TCP and does not need to change switch hardware.

Until now, D3, D2TCP and LPD all try to set deadline to flows. As in data center networks, those time urgent flows are always the short ones, another class method tries to reduce average flow completion time or tail to help these ones finish before time constraints. As at single bottleneck link, SJF is considered to be optimal, so flow size is often chosen as the parameter.

L2DCT [13] bases on DCTCP and it alters both the additive increase part as well as the multiplicative decrease part of the window computation. L2DCT also uses gamma function as its penalty function, which is the same as D2TCP. In L2DCT, short flow have larger congestion window increment and smaller congestion window decrement than large flows. As a result, average flow completion time will be reduced.

PDQ [9] is a schedule method. PDQ works in distributed fashion which is different from D3. PDQ can emulate different scheduling policies like Earliest Deadline First (EDF) or Shortest Job First (SJF). However, PDQ should change switch hardware and can not coexist with TCP.

pFabric [5] advocates datacenter transport should decouple flow scheduling from rate control. For flow

scheduling, it uses Shortest Remaining Processing Time (SRPT) policy and for rate control. Flows start at line rate and throttle back only under high and persistent packet loss. Combining these two characters, pFabric achieve near-optimal performance for flow completion time. pFabric changes switch and can not co-exist with TCP.

PASE [12] claims protocols (DCTCP, PDQ, pFabric, etc) are not competitors, they are complementary. PASE does not change network elements and its performance is better than those state-of-art protocols [12].

We can summarize that all these protocols, no matter explicit or implicit, all need to set parameters (flow size, deadline, remaining flow size, remaining time, etc) or change network hardware. However, in data center, traffic is mixture of delay-sensitive flow, bandwidth-sensitive flow and long-live background flow, setting deadline to flows just reducing the percentage of flows missing deadline and using flow size can only reduce the average flow completion time of flows. Neither of the methods can meet both the demands at the same time. What we need is to meet the requirements simultaneously and do not need the user to provide flow information beforehand.

III. MOTIVATION

The state-of-art rate control algorithms, no matter setting explicit deadline to flows or setting high priority to short flows, can just meet either the requirements of reducing percentage of flows missing deadline or decreasing the average flow completion time, but they can not meet the requirements simultaneously.

To test this, in ns-2, we start 10 senders which connect with one receiver through a switch with $K=25$. All links in our simulation have the bandwidth of 1Gbps. Among the 10 servers, 2 of them send background traffic and the other 8 send short flows whose size range from 100KB to 1MB. We divide the short flows equally into two sets and have different expectation for the two sets. For the first set, we hope they can finish transferring before explicit time constraints 30ms and for the second set, we hope they have short average flow completion time. The simulation lasts 10 minutes and Fig .1 shows the result.

Fig .1(a) shows the percentage of flows missing deadline and Fig .1(b) depicts the average flow completion time. We can see for deadline miss comparison, D2TCP works better than L2DCT and DCTCP and for FCT, L2DCT performs better. We can explain this from flows' priority. For deadline miss comparison, in D2TCP, urgent flows have higher priority, so that they have larger bandwidth than the lax ones and as a result, more urgent ones will finish before their time constraints. However, in DCTCP and L2DCT, the first set flows have the same bandwidth as the second sets, so that a relatively higher percentage of flows miss deadline. For average flow completion time, L2DCT works better than D2TCP, this

is because in L2DCT, short flows have higher priority than large ones, but D2TCP only optimizes the first flow set and ignores the second sets. Fig .1(c) shows the bandwidth of background traffic. We can see that both L2DCT and D2TCP are more aggressive than DCTCP.

From Fig .1 we can conclude that D2TCP can reduce the percentage of flows missing deadline and L2DCT can cut average flow completion time. They can not satisfy both goals simultaneously. The reason for this is that D2TCP uses deadline as rate control factor, so urgent flows will have larger bandwidth than the lax ones. However, for those small ones which call for large bandwidth, D2TCP works as DCTCP. L2DCT uses flow size as rate control factor, so that small flows will have larger bandwidth than large ones. However, in data center, not all short urgent flows have explicit time constraints, some flows just need to transfer as quickly as possible (e.g, small request). As a result, L2DCT will lead to short average flow completion time, but maintains a relatively higher percentage of flow missing deadline.

As traffic in datacenter is mixture of deadline-aware flows and bandwidth sensitive flows, so that a protocol is needed to satisfy both goals simultaneously. We find, still now, neither using deadline nor using flow size can reduce the percentage of flows missing deadline or cut AFCT simultaneously. We propose to use flow duration time as the congestion window adjustment factor. That means when flows start, they will have the highest priority and with time goes by, their priority drops. We think this a good choice, because on one hand, as, short flows have larger average, so that we can have short average flow completion time; on the other hand, flows whose time constraints within the time threshold will also have larger average, so that a smaller percentage of flows will miss deadline.

IV. FDRC PROTOCOL

A. Flow Duration Time Based Rate Control

Flow duration time is a good option to use as the congestion window factor and based on this, we propose FDRC which means Flow Duration Time Based Rate Control. Algorithm 1 shows the way to compute the congestion window factor. In Algorithm 1, four arguments are needed. *threshold_tight* and *threshold_lax* are time threshold, DMAX and DMIN are deadline factor's upper and lower bound. When duration time is shorter than *threshold_tight*, flows stay at the highest priority. If flow duration time is larger than *threshold_lax*, flows stay at the lowest priority. For flow duration time between *threshold_tight* and *threshold_lax*, priority drops with time goes by.

Algorithm 1 runs per RTT. With time goes on, priority will drop, so those long-live background flows' average bandwidth will become smaller. As a result, short and urgent flows will have relatively higher average band-

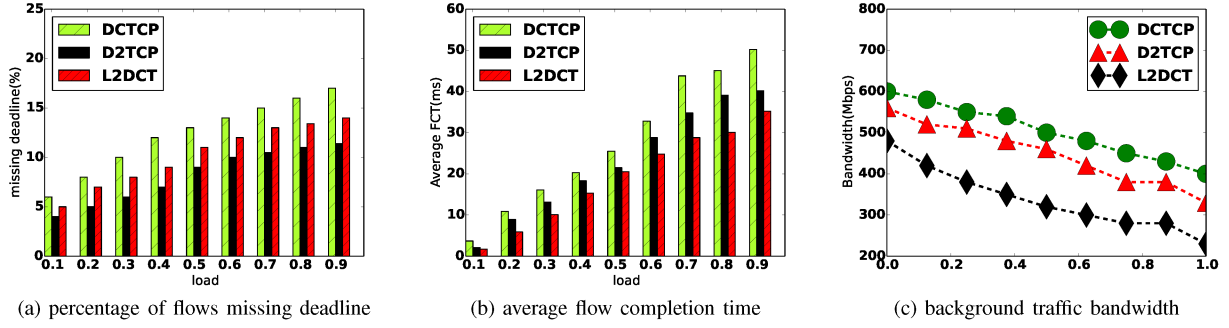


Fig. 1: motivation example: 50% of flows have explicit deadline while the other flows should transfer as quickly as possible

width than the long and lax ones, so that they can finish transferring before their time constraints.

Algorithm 1 Congestion window factor computation algorithm

```

1: durtime = time_now - time_start
2: if durtime ≤ threshold_tight then
3:   d = DMIN
4: else if durtime ≥ threshold_lax then
5:   d = DMAX
6: else
7:   d = DMIN + (durtime - threshold_tight) * (DMAX - DMIN) / (threshold_lax - threshold_tight)
8: end if

```

B. FDRC protocol

FDRC is based on DCTCP and uses ECN marking schema at the switch side. When queue size is larger than switch's threshold K , packets arriving will be marked. Then the receiver will check if the coming packets are marked and decides whether sends ACK back with ECN. The sender computes congestion degree α per RTT:

$$\alpha = (1 - g) \times \alpha + g \times F \quad (1)$$

F is the fraction of packets that are marked last RTT. g is weight factor. FDRC's congestion window function will be computed as:

$$f = \alpha \times d \quad (2)$$

According to [16], the additive increase part and multiplicative decrease part can be the same, so we alter both parts as:

$$p = \begin{cases} w + (1 - f) \\ w * (1 - f) \end{cases} \quad (3)$$

C. FDRC analysis

In this section, we analyze FDRC's performance in theory. We build two models to analyze FDRC: Sawtooth model and Fluid model.

1) *FDRC Sawtooth model*: We first build a two-flow sawtooth model [16] to analyze the bandwidth ratio of FDRC. R_1 and R_2 present the rate of $flow_1$ and $flow_2$. Assume flows' congestion windows increase from w_{min} to w_{max} , we have $w_{av} = \frac{w_{min} + w_{max}}{2}$. Flow's congestion windows behave like periodic sawtooth with fixed period N , we have $w_{max} = w_{min} + N \times (1 - f)$ and $w_{min} = w_{max} * (1 - f)$. Let w_1 be the maximum window size of flow1 and w_2 be the maximum window size of flow2, so the rate ratio can be computed as:

$$\begin{aligned} \frac{R_1}{R_2} &= \frac{(2 - f_1) \times w_1}{(2 - f_2) \times w_2} \\ &= \frac{2 - \alpha d_1}{2 - \alpha d_2} \times \frac{1 - \alpha d_1}{1 - \alpha d_2} \times \frac{d_2}{d_1} \end{aligned}$$

When d_1 and d_2 are small, we have:

$$\frac{R_1}{R_2} \simeq \frac{d_2}{d_1} \quad (4)$$

From (4) we can see for two-flow system, if d is small, rate ratio of the two flows inverses to their factor. To test our model, we start two long flows ($f1.start_time=0s$, $f2.start_time=0.5s$). Setting $threshold_{lax}=0.6s$, $threshold_{tight}=0.3s$, $DMIN=0.05$ and $DMAX=0.1$. As $threshold_{lax}=0.6s$ and flow1 starts at $t=0s$, so after $t=0.6s$, flow1's factor= $DMIN$. Flow2 starts at $t=0.5s$, so from $t=0.5$ to $t=0.8$, it has the maximum factor $DMAX$. After $t=0.8s$, its factor begins to decrease. From $t=1.1s$, flow1 and flow2 have the same factor. Fig. 2(a) shows the rate of the flows. From Fig. 2(a) we can see, from $t=0.6s$ to $t=1.1s$, flow1's bandwidth is larger than flow2 and after $t=1.1s$, their bandwidth are almost equal. Fig. 2(b) shows their bandwidth ratio and factor ratio, we observe from $t=0.6s$ to $t=0.8s$, the factor ratio is $DMAX/DMIN=2$ and the rate ratio of the two flows fluctuates around this. Then from $t=0.8s$ to $t=1.1s$, the ratio drops and rate ratio of the two flows also drops. After $t=1.1s$, the factor ratio is 1 and the two flows have nearly the same bandwidth.

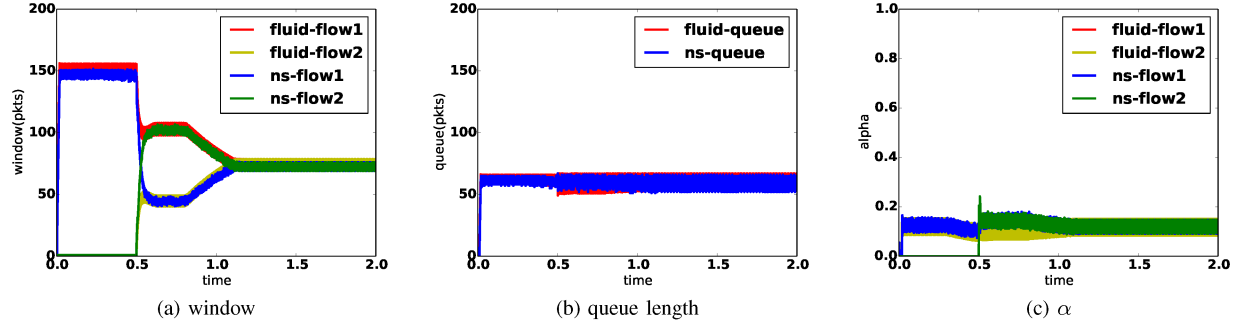


Fig. 3: Fluid model and ns-2 simulation comparison

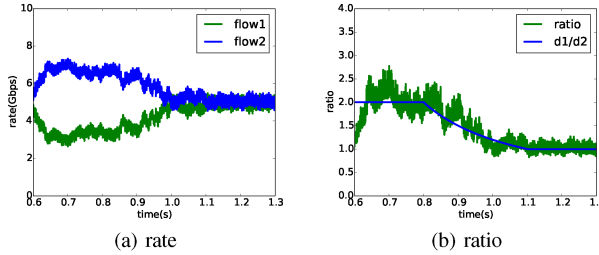


Fig. 2: (a) shows the bandwidth of the two flows. (b) shows the rate ratio and factor ratio

2) *FDRC Fluid model*: To understand the behavior of FDRC, we build fluid model according to [4] and [11]. Assume N flows transfer to the same receiver node through the bottleneck link whose capacity is C , the propagation delay is pd and FDRC fluid model is:

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

$$\hat{p}(t) = 1_{\hat{q}(t) > 1} \quad (6)$$

$$\frac{dq}{dt} = \sum_{i=1}^N \frac{w_i(t)}{R(t)} - C \quad (7)$$

$$\frac{dw_i}{dt} = \frac{1 - \alpha_i(t) \times d_i(t)}{R(t)} - \frac{w_i(t) \times \alpha_i(t) \times d_i(t)}{R(t)} p(t - R^*) \quad (8)$$

Equation (5) and equation (6) model the marking process of the switch. Equation (7) models the queue and Equation (8) describes the congestion window of FDRC.

To verify our model, we start two long flows (f1.time_start=0, f2.time_start=0.5s). $DMAX = 1$, $DMIN = 0.5$ and the threshold: $threshold_tight = 0.3$, $threshold_lax = 0.6$. For flows whose duration time are less than 0.3 will have a largest value of deadline factor and for flows whose duration time are larger than 0.6 will have smallest factor. Others' factors will drop with time goes by. We use FDRC model to predict congestion window,

α and queue. Then we compare the result with ns-2 simulation.

Fig.3(a) depicts congestion window. We can observe that at first only flow1 starts, so flow1 occupies the total link bandwidth. Then at $t=0.5$, flow2 starts. As at $t=0.5$, flow2's factor is larger than flow1's, so flow2 has a larger congestion window. Then at $t=1.1$ they have the same value of congestion function factor. Fig.3(b) shows queue size and Fig.3(c) depicts α . We can see that both Fluid model's queue length and α match quite well with the simulation result.

3) *Parameters selection*: In FDRC protocol, four global parameters should be set. Time threshold can be set according to application demand. For example, in data center, flows may have deadline ranging from tight to lax, so $threshold_tight$ can be set as tight deadline and $threshold_lax$ can be set as lax deadline. In this part we will analyze how to set $DMIN$ and $DMAX$.

At first, we get the fixed point of fluid model $(w_i, \alpha_i, q) = (\frac{1}{d_i} - 1, 1, \sum_{i=1}^N \frac{1}{d_i} - N - C \times pd)$. The fixed point means all packets are all marked and senders have fixed congestion window. In reality, queue can not be too large, because a large queue size can result to a long queue delay. We assume in FDRC system, queue size should be shorter than $K+N$, so we have:

$$\sum_{i=1}^N \frac{1}{d_i} - N - C \times pd \leq K + N \quad (9)$$

As the factor d ranges between $DMIN$ and $DMAX$, so for the scenery, flows' factor have the value of $DMIN$:

$$N \times \frac{1}{DMIN} - N - C \times pd \leq K + N \quad (10)$$

$DMIN \geq \frac{N}{2 \times N + C \times pd + K}$. From (3), we know the congestion window function $f < 1$, when $\alpha \times d$ achieves 1, $d < 1$, so $DMAX$ and $DMIN$ should be smaller than 1. $DMIN$ and $DMAX$ should satisfy:

$$\frac{N}{2 \times N + C \times pd + K} \leq DMIN < DMAX \leq 1 \quad (11)$$

According to (11), we can conclude value of DMIN and DMAX can not be too small. Because small value of DMIN can make queue long, as a result flows will have large queue delay and long latency.

V. EVALUATION

In this section, we test the performance of FDRC under ns-2 simulation and small testbed. We first solve the motivation problem that is proposed in the motivation section. Then, we compare deadline miss between different protocols and our simulation result shows the percentage of deadline missing is only 1/5 of DCTCP. After that, we verify the performance of flow completion time, we observe FDRC reduces about 50% FCT than DCTCP. At last, we evaluate FDRC in our small data center testbed.

A. motivation resolved

In this section, we first solve the problem that is proposed at section III. For FDRC, we set $threshold_{lax}=30ms$ and $threshold_{tight}=10ms$. DMAX = 0.5 and DMIN = 0.1. Fig.4 shows the result. We can see, FDRC has less percentage of flows missing deadline than D2TCP and flow completion time is shorter than L2DCT.

The reason for this is that FDRC uses flow duration time as its rate control factor. As for FDRC, $threshold_{lax}=30ms$ and $threshold_{tight}=10ms$, so flows whose time constraints ranging from 10ms to 30ms, will have larger average bandwidth than others. For those emergency flows will have larger bandwidth than the lax ones, so less flows will miss deadline. As flows have highest priority at first, then drops with time goes by, so short flows will have larger average bandwidth than long-lived flows, as a result, the average flow completion time will reduce.

B. At Scale Simulation In Spine-Leaf Data Center Topology

To demonstrate FDRC's performance with real data center traffic and complex data center topology, we construct the spineleaf data topology in ns-2 [2]. The topology is shown as Fig. 5. In our simulation, 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 14.6us and the end to end trip latency across the leaf switch (2 hops) is 13.3us. Among the time, 10us is cost at the end server, this simulates the computation time of the servers. All the protocol of experiment parameters we set are just the same as [5].

Benchmark workload. We use two types of real data center workloads: web search workload and data mining workload [3] [5]. For web search workload, 70% flows are smaller than 1MB and more than 50% of flows are between 100KB and 1MB. For data mining workload,

more than 80% flows are smaller than 10KB and more than 90% bytes are from the 4% large flows ($> 1MB$). In our simulation, different load means different rate of flows arrivals. As today's datacenter network are often oversubscribed [10], so in our experiments, we vary the oversubscription factor x 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. We compare FDRC with the state-of-art deadline-aware congestion control algorithm LPD-e [16], D2TCP [14] and the deadline-agnostic protocol DCTCP [3]. For traffic that are deadline-agnostic, we consider average flow completion time and compare FDRC with pFabric [5], DCTCP [3] and L2DCT [13].

Parameter selection. Four important parameters are needed in FDRC. As in data center network, flows always have deadline constrains ranging from 10ms to 30ms [15], so we choose $threshold_{lax}=30ms$ and $threshold_{tight}=10ms$. The factor boundary DMAX and DMIN can be set according to (11). In our simulation, DMAX is 0.5 and DMIN is 0.1.

1) Deadline miss comparison: In data center, delay-sensitive business such as web search should transfer the computation results to client within some milliseconds, otherwise user experience will be affected. In this section, we test the performance of FDRC when traffic are delay-sensitive. In our experiment, we set *Uniform* distribution deadline to flows to present users' expectation. Flows arrive across a Poisson Process. Flow's source and destination are random chosen. We also evaluate LPD-e, DCTCP and D2TCP and compare them with FDRC. Varying the oversubscription factor x and repeat each group of experiment 100 times, then Fig. 6 shows the result.

We can see overall, FDRC performs better than D2TCP and DCTCP. On average, FDRC reduces about 5X percentage of flows missing deadline than DCTCP and reduce about 2X than D2TCP. The reason for its advantage is that FDRC has two threshold $threshold_{lax}$ (30ms) and $threshold_{tight}$ (10ms), when flow duration time is shorter than $threshold_{tight}$, flow has the largest value of congestion window factor, so it can have large value of congestion window, thus rate. However, FDRC sometimes performs worse than LPD-e, this is because

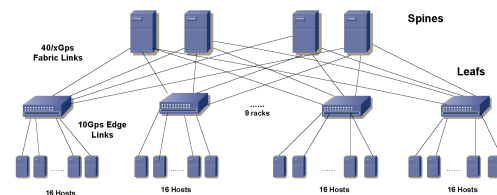


Fig. 5: The spine-leaf Data Center topology

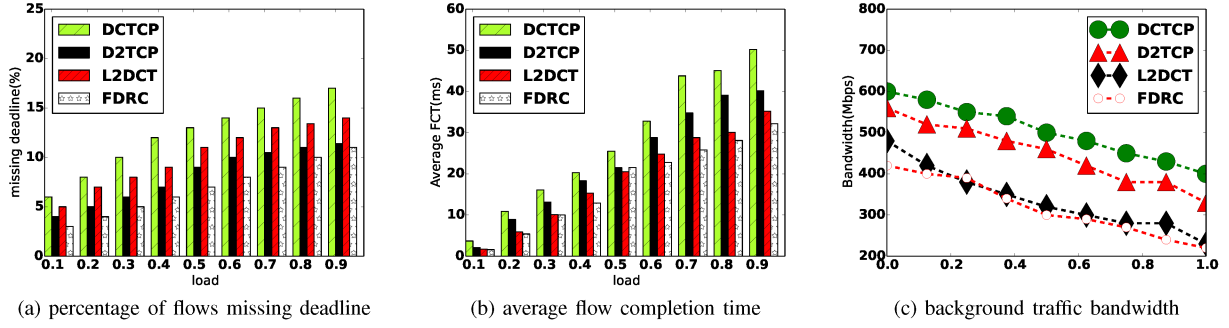


Fig. 4: motivation example: compare FDRC's performance with DCTCP, D2TCP and L2DCT

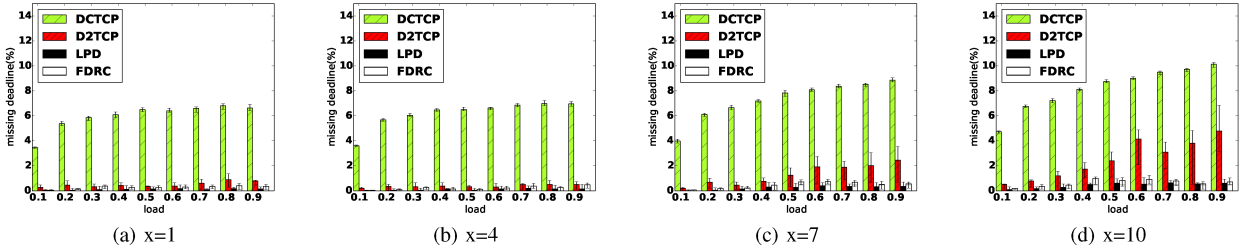


Fig. 6: Deadline miss comparison between FDRC, LPD-e, D2TCP and DCTCP under web search workload. Flows' deadline have uniform distribution between 10ms and 30ms

LPD-e adjusts congestion window using explicit deadline set by user, while FDRC is an implicit and indirect method. Although FDRC performs a little worse than LPD-e (about 0.5% more missing deadline), we regard FDRC as a better choice, because sometimes accurate deadline may be not easily known beforehand.

Specially, FDRC performs even well for the heavy congestion link condition. As Fig. 6(a) shown, when the oversubscription factor $x=1$, FDRC is only 20% better than D2TCP and when the factor $x=10$, Fig. 6(d) shows it performs about 5X times better than D2TCP. The reason for this is that FDRC's follows the principle of more load, more differentiation [16]. The result is more or less the same for data mining workload just as Fig. 7 shows. Note the reason for percentage of flow missing for data mining workload is a little larger than web search is that data mining traffic includes large flows ($> 10MB$) which can easily miss deadline.

In data center network, not all flows are latency-constraint, some flows are background flows. In this experiment, we divide the web search load traffic into two groups. The number of flows in each group is the same. For the first group, all flows have deadline which are exponential distribution. The second group flows are background flows which are deadline-agnostic. We use three kind deadline: tight (10ms), mild (20ms) and lax (30ms) [15] in our simulation. All flows arrive across a Poisson Process. The source and destination server are

random chosen. Fig. 8 shows the result. We can see that FDRC as well as D2TCP works better than DCTCP and FDRC performs better than D2TCP in most cases. We observe that FDRC only improves about 20%-30% over D2TCP, while in the previous experiment, it improves 2X. The reason for this is that, in this experiment, only half of the flows have deadline, FDRC treats the two group fair, but D2TCP let the first group flow have large congestion window.

2) *Flow completion time comparison:* For the whole fabric network, optimal algorithm for minimizing average FCT is equivalent to the NP-hard sum multi-coloring problem [6], but when scheduling over a single link is the Shortest Remaining Processing Time (SRPT) policy which always schedules the flow that has the least

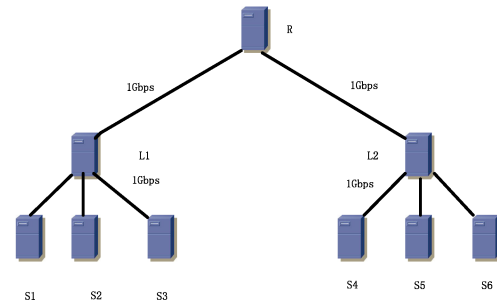


Fig. 12: The small scale data center testbed

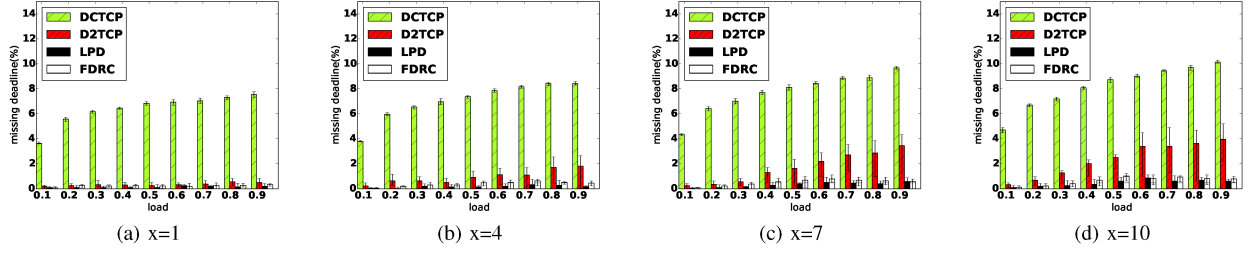


Fig. 7: Deadline missing comparison between FDRC and LPD, D2TCP, DCTCP under data mining workload. Flows' deadline have uniform distribution between 10ms and 30ms

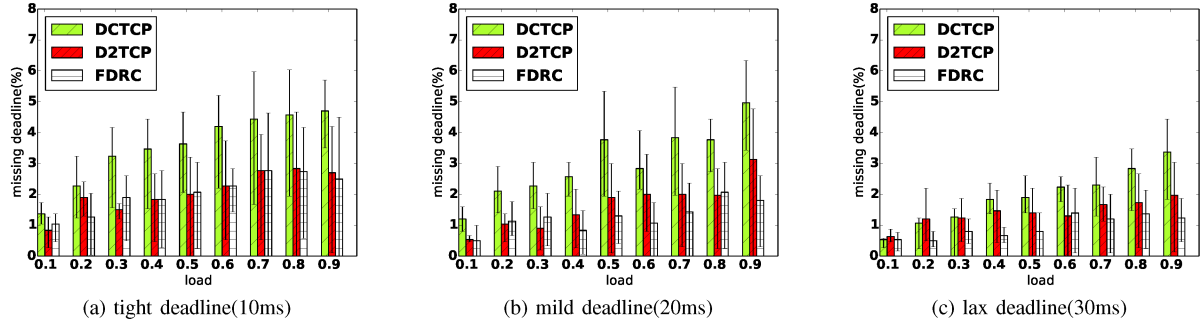


Fig. 8: Web Search workload of deadline missing comparison between FDRC and D2TCP, DCTCP. Note, in this experiment, deadline has exponential distribution and only half of the flows have deadline.

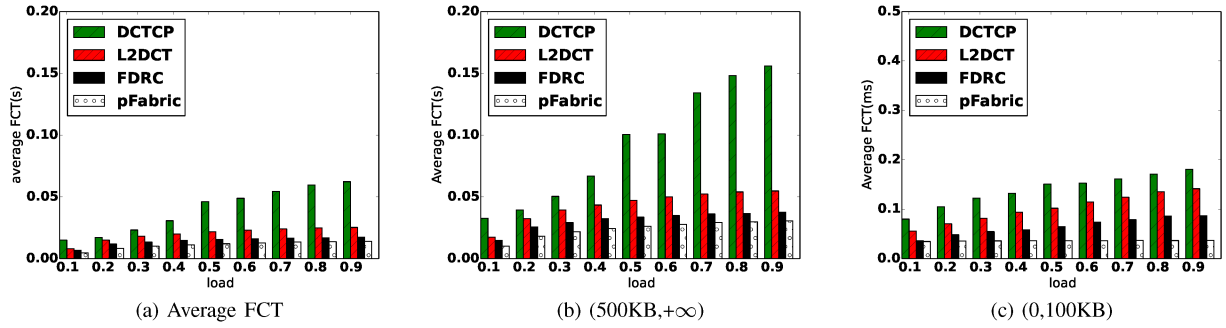


Fig. 9: Web Search workload, comparison of average flow completion time between FDRC, DCTCP, L2DCT and pFabric. Fig. (a) is the result of total flow. Fig. (b) shows the average of large flows ($> 500KB$) and Fig. (c) depicts the small ones

work remaining. FDRC reduces flow completion time by letting those short flows have higher priority throughout their transfer life, which approximates SRPT schema.

We compare the performance between FDRC and DCTCP, L2DCT, pFabric. Fig. 9 depicts the flow completion time for web search traffic. From Fig. 9(a), we can see that FDRC performs about 10% better than L2DCT, but 20% worse than the nearly optimal state-of-art protocol pFabric [5]. To see some detail of the experiment, we divide the traffic into large and small and show their respect FCT. From Fig. 9(b) we observe

the large flows' average completion time is about 40ms, which is about 30%-40% larger than pFabric. Fig. 9(c) shows for those small flow, FDRC only lags about 5% to pFabric. Fig. 10 shows the experiment result for data mining traffic. As many traffic flow in data mining work load is less than 10KB, so the average completion time for data mining traffic is shorter than the web search environment.

We vary the oversubscription factor from 5 to 40 to test the performance of FDRC. Fig. 11 shows the experiment result. From Fig. 11(a), we can see that

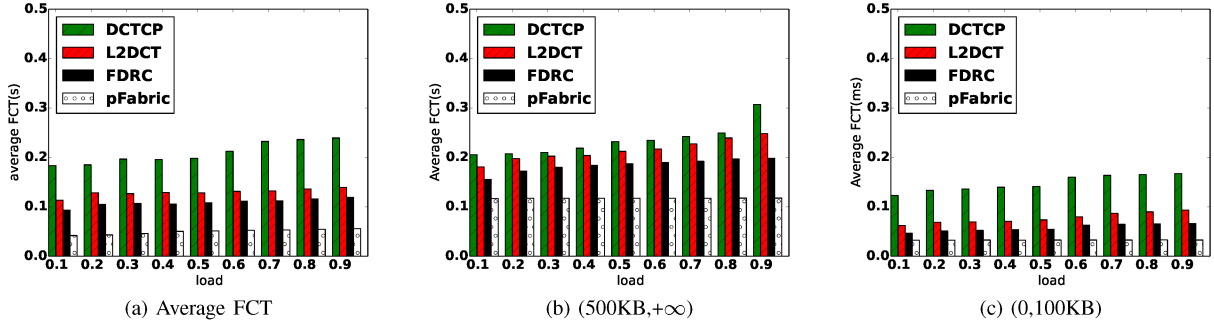


Fig. 10: Data Mining workload, comparison of average flow completion time between FDRC, DCTCP, L2DCT and pFabric. Fig. (a) is the result of total flow. Fig. (b) shows the average of large flows(> 500KB) and Fig. (c) depicts the small ones

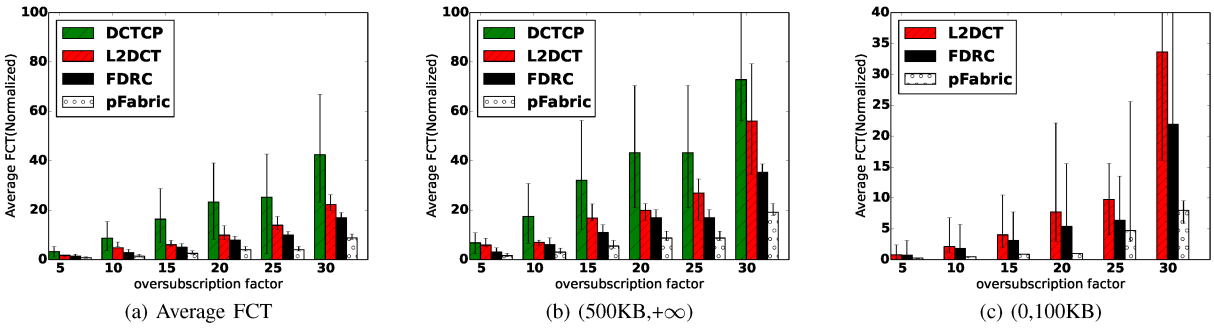


Fig. 11: Web Search workload, overall normalized average FCT for FDRC, DCTCP and pFabric when we varying the oversubscription factor from 5 to 30. Note that DCTCP does not appear in Fig.(c) as its value is larger than the plot range

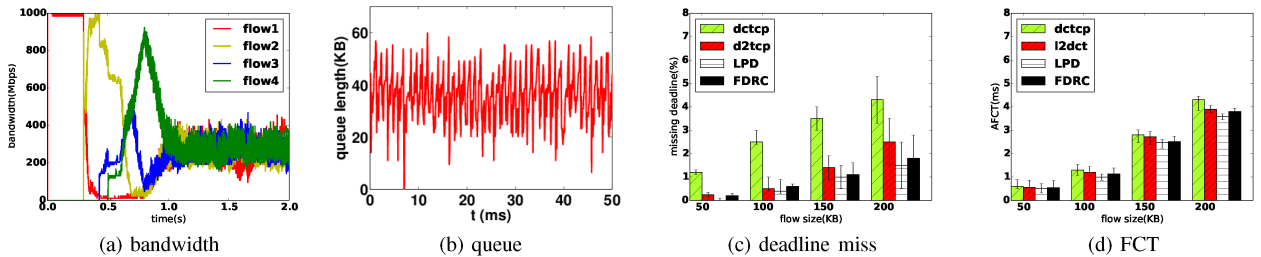


Fig. 13: Performance of FDRC under small testbed

comparing with DCTCP, FDRC reduces about 50% completion time. As FDRC also follows the principle "more load, more differentiation" [16], its performance can be guaranteed when congestion load is heavy. That's can explain why its performance improves more for the large oversubscription comparing with the small value.

C. small testbed evaluation

To test the performance of FDRC under a real environment. Based on DCTCP, we implement D2TCP and LPD, L2DCT in Linux kernel 3.2.61. We use the linux

serve as switch which runs FreeBSD 9.1 along with DummyNet [1] [7] to construct a two-way bottleneck link, so that packets can be forwarded and marked at line speed. Due to our limited hardware, we can only set up a small tree-based scale topology of 3 switches with 6 servers just as Fig. 12 shown.

1) *bandwidth*: FDRC flows can have higher priority when they start and their priority will drop with time goes by. In our testbed experiment, we set DMAX=0.5 and DMIN=0.1. The time threshold threshold_lax=200ms, threshold_tight=100ms. K=40KB at the

switch server. We then start 4 flows, flow1 is from S1, starts at $t=0$ s. flow2 from S2 and starts at $t=0.3$ s. flow3 is from S3 and starts at $t=0.4$ s. flow4 is from S3 and starts at $t=0.5$. All flows' destination are server R.

Fig. 12(a) shows the result. We can see all flows will have largest bandwidth when start and begin to slow with time goes by. Then at last, as all flows have the same priority, they share the link bandwidth. Fig. 12(b) shows the queue length on server L1, we can see FDRC can keep the queue at low and steady level.

2) *deadline miss comparison*: In this section we test the performance when there are delay-sensitive flows in data center. In our experiment, firstly, each server S_i first builds up a tcp connection with the random server. Each connection lasts T min. During the T min, server S_i will send small query flows to receiver. After T min, server S_i will choose another random server and repeat the process. We repeat the process 100 times and each group T=5 minutes. We set DMAX=0.5, DMIN=0.1 and the time threshold threshold_lax=200ms, threshold_tight=100ms, varying flow size, then Fig. 13(c) shows the result.

We can observe FDRC performs better than DCTCP and D2TCP. It reduces about half the number of flow deadline missing than DCTCP and 30% less than D2TCP. But FDRC's performance is about 5% worse than LPD. The experiment result is almost the same as our simulation.

3) *flow completion time comparison*: In this section we test FDRC's performance on reducing flow completion time. In our experiment, firstly, each server S_i first builds up a tcp connection with the random server. Each connection lasts T min. During the T min, the server random sends short flows to receiver. After T min, server S_i will choose another random server and repeat the process. We repeat the process 100 times and each group T=5 minutes, then Fig. 13(d) shows the result.

We can see that FDRC performs about 20% better than DCTCP, about 10% better than L2DCT and about 5% worse than LPD.

VI. CONCLUSION

In this paper, we use flow duration time to compute correction function factor. We advocate this suitable, because by setting threshold, those urgent flows will get more bandwidth than the background flows, thus they can finish transferring before their time constraints. Based on this, we propose FDRC and model FDRC. Then we test the performance of FDRC under ns-2 and linux kernel. Our experiments show FDRC performs 30% than D2TCP and L2DCT on average.

Although FDRC performs well under both simulation and small testbed, it requires rigorous analysis on stability and convergence properties, we will study this in the near future.

VII. ACKNOWLEDGEMENT

This work is supported by the National High Technology Research and Development Program of China (863 Program) No. 2015AA015603, No. 2015AA016105 and the National Natural Science Foundation of China (Grant No. 61402253).

REFERENCES

- [1] The dummynet project. [online]. available. <http://info.iet.unipi.it/luigi/dummynet/>.
- [2] The network simulator — ns-2. <http://www.isi.edu/nsnam/ns/>.
- [3] 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.
- [4] M. Alizadeh, A. Javanmard, and 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*, pages 73–84. ACM, 2011.
- [5] 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.
- [6] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shachnai. Sum multicoloring of graphs. *Journal of Algorithms*, 37(2):422–450, 2000.
- [7] Y. Cao, M. Xu, X. Fu, and E. Dong. Explicit multipath congestion control for data center networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 73–84. ACM, 2013.
- [8] Y. Guo, Z. Wang, X. Yin, X. Shi, and J. Wu. Traffic engineering in sdn/ospf hybrid network. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 563–568. IEEE, 2014.
- [9] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.
- [10] J. Lee, Y. Turner, M. Lee, L. Popa, S. Banerjee, J.-M. Kang, and P. Sharma. Application-driven bandwidth guarantees in datacenters. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 467–478. ACM, 2014.
- [11] V. Misra, W.-B. Gong, and 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*, volume 30, pages 151–160. ACM, 2000.
- [12] A. Munir, G. Baig, S. M. Iteza, I. A. Qazi, A. X. Liu, and F. R. Dogar. Friends, not foes: synthesizing existing transport strategies for data center networks. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 491–502. ACM, 2014.
- [13] 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.
- [14] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *SIGCOMM '12*, pages 115–126, 2012.
- [15] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: Meeting deadlines in datacenter networks. *SIGCOMM '11*, pages 50–61, 2011.
- [16] H. Zhang, X. Shi, X. Yin, F. Ren, and Z. Wang. More load, more differentiation: a design principle for deadline-aware congestion control. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 127–135. IEEE, 2015.
- [17] Y. Zhang, K. Xu, H. Wang, and M. Shen. Towards shorter task completion time in datacenter networks. In *Computing and Communications Conference (IPCCC), 2015 IEEE 34th International Performance*, pages 1–8. IEEE, 2015.