Deadline-Aware Datacenter TCP (D^2TCP)

Balajee Vamanan
Purdue University
bvamanan@ecn.purdue.edu

Jahangir Hasan
Google Inc.
jahangir@google.com

T. N. Vijaykumar
Purdue University
vijay@ecn.purdue.edu

ABSTRACT

An important class of datacenter applications, called Online Data-Intensive (OLDI) applications, includes Web search, online retail, and advertisement. To achieve good user experience, OLDI applications operate under soft-real-time constraints (e.g., 300 ms latency) which imply deadlines for network communication within the applications. Further, OLDI applications typically employ tree-based algorithms which, in the common case, result in bursts of children-to-parent traffic with tight deadlines. Recent work on datacenter network protocols is either deadline-agnostic (DCTCP) or is deadline-aware (D^3) but suffers under bursts due to race conditions. Further, D^3 has the practical drawbacks of requiring changes to the switch hardware and not being able to coexist with legacy TCP.

We propose Deadline-Aware Datacenter TCP (D^2TCP), a novel transport protocol, which handles bursts, is deadline-aware, and is readily deployable. In designing D^2TCP, we make two contributions: (1) D^2TCP uses a distributed and reactive approach for bandwidth allocation which fundamentally enables D^2TCP's properties. (2) D^2TCP employs a novel congestion avoidance algorithm, which uses ECN feedback and deadlines to modulate the congestion window via a gamma-correction function. Using a small-scale implementation and at-scale simulations, we show that D^2TCP reduces the fraction of missed deadlines compared to DCTCP and D^3 by 75% and 50%, respectively.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols.

General Terms

Algorithms, Design, Performance.

Keywords

Datacenter, Deadline, SLA, TCP, OLDI, ECN, Cloud Services.

1. INTRODUCTION

Datacenters are emerging as critical computing platforms for ever-growing, high-revenue, online services such as Web search, online retail, and advertisement. These services employ Online Data Intensive applications (OLDI) [18] which have two distinguishing properties: (1) Because application latency affects user experience, and hence revenue [13], OLDI applications operate under soft-real-time constraints (e.g., 300 ms latency). (2) The applications employ tree-based, divide-and-conquer algorithms where every query operates on data spanning thousands of servers [12].

An OLDI query's overall time budget gets divided among the nodes of the algorithm's tree (e.g., 40 ms for a parent-to-leaf RPC). To avoid missing its deadline, a parent node sends out an incomplete response without waiting for slow children that have missed their deadlines. Because such incomplete responses adversely affect response quality, and hence revenue, achieving fewer missed deadlines is important. While a node's response time includes both computational and network latencies, we focus on reducing the network delay. We note that in addition to achieving fewer missed deadlines, a network protocol that allows tighter network budgets is invaluable as it allows more time for computation, thus producing higher-quality responses.

A key reason for increased network delay is that all the children of a parent node face the same deadline and are likely to respond nearly at the same time, causing a fan-in burst at the parent in the common case [10][1][25]. Further, because typical data centers host multiple applications at the same time to enable flexible use and high utilization of the datacenter resources, OLDI flows with different deadlines and background flows with no deadlines (e.g., Web index update) share the network. As such, multiple such bursts coinciding in time at a switch may lead to congestive packet drops and TCP retransmits, which frequently result in missed deadlines. We emphasize that the fan-in bursts are fundamental to OLDI applications and are not artifacts of the network. The shallow- and shared-buffer nature of datacenter switches, combined with the buffer-filling nature of long-lived TCP flows, precludes absorbing these bursts in packet buffers [1]. Current datacenters alleviate this problem by a combination of two approaches: (1) over-provision the network link bandwidths to absorb the bursts, and (2) increase the network's time budget (e.g., equal to, say, the 99th percentile of the observed network delay) leaving less time for computation. While the former incurs high cost, the latter either degrades response quality (e.g., less time for Page Rank in Web search), or requires more machines to compensate for less computation per machine and may worsen fan-in bursts by increasing the fan-in degree (i.e., more children per parent).

Recent work on datacenter networks either reduces the tail-end network latency or proposes deadline-aware networks. DCTCP [1] is an elegant proposal that targets the tail-end latency by gracefully throttling flows in proportion to the extent of congestion, thereby reducing queuing delays and congestive packet drops and, hence, also retransmits. DCTCP reduces the 99th-percentile of the network latency by 29%. Unfortunately, DCTCP is a deadline-agnostic protocol that equally throttles all flows, irrespective of whether their deadlines are near or far. D^3 shows that as much as 7% of flows may miss their deadlines with DCTCP [25]. D^3 tackles missed deadlines by pioneering the idea of incorporating deadline awareness into the network. In D^3, the

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.

switches grant a sender's request for the required bandwidth based on the flow size and deadline.

While D^3 improves upon DCTCP, D^3 has significant performance and practical shortcomings, which we cover in detail in Sections 2.4.2 and 2.4.3. On the performance side, D^3 employs a centralized and pro-active approach in which the switches allocate bandwidth greedily on a first-come-first-served basis. Such a greedy approach may allocate bandwidth to far-deadline requests arriving slightly ahead of near-deadline requests. We show in Section 4.2.2 that, due to this race condition, D^3 inverts the priority of 24%-33% requests, thus contributing to missed deadlines. Fixing these priority inversions is a hard problem as we explain in Section 2.4.2. On the practical side, D^3 has some major drawbacks as well. D^3 requires custom switching chips to handle requests at line rates. Such custom hardware would incur high cost and long turn-around time that hinders near-term deployment. Further, because legacy TCP flows do not recognize D^3's honor-based bandwidth allocation, D^3 cannot coexist with TCP. Such lack of protocol interoperability prevents incremental deployment necessary for introducing new technologies and for high availability during upgrades. As such, any datacenter protocol must be able to coexist with legacy TCP to be deployable in the real world.

In summary, DCTCP improves tail latency but is not deadline-aware whereas D^3 is deadline-aware but it has both performance and practical shortcomings. We stipulate that a datacenter network protocol should:

- meet OLDI deadlines, especially in fan-in-burst-induced congestion;
- achieve high bandwidth for background flows;
- work with existing switch hardware; and
- be able to coexist with legacy TCP.

We propose Deadline-Aware Datacenter TCP (D^2TCP), a novel transport protocol based on TCP that meets the above requirements. In designing D^2TCP, we make two contributions.

Our first contribution is D^2TCP's distributed and reactive approach. Because having global up-to-date information for all flows in a datacenter is technically infeasible given the rapid arrival rate and latency of flows, any network scheduling scheme must work with incomplete information. D^2TCP approaches this challenge by inheriting TCP's distributed and reactive nature, and adding deadline awareness to it. In contrast to D^3's centralized bandwidth allocation at the switches, which rules out per-flow state, D^2TCP's distributed approach allows the hosts to maintain the needed state without changing the switch hardware. In contrast to D^3's pro-active approach, which does not allow for correcting the decisions resulting from inaccurate information, D^2TCP's reactive approach allows senders to correct any temporary and small over-subscription of the network which can be absorbed by typical packet buffers.

Our second contribution is D^2TCP's novel congestion avoidance algorithm which uses ECN feedback and deadline information to modulate the congestion window size via a gamma-correction function [2]. The key idea behind the algorithm is that far-deadline flows back off aggressively and near-deadline flows back off only a little or not at all. Our algorithm simultaneously satisfies the four conditions stipulated above so that D^2TCP (1) achieves deadline-based prioritization in the presence of fan-in-burst-induced congestion so that flows with nearer deadlines are prioritized, while ensuring that congestion does not worsen; (2) achieves high bandwidth for background flows even as the short-lived D^2TCP flows come and go; (3) requires no changes to the switch hardware, so that deployment amounts to merely upgrading the TCP and RPC stacks; and (4) coexists with legacy TCP, allowing incremental deployment.

We demonstrate a real implementation of D^2TCP on a small 16-server testbed, and show that even at such small scale where fan-in-burst-induced congestion is much less severe than at real scale, D^2TCP reduces the fraction of missed deadlines by 20% compared to DCTCP, while requiring fewer than 100 additional lines of kernel code.

We perform further evaluations using at-scale simulations to show that D^2TCP

- reduces the fraction of missed deadlines compared to DCTCP and D^3 by 75% and 50%, respectively;
- achieves nearly as high bandwidth as TCP for background flows without degrading OLDI performance;
- meets deadlines that are 35-55% tighter than those achieved by D^3 for a reasonable 5% of missed deadlines, giving OLDIs more time for actual computation; and
- coexists with TCP flows without degrading their performance.

Note that we do not present formal proofs for D^2TCP's fairness and stability.

The remainder of the paper is organized as follows. In Section 2, we discuss the nature of OLDI applications and pinpoint the issues with the previous proposals. We describe the details of D^2TCP in Section 3. In Section 4, we describe our experimental methodology and present our experimental results. We discuss some related work in Section 5, and conclude in Section 6.

2. OLDIs AND DATACENTER NETWORK PROTOCOLS

We describe the nature of today's datacenter applications, and how this nature interacts with the previous proposals for datacenter network protocols.

2.1 Online Data Intensive Applications

As the name suggests, there are two defining properties of Online Data Intensive (OLDI) applications: *online* and *data-intensive* [18]. *Online* implies an interactive nature, wherein a user typically inputs a query via a browser and expects an immediate response. Consequently, OLDI applications are designed to respond within a short deadline (e.g., 300 milliseconds). *Data-intensive* means that the applications consult large data sets (e.g., entire index of the web) for computing the response. The large volume of data is typically distributed over thousands of servers, and each query hits every server.

The two properties combined lead to tree-based, divide-and-conquer algorithms for OLDI applications [12], as shown in Figure 1. The specific example we show is a two-level tree, however, the properties we describe hold for shallower and deeper trees as well. The user query arrives at the root, which broadcasts the query down to the leaves across which the data is partitioned. Each leaf sends its response to its parent which aggregates the results from all the leaves and sends a response to the root. The root, in turn, aggregates all the parents' results and ships off the final response to the user. This overall application architecture is called *scatter-gather* or *partition-aggregate*. The propagation of the request down to leaves and of the responses back up to the root must complete within the deadline. That overall budget gets divided among the levels of the tree. For example, a leaf may have to respond to its parent within 30 milliseconds (Figure 1). When this deadline expires, the parent aggregates the available results and ships off the final response. Any leaf that misses its deadline

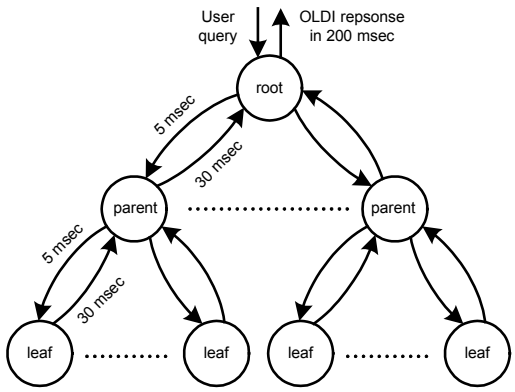

Figure 1: OLDI architecture

fails to contribute to the final response. Such missed deadlines result in incomplete, lower-quality responses.

It is instructive to note that each leaf's budget gets divided into two parts: the computation time on the leaf, and the communication latency between the leaf and its parent. This division balances two competing demands. A generous budget for communication ensures fewer missed deadlines by the network, but also means less time for computation, thereby penalizing the quality of results (e.g., less time for Page Rank in Web search). The upshot is that not only is it desirable to have fewer missed deadlines, but there is also great value in tightening the communication budget to give more time for computation.

2.2 OLDI Fan-in Congestion

Consider a parent's subtree when the parent sends a query to its leaves. Because all leaves receive the query at nearly the same time, and because they all face the same deadline, the leaves are likely respond around the same time. The result is a burst of responses fanning in to the parent in the common case [10] [1]. While such bursts can be smoothed by inserting jitter, doing so increases the tail-end latency [9], and is therefore of limited use. Further, because datacenters run multiple applications at the same time to improve utilization, multiple fan-in bursts (from same or different applications) may coincide in time at the same switch.

In addition to the bursty fan-in traffic described above, datacenter networks also carry background traffic consisting of long-lived flows. These flows push new control information and data to the nodes of the OLDI applications. While this traffic is usually not constrained by tight deadlines, it does involve large data transfers. Given the nature of TCP, these long flows tend to exercise the switch buffers to high utilization [1].

The combined network traffic described above often results in fan-in-burst-induced congestions which cause tail drops as the packet buffers fill up. Absorbing these bursts in larger packet buffers is precluded by two factors. First, datacenter switches employ network ASICs with on-chip packet buffer memory. Given the limitations of die size, on-chip packet buffers are naturally shallow. Switch designs with larger off-chip packet buffers are significantly more expensive and complex [14], and reserved for high-end core routers that must buffer for Internet scale RTTs. Second, the nature of long-lived TCP flows is to fill up larger buffers, which may lead to longer queuing delays and still cause missed deadlines for OLDI traffic [1].

While fan-in-burst-induced congestions are a fundamental characteristic of OLDI applications, the manner in which today's datacenter networks handle such fan-in bursts contribute to missed deadlines in two ways.

First, a TCP/IP network uses packet drops as a feedback to inform the senders about on-going congestion[1]. Under this mechanism, the sender must wait for a timeout to detect packet loss even as the deadline expires. Furthermore, the sender also halves its transmission rate to alleviate the congestion. The net result is that the leaves involved in a fan-in-burst-induced congestion are likely to miss their deadlines. Current datacenters address this problem by a combination of two approaches: (1) increasing the network link bandwidths and (2) increasing the network time budget to be greater than, say, the 99th percentile network latency. While the former incurs high cost, the latter reduces the time budget for computation, which either penalizes the response quality or requires more machines to compensate for less computation per machine and may worsen fan-in bursts by increasing the fan-in degree (i.e., more children per parent).

Second, TCP treats all congested flows equally which is sub-optimal when the congested flows' deadlines are different. Ideally, a network should prioritize the flows that are about to miss their deadlines while throttling the flows that can afford to wait. However, TCP is a *fair-share*[2] protocol which lacks such deadline-based distinction. Because fan-in bursts are common in OLDIs and fair-share protocols are deadline-agnostic, such protocols are not well-suited to datacenters [25].

2.3 Datacenter TCP (DCTCP)

Explicit Congestion Notification (ECN) [21] is an extension to the TCP/IP protocol that enables congestion feedback without using packet drops as the feedback mechanism. ECN relies on Active Queue Management (AQM) schemes like RED [8] to track congestion at a switch. When a switch encounters sustained congestion, it marks the Congestion Encountered (CE) bit in the IP header, thereby informing the endpoints about the congestion. The endpoints observe this CE bit feedback and reduce their transmission rate.

DCTCP [1] shows that ECN does not suffice to solve OLDI's fan-in burst problem. In datacenters, the number of congested flows is small enough that their congestion windows tend to be synchronized with each other. Furthermore, the traffic is bursty in nature. Therefore, halving the flows' windows in response to the ECN feedback causes the flows to thrash instead of gracefully converging to the available bandwidth. In summary, one bit of ECN feedback merely indicates the *presence* of congestion, but it carries no information about the *extent* of congestion.

To address this problem, DCTCP elegantly aggregates the one-bit ECN feedback from multiple packets and multiple RTTs to form a multiple-bit, weighted-average metric for sizing the window. Using this metric, the senders modulate their window sizes in a graceful manner, without thrashing. As a result, DCTCP reduces the 99th percentile of network latency in OLDIs by 29%. Thus, DCTCP frees up more time for computation in OLDIs.

2.4 Deadline Driven Delivery (D^3) Protocol

While DCTCP addresses the first issue mentioned in Section 2.2 (packet drops causing time-outs), DCTCP, being a fair-share protocol, does not address the second issue (lack of deadline awareness). As such, a later work called Deadline Driven Delivery (D^3) shows that DCTCP may cause up to 7% of the deadlines to

[1] The use of RED [8] without ECN merely triggers this feedback earlier than the onset of full congestion, but the feedback mechanism remains the same -- a packet drop.

[2] Strictly speaking TCP has an RTT-bias but we use the term fair-share as TCP *tries* to treat flows equally and the bias is an undesired side-effect.

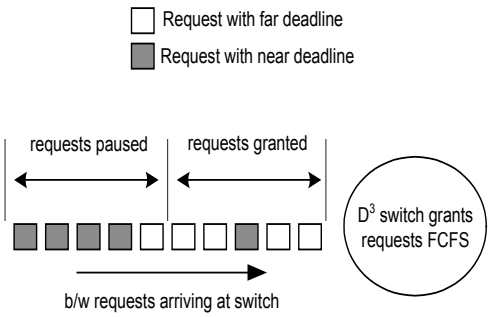

Figure 2: D3's priority inversion

be missed [25]. Citing TCP's problems, D^3's authors argue that TCP is fundamentally ill-suited for OLDIs and opt for an altogether new protocol.

D^3 pioneered the idea of incorporating deadline awareness into the network. To that end, D^3 proactively allocates bandwidth based on the idea of bandwidth reservation before data transmission. As applications know the deadline for a message, they pass this information to the transport layer in the request to send. Based on their deadlines and the amount of remaining data, senders must request bandwidth every round trip time (RTT) and send only the corresponding amount of data. Switches receive these requests, compute a decision, and place the decisions into the packet header. Thus, the senders learn how much data to transmit in the next RTT. Because requests are made in a deadline-aware manner, D^3 dramatically reduces the fraction of missed deadlines compared to DCTCP's [25].

2.4.1 D^3 operation

For every RTT, a sender computes the needed bandwidth as the amount of remaining data divided by the time until the deadline. The sender places the request into the D^3 packet header. The very first request in a flow uses a *SYN*-like packet carrying the request and no data payload. In subsequent RTTs, the bandwidth request for the next RTT is piggybacked on the data packets of the current RTT.

The switches receive these packets and extract the bandwidth request. The switches also maintain state for the bandwidth already allocated (and hence the remaining bandwidth) along the path in question. Based on this state, each switch makes a greedy choice and tries to grant as much of the request as possible. The switch places its grant-response in the packet header and forwards the packet. Each switch along a packet's path performs the same action, creating a vector of bandwidth grants in the header. The receiver copies this vector into the ACK packet back to the sender. The sender extracts the vector and chooses the minimum of all the grants to decide how much data to transmit for the next RTT.

2.4.2 Challenges in centralized and proactive scheduling

D^3 employs a centralized and pro-active approach to scheduling the network. To maximize network utilization, the switches allocate bandwidth greedily on a first-come-first-served basis. Because fan-in-burst-induced congestion is common in OLDIs, near- and far-deadline traffic often competes for bandwidth. Unfortunately, D^3's greedy approach may allocate bandwidth to far-deadline requests arriving slightly ahead of near-deadline requests (see Figure 2). Due to this race condition, D^3 causes frequent priority inversions, which contribute to missed deadlines. Indeed, our results in Section 4.2.2 show that D^3 inverts the priority of 24%-33% requests.

Fixing these priority inversions within a centralized, proactive design space is hard for multiple reasons. First, given the rapid arrival rate and tight deadlines of flows, it is not feasible for a switch to wait and gather information on all flows before making decisions. Instead the switch must make decisions on-the-fly without knowledge of near-future requests. Second, because maintaining detailed per-flow state at the switches is prohibitively complex, D^3 has no memory of which flows recently encountered priority inversions. If requests continue to arrive in an unfavorable order, the same priority inversions may happen again in subsequent RTTs. Third, maintaining some spare bandwidth in anticipation of future bursts is also hampered by the lack of detailed per flow state. Allocation of such spare bandwidth requires a priority list, so the switch can decide whether a given request should be granted out of the spare pool or be denied. Such a priority list again requires tracking detailed per-flow state. Further, a large spare may underutilize the network whereas a small spare may be insufficient to absorb bursts. A scheme that dynamically adapts the amount of the spare bandwidth, must balance the conflicting needs of maximizing utilization and accommodating near-deadline bursts, which is an open and complex research problem.

The above analysis assumes that D^3's bandwidth requests contain deadline information, even though as proposed in [25], D^3's bandwidth requests do not contain deadline information.

2.4.3 Challenges in practical deployment

To handle requests at line rates, D^3 requires custom switch ASICs. Unfortunately, such OLDI-specific, custom silicon would not only incur high cost due to low volumes, but also incur long turn-around time hindering near-term deployment.

Further, D^3, as proposed, cannot coexist with TCP. TCP flows passing through the same switches as D^3 flows would not recognize D^3's honor-based bandwidth allocation. Placing TCP and D^3 flows in separate QoS classes would provide isolation but would also require partitioning the bandwidth among the QoS classes. Such partitioning has far-reaching consequences and raises the hard issue of optimizing multiple applications' bandwidth shares, fairness, and network utilization. Tunneling TCP traffic inside D^3 is problematic because it results in two nested flow control loops with conflicting behavior. For example, TCP may increment its window size, while D^3 is decrementing it, and the surplus segments would incur TCP time-outs, back-off and retransmits. Protocol interoperability requirements in datacenters, though less stringent than those in the Internet, cannot be ignored completely as suggested by D^3. Due to the lack of interoperability, the upgrade of switches and applications to D^3 would all have to occur in an atomic manner. For such large, invasive changes to be atomic, the datacenter would have to be unavailable for a long enough time that the changes are all but implausible. Indeed, datacenter infrastructure upgrades are almost always incremental for this reason, so that old and new technologies coexist for some time. Finally, it may not be reasonable to expect *all* application writers to abandon TCP and develop D^3 versions just because *some* applications desire the use of D^3. As such, any datacenter protocol must be able to coexist with legacy TCP to be deployable in the real world.

3. Deadline-Aware Datacenter TCP (D^{2}TCP)

We now describe the design of Deadline-Aware Datacenter TCP (D^{2}TCP), a novel protocol for datacenter networks. We set out with the explicit goals of not requiring custom hardware and supporting coexistence with legacy TCP. The basic idea behind D^{2}TCP is to modulate the congestion window size based on both deadline information and the extent of congestion. In designing

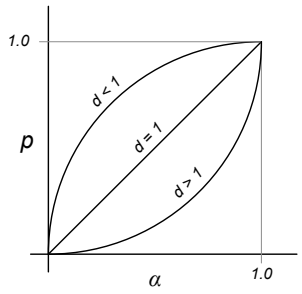

Figure 3: Gamma-correction function for congestion avoidance ($p = \alpha^d$)

D^{2}TCP, we make two contributions: D^{2}TCP's distributed and reactive approach for allocating bandwidth and D^{2}TCP's deadline-aware congestion avoidance algorithm.

3.1 Distributed and Reactive Allocation

Meeting deadlines in fan-in-burst-induced congestion requires knowledge of the flows' deadlines and the extent of congestion. However, having complete and up-to-date global information for all flows in a datacenter is technically infeasible given the rapid arrival rate and latency of flows. Therefore, any network scheduling scheme must make decisions with incomplete information, and the challenge is to choose a compromise that is well suited to OLDIs in datacenters.

D^3's centralized and proactive approach compromises the handling of future bursts in order to maximize utilization. Because OLDI traffic is bursty, D^3 suffers from frequent priority inversions, making D^3's compromise not well suited to datacenters. In contrast, D^{2}TCP inherits TCP's distributed and reactive nature, and adds deadline awareness to it. While D^{2}TCP, like D^3, also makes decisions without complete and accurate information, D^{2}TCP's compromises are better suited for datacenters.

D^{2}TCP modulates the window size based on the deadlines and the extent of congestion. Each D^{2}TCP sender sizes its window without knowing how many other flows are congested and by how much other flows will back off. The risk here is that multiple congested senders with tight deadlines may refuse to back off and over-subscribe to bandwidth at the congestion point. Fortunately, due to their stateful nature, the senders can correct this over-subscription by reacting to future congestion feedback in a careful and calculated manner, ensuring that the oversubscription is only temporary and small. Networks are equipped to deal with temporary oversubscriptions by virtue of packet buffers. We note that while D^{2}TCP inherits it's distributed and reactive nature from TCP, our contributions are in adding deadline awareness without abandoning TCP's time-tested distributed and reactive approach; and in identifying and analyzing the fundamental difference between D^3 and D^{2}TCP, and explaining why those differences matter in the context of datacenter network protocols.

To summarize, in contrast to D^3's centralized bandwidth allocation at the switches, which rules out per-flow state, D^{2}TCP's distributed approach allows the hosts to maintain the needed state without changing the switch hardware. In contrast to D^3's pro-active approach, which does not allow for correcting the decisions resulting from inaccurate information, D^{2}TCP's reactive approach allows senders to correct any temporary and small over-subscription of the network. Thus, the compromises made by a distributed, reactive scheme are better suited for datacenters.

3.2 Deadline-aware Congestion Avoidance

Our second contribution is D^{2}TCP's novel congestion avoidance algorithm. Like D^3, we assume that applications pass the deadline information to the transport layer in the request to send data. This information then enables D^{2}TCP to modulate the congestion window size in a deadline-aware manner. When congestion occurs, far-deadline flows back off aggressively, while near-deadline flows back off only a little or not at all. With such deadline-aware congestion management, not only can the number of missed deadlines be reduced, but also tighter deadlines can be met because the network adapts to the time budget. D^{2}TCP requires no changes to the switch hardware, and only requires that the switches support ECN, which is true of today's datacenter switches. Therefore, D^{2}TCP deployment amounts to merely upgrading the TCP and RPC stacks.

3.2.1 Congestion avoidance algorithm

The easiest way to explain the D^{2}TCP congestion avoidance algorithm is to start with DCTCP and build deadline awareness on top of it. We expect that the switches are ECN-capable and are configured to mark CE bits when the packet buffer occupancy exceeds a certain threshold. Like DCTCP, we maintain α, a weighted average that quantitatively measures the extent of congestion:

$$\alpha = (1 - g) \times \alpha + g \times f$$

Here f is the fraction of packets that were marked with CE bits in the most recent window, and g is the weight given to new samples. We now define d as the deadline imminence factor, and explain its derivation later in Section 3.2.3. For now it suffices to know that a larger d implies a closer deadline. Based on α and d we compute p, the penalty function applied to the window size, as follows:

$$\boldsymbol{p = \alpha^d}$$

This function was originally proposed for color correction in graphics [2], and was dubbed gamma-correction because the original paper uses γ as the exponent. Note that being a fraction, $\alpha \leq 1$ and therefore, $p \leq 1$. After determining p, we resize the congestion window W as follows:

$$W = W \times \left(1 - \frac{p}{2}\right), \quad if\ p > 0$$
$$= W + 1, \quad if\ p = 0$$

In the case where α is zero (i.e., no CE-marked packets, indicating absence of congestion) and therefore p is zero, the window size is grown by one segment similar to TCP. And when all packets are CE-marked, $\alpha = 1$ and therefore $p = 1$, then the window size gets halved similar to TCP. For α between 0 and 1 the window size is modulated by p.

Figure 3 plots p, with a number of different curves for various values of d. The straight line in the middle shows $d = 1$, and hence $p = \alpha$. The curves below the straight line are for $d > 1$ (i.e., near-deadline flows incur lower penalty), and the curves above are for $d < 1$ (i.e., far-deadline flows incur higher penalty). Note that when $p = \alpha$ the behavior matches DCTCP. Essentially, DCTCP is a special case of D^{2}TCP where $d = 1$ for all flows all the time, so they back off equally upon congestion irrespective of their deadlines. Accordingly, in D^{2}TCP we use $d = 1$ for long flows that do not specify deadlines (i.e., D^{2}TCP behaves like DCTCP).

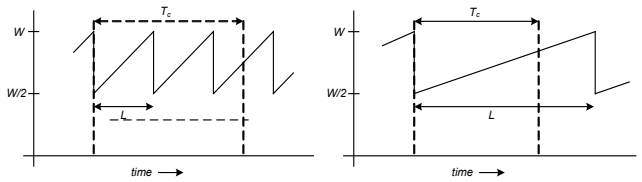

Figure 4: Sawtooth waves for deadline-agnostic behavior (a) $T_c > L$ and (b) $T_c < L$

3.2.2 Impact of d on congestion behavior

Looking at Figure 3 we see that when $d > 1$ (i.e., near-deadline flows), p increases only slowly in response to increases in α until α gets close to 1, at which point p rapidly converges to 1 as well. In other words, minor and mild congestions do not penalize near-deadline flows by much. However, severe congestions cause a full backoff similar to TCP and DCTCP, That is, when $\alpha = 1$, $p = 1$ the window size gets halved just like regular TCP.

When $d > 1$ (i.e., far-deadline flows), p increases rapidly even with small increases in α, and approaches 1 as α catches up to 1. Thus, even minor congestions cause rapid reduction in far-deadline flows' window sizes, but severe congestions do not penalize the flows any more than regular TCP or DCTCP would.

The combination of $d < 1$ and $d > 1$ behaviors complement each other under congestion situations. Far-deadline flows relinquish bandwidth so that near-deadline flows can have greater short-term share in order to meet their deadlines. Furthermore, if congestion continues to worsen after far-deadline flows have backed off, then two possible scenarios are at play: (1) there are many near-deadline flows not reducing their share and (2) there may be regular TCP flows consuming bandwidth in a deadline-agnostic manner. In both scenarios, the $d > 1$ condition ensures that as α grows even near-deadline flows will throttle themselves, allowing other near-deadline and regular TCP flows to make progress. In fact, if the near-deadline flows have different deadlines and hence different d, then the flows' back-off behavior will diverge as the congestion worsens. As a result, only the flows with the most imminent deadlines will win.

In summary, the gamma-correction function provides iterative feedback to near-deadline flows so that they do not drive the network to congestive collapse.

3.2.3 Determining d based on deadlines

We now explain how we determine d based on a given deadline value. The value of d should be such that the resulting congestion behavior allows the flow to safely complete within its deadline. We use deadline-agnostic congestion behavior ($p = \alpha$ and $W = \frac{w}{2}$ upon full congestion) as a starting point. Say T_c is the time needed for a flow to complete transmitting all its data under deadline-agnostic behavior; and D is the time remaining until its deadline expires. Now if the flow can just meet its deadline under deadline-agnostic congestion behavior (i.e., $T_c \cong D$) then $d = 1$ is appropriate. It also follows that if $T_c > D$ then we should set $d > 1$ to indicate a tight deadline, and vice versa. Therefore, we compute d as:

$$\boldsymbol{d = \frac{T_c}{D}}$$

Because d appears in the exponent of the gamma-correction function, extreme values for d can result in p behaving like a binary value for the useful mid-range of α. In other words, the mere *presence* of congestion would determine window sizing behavior, instead of the *extent* of congestion. Essentially, we want the gamma-correction function to yield gentle curves about the straight line of $d = 1$. Therefore, we propose capping d to be within a desired range (Section 4.1.1). We explore the effects of varying this cap in Section 4.2.4.

We now compute T_c using Figure 4 which shows the sawtooth wave for deadline-agnostic congestion behavior. We pessimistically assume that the flow encounters full-on congestion (i.e., $\alpha = 1$ and $p = 1$); if there is less congestion then the flow will complete sooner, making T_c an upper bound for time to completion. We assume that a flow's current window size is W and it has B bytes remaining to transmit. Further, we make the following simplifying assumptions that congestion occurs in a repeating pattern: (1) Full-on congestion occurs when the window size is W; (2) consequently, the window size is halved in the next RTT eliminating the congestion; and (3) in response, the window size is increased by one segment every RTT until the window size reaches W again at which point full-on congestion occurs again. The resultant sawtooth wave has a time period L.

While T_c can be computed using a precise analysis, we found that a reasonable approximation suffices for achieving successful deadline-aware congestion avoidance. In Figure 4(a), the average window size over the duration of T_c is $\frac{3}{4}W$. Therefore, T_c can be approximated as:

$$\boldsymbol{T_c = B/(\frac{3}{4}W)}$$

Though the average window size is different in the case of Figure 4(b), we still use this approximation for T_c in all our evaluations. A precise analysis of Figure 4(a) yields the following expression:

$$B = \left(\frac{W}{2} + \frac{W}{2} + 1 + \frac{W}{2} + 2 + \cdots \frac{W}{2} + L - 1\right) * \frac{T_c}{L}$$

Solving for T_c, we get:

$$T_c = \frac{B}{\frac{3W}{4} - \frac{1}{2}} \qquad \text{if } T_c > L$$

Similarly, an analysis of Figure 4(b) yields:

$$B = \frac{W}{2} + \frac{W}{2} + 1 + \frac{W}{2} + 2 + \cdots + \frac{W}{2} + T_c - 1$$

Solving for T_c, we get:

$$T_c = -\frac{W-1}{2} + \sqrt{1/4(W-1)^2 + 2B} \qquad \text{if } T_c < L$$

We found that D^{2}TCP performs similarly with precise and approximate expressions. Therefore, we do not present any evaluations using the precise expressions.

3.2.4 Stability and convergence

As stated above, our computation of d makes some approximations. Fortunately, D^{2}TCP has some in-built self-correction. If the algorithm under-estimates a flow's d, the flow may back off too much. However, in subsequent RTTs, the flow's D will decrease more than a commensurate decrease in its B. As a result d will increase, causing the flow to increase its transmission rate. The same is true for over-estimations of d. The flow will transmit faster than needed, and its B will decrease faster than needed. Consequently, subsequent RTTs will yield a lower d, thus correcting the over-estimation. Furthermore, if the aggressive transmission leads to severe congestion, then the natural response of the gamma function will dial up the value of p and throttle the flow as well.

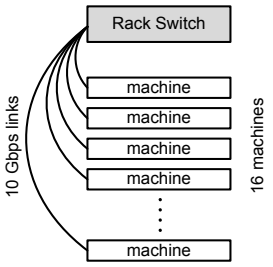

Figure 5: Real implementation testbed

Another issue to consider with deadline-aware scheduling is the possibility of inadvertently driving the network into congestive collapse. Imagine if all flows were to demand unreasonably tight deadlines, then their resulting congestion avoidance behavior would be to refuse downsizing their windows, effectively overloading the network. However, D^{2}TCP's gamma-correction function guards against such overload at two levels: (1) The effect of tighter deadlines is captured by the value of d which, in turn, determines the window size (hence sending rate). Because we cap the maximum value of d at 2.0, we limit how aggressively near-deadline flows can send. Therefore, deadlines that are tighter beyond a certain limit get rounded down so applications will not drive the network into congestive collapse. (2) At extreme congestion, as α and p approach 1, D^{2}TCP defaults to TCP. Therefore, D^{2}TCP's worst case stability is similar to that of TCP.

3.2.5 D^{2}TCP summary

The D^{2}TCP algorithm is as follows: A sender that does not encounter CE-marked packets increases its window size by one segment; otherwise, the sender computes α, T_c, and then d. The sender then uses the gamma-correction function to obtain p and resizes the congestion window based on p.

We do not change other aspects of TCP, such as slow start, and retransmission and timeout when there is a packet loss.

Note that when flows do not have a deadline (e.g., background flows) we use $d = 1$ so that D^{2}TCP defaults to DCTCP. The stability of DCTCP for long flows is examined in [1].

4. EVALUATION

We evaluate D^{2}TCP using both simulations and a real implementation. We use the real implementation to run a set of microbenchmarks on a small testbed, and to validate our simulator. We rely on simulations to evaluate production-like workloads at the scale of thousands of servers. Furthermore, because D^3 requires custom hardware, we limited our comparisons against D^3 to simulations.

4.1 Small-scale Real Implementation

We first present evaluations on a small-scale real implementation of DCTCP and D^{2}TCP. We use a set of microbenchmarks that examine the basic functionality of D^{2}TCP as a deadline-aware network protocol.

4.1.1 Implementation methodology

We started with the publicly available DCTCP source code from [4]. We then built our D^{2}TCP implementation on top of DCTCP, which amounted to around 100 additional lines of code. We instrumented D^{2}TCP to operate over only a select range of TCP ports, thus allowing us to use the same kernel for different protocols. We deployed this implementation on the testbed depicted in Figure 5. The testbed consists of a Top-of-Rack switch

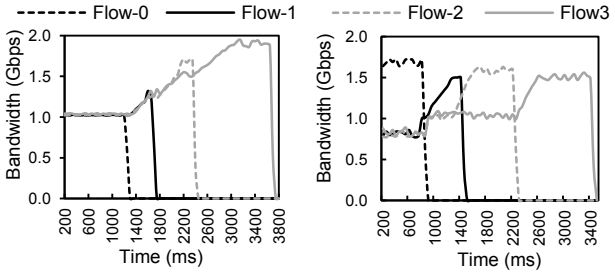

Figure 6: Throughput for DCTCP (left) vs. D^{2}TCP (right)

connected to 16 server machines. The switch is based on a Broadcom Scorpion ASIC with 24x 10Gbps ports and a 4MB shared packet buffer, and the servers connect to the switch via 10Gbps links.

We set the key parameters of DCTCP and D^{2}TCP to match those in [1]: (1) g, the weighted averaging factor is 1/16; and (2) K, the buffer occupancy threshold for marking CE-bits, is 20 for 1Gbps links, and 65 for 10Gbps links. For D^{2}TCP we cap d, the deadline imminence factor, to be between 0.5 and 2.0 (except in Section 4.2.4, where we explore the effects of varying this cap). We set RTO_{min} for all the protocols to be 20 ms.

4.1.2 Deadline awareness

We begin by examining the fundamental ability of D^{2}TCP to schedule the network in a deadline aware manner via deadline-aware congestion avoidance. In this experiment we have four hosts transmit flows to a fifth "root" host. We choose flow sizes and deadlines to illustrate the impact of a deadline-aware protocol. We set the flow sizes as 150, 220, 350 and 500 MB, with respective deadlines of 1000, 1500, 2500 and 4000 ms. Note that the flow sizes and deadlines in this synthetic test are not intended to model a real workload.

In Figure 6 we show the throughput achieved by the various flows over time, for both DCTCP and D^{2}TCP. The difference between DCTCP and D^{2}TCP is most noteworthy in the 0-2200 ms timeframe. As expected, DCTCP grants all flows equal bandwidth, and consequently the near-deadline flows miss their deadlines. In contrast, D^{2}TCP's deadline-aware congestion avoidance allows the near deadline flows to take a larger share of the available bandwidth, and the far-deadline flows commensurately relinquish bandwidth. Consequently, the flow completion times for flow #0 and #1 under D^{2}TCP are significantly shorter than those under DCTCP. As the number of active flows decrease, the opportunity for deadline-aware scheduling among them also decreases. Consequently, flow #2 and #3 have similar completion times under both schemes. These results establish the utility of a deadline aware network protocol.

4.1.3 Mixing deadline and non-deadline traffic

Recall that when flows have no deadlines we set $d = 1$ which causes D^{2}TCP to behave identical to DCTCP. DCTCP's stability and convergence for traffic patterns that consist entirely of such long non-deadline flows are examined in [1] in detail. We do not evaluate such traffic mixes in our paper, as the results for D^{2}TCP would be identical to that for DCTCP.

Instead, we examine traffic patterns that consist of a mix of deadline and non-deadline traffic. To that end, we set up a small-scale OLDI application with one root and the number of leaves varying between 20 and 40. Because the testbed has only 16 server machines, we run multiple leaves on each physical machine. The root periodically sends a query to all the leaves, which in turn, idle for a fixed "computation time" and then respond to the query. The replies from the leaves are sized 100-

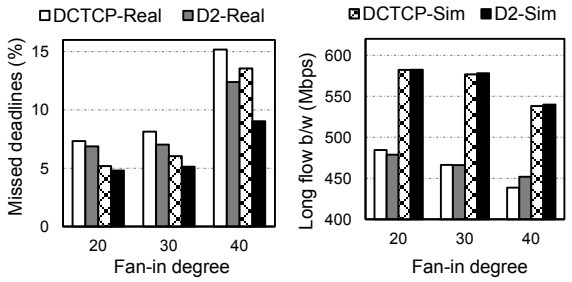

Figure 7: Missed deadlines (left) & bandwidth of long flows (right) in real implementation vs. simulation

500 KB, and have a deadline of 5-25 ms. In addition to this OLDI traffic, we have two servers initiate long-lived flows of 10MB to the root, once every 80 ms. The experiment lasts for the duration of 1000 OLDI queries. The aggregate traffic results in a network utilization of around 10% which is a realistic load for datacenters, and not an artificial overload scenario. Our goal in this experiment is to compare DCTCP and D^{2}TCP in the real implementation.

In Figure 7(left) we show the percentage of missed deadlines in the real implementation and in simulation of DCTCP and D^{2}TCP (shown as DCTCP-Real, D2-Real, DCTCP-Sim, and D2-Sim), while varying the fan-in between 20 and 40. We discuss DCTCP-Sim and D2TCP-Sim results in Section 4.1.4. We see that, across the board, D2-Real misses fewer deadlines than DCTCP-Real, with the difference increasing as we increase the fan-in. At a fan-in of 40, DCTCP-Real misses 15.2% of deadlines, whereas D2-Real misses 12.3% of deadlines.

In Figure 7(right) we show the throughput achieved by the non-deadline long flows for varying fan-in degrees. We see that at a fan-in degree of 40, the throughput achieved by D2-Real is 451 Mbps, which compares favorably to the throughput of 438 Mbps for DCTCP-Real. Therefore, the performance of deadline flows is not improved by degrading the throughput of non-deadline flows.

4.1.4 Simulator validation

To validate our simulator, we compare the real implementation and simulation results in Figure 7. We notice that the absolute numbers from the simulation are slightly off from the real implementation. For example, the percentage of missed deadlines for both DCTCP and D^{2}TCP under simulation is lower than those under real implementation. Such discrepancy is expected as simulations do not capture all the details and nuances of a real system, such as burst-smoothing jitter caused by unpredictable system events, interrupt coalescing, Large Segment Offload (LSO), and other TCP quirks.

Nevertheless, the relative performance difference between DCTCP and D^{2}TCP, and the trend in that difference, are similar across simulation and real implementation. At a fan-in of 30 in Figure 7, D^{2}TCP achieves 16% reduction in missed deadlines over DCTCP under simulations, whereas under real implementation the reduction is 15%. As we increase the fan-in degree, the relative performance difference between DCTCP and D^{2}TCP increases under both simulation and real implementation. Because of these key similarities, we believe that our at-scale simulation results are trustworthy. In addition, we also ensure that our simulation results closely match published DCTCP and D^3 results [25], as we show in the next section.

4.2 At-Scale Simulations

We now present our at-scale simulations. We model the network topology and the traffic after typical production deployments. Recall from Section 1 that D^{2}TCP's goal is to reduce the percentage of missed deadlines without degrading throughput for long-lived flows. Therefore, we focus on these metrics in our at-scale simulations.

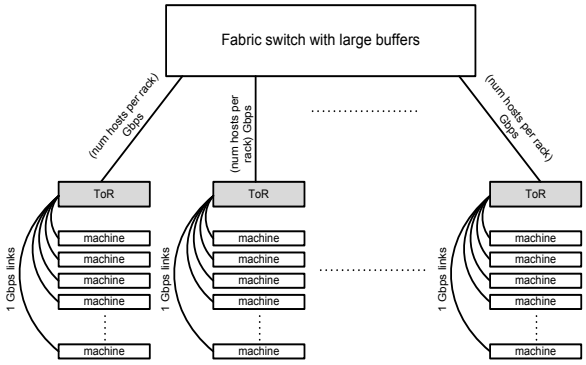

Figure 8: Simulated network

4.2.1 Simulation methodology

We implemented DCTCP and D^{2}TCP on top of ns-3's TCP New Reno protocol [19], and enabled the marking of CE bits in the switch model of ns-3. For D^3, we wrote both the end-host protocol and the switch logic, based on the details in [25]. We set D^3's base rate to be one segment per RTT. Further, we use the same RRQ packet format described in [25] including the 8-bit bytes-per-microsecond field. All DCTCP and D^{2}TCP parameters match those in Section 4.1.1.

We performed our simulations on the network depicted in Figure 8, which uses a fat-tree topology typical of datacenter networks. There are 25 racks with each rack having up to 40 end-host machines. Thus, our simulations capture the behavior of a 1000-machine deployment. Each end-host connects to the top-of-rack (ToR) switch via a 1 Gbps link. Because the bottleneck in datacenter networks is usually the ToR switch [10] [1] [25], we abstracted away the rest of the fat-tree topology, replacing it with one large fabric switch with large buffers. Each ToR is connected to the fabric switch via a single link with a line rate equal to 1 * *number-of-hosts-in-a-rack* Gbps. We sized the packet buffers in the ToR switches to match typical buffer sizes of shallow-buffered switches in real data centers (4MB) [1]. We set the link latencies to 20 µs, achieving an average of RTT of 200 µs, which is representative of datacenter network RTTs.

We ran a set of five synthetic OLDI applications on the network, equally dividing the total number of end-hosts among the applications. The assignment of an application node to a physical end-host is random to capture the effects of (1) applications dynamically requesting and relinquishing virtual machines in the data center, and (2) virtual to physical machine assignment being completely fluid. Each application consists of a set of five identical OLDI trees, each with one parent and *n* leaves, which have the same settings for leaf-to-parent message size and for deadlines. These settings are different across the five applications. We varied *n*, the number of leaves per parent, in the trees to explore varying degree of fan-in-bursts.

The distributions of message sizes and of deadlines in real OLDIs are publicly available [1]. However, details such as the exact deadline for given leaf-to-parent ratio and message size, are not publicly available. Therefore, like [25], we chose semi-synthetic values for deadlines using the aforementioned distributions.

Whether a message misses its deadline or not depends entirely on the precise deadline assigned to a message size. As

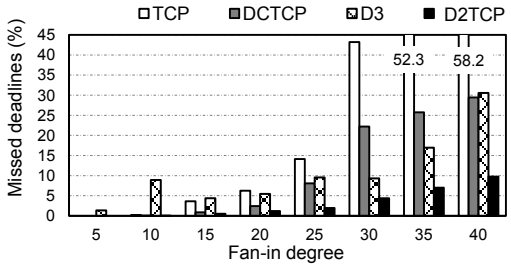

Figure 9: Missed deadlines for low variance (10%)

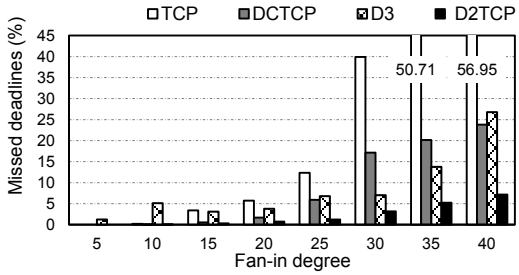

Figure 10: Missed deadlines for med. variance (50%)

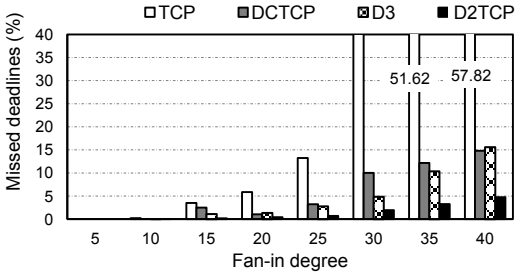

Figure 11: Missed deadlines for high variance (Exp)

such, the choices of the deadlines directly impact the results of any experiment. Therefore, we carefully calibrated our message sizes and deadlines such that the number of missed deadlines for DCTCP and D^3 are in line with the results in the D^3 paper [25]. This corroboration is discussed in detail in Section 4.2.2. We set the five OLDI applications' message sizes to 2, 6, 10, 14, and 18 KB and deadlines to 20, 30, 35, 40 and 45 milliseconds, respectively. We used these calibrated message sizes and deadlines in all our experimental results, except in Section 4.2.6. Note that the message sizes (fixed size of few KBs), long flow sizes (a few MB), and number of concurrent long-lived connections (1 connection) are chosen to match the characteristics of production workloads [1]. In all our experiments, the network utilization is 10-20% which is a realistic load for datacenters, and not an artificial overload scenario.

4.2.2 OLDI performance

Recall that in OLDIs the parent-to-leaf RPC has an overall budget that gets divided into computation and network parts. If all leaves were to finish computation at exactly the same instant for every single query, then the network deadline would be a hard and fixed value. In reality, the computation time can vary across leaves if the work for a query is not evenly balanced. While the computation budget forms a hard upper bound, some leaves may respond sooner resulting in a slightly looser *effective* network deadline. In addition, some applications attempt to smooth fan-in bursts via user-injected jitter [9]. As such, the exact nature and architecture of an OLDI application can affect the distribution of

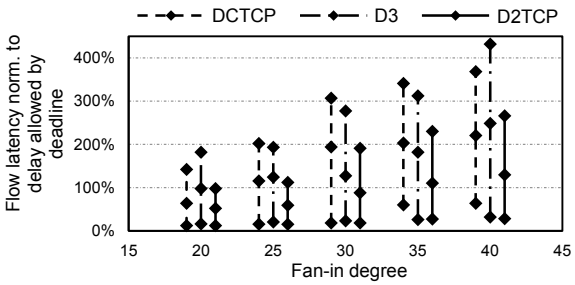

Figure 12: Normalized latencies at the 50th, 90th, and 99th percentiles

effective deadlines. We stipulate that a robust network protocol should work across a spectrum of deadline distributions. To that end, we evaluate three deadline distributions which use the same base deadlines as specified in Section 4.2.1, to which each adds a different variation. Our low variance case models a 10% uniform-random variation added to the base deadline. The medium variance case adds a 50% uniform-random variation, and the high variance case models a one-sided exponential distribution with mean equal to the base deadline. Our high variance case matches the deadline distribution used in [25].

We now compare D^{2}TCP against DCTCP and D^3 in terms of the fraction of missed deadlines for our benchmark. Figure 9, Figure 10, and Figure 11 correspond to the low, medium and high variance deadline cases. In all three graphs, the Y axis shows the fraction of missed deadlines for TCP, DCTCP, D^3, and D^{2}TCP as we vary the degree of burstiness on the X axis by increasing the fan-in degree (i.e., number of leaves per parent) from 5 to 40. Typical OLDI applications' fan-in degrees fall in this range [1] [25].

Across all three graphs, we see that TCP misses a rapidly increasing number of deadlines as we increase the fan-in degree (i.e., burstiness). While DCTCP and D^3 improve over TCP, they still miss a significant fraction of deadlines. For the medium variance case both DCTCP and D^3 miss around 25% of the deadlines at the fan-in degree is 40. In comparison, D^{2}TCP keeps the fraction of missed deadlines under 7% at a fan-in degree of 40. Because this trend holds true for all three deadline variation cases, we argue that D^{2}TCP is robust enough to handle a wide spectrum of deadline distributions. For the remainder of the results section, we use only the medium variance case for all remaining experiments.

All the schemes perform better with the higher-variance deadlines. Such behavior is expected because higher variance smoothes out fan-in-burst-induced congestion. Note that for the high variance case, the fraction of missed deadlines for D^3 fall in the range of 0-15% which is close to that of 0-9% reported in the D^3 paper [25], confirming that our D^3 implementation is reasonable.

To explain the above results, we show another set of data for the medium variance case. Figure 12 shows the 50th, 90th, and 99th percentile latencies for DCTCP, D^3 and D^{2}TCP normalized to the delay allowed by the deadline. On each line, the three points from bottom to top correspond to the 50th, 90th, and 99th percentile latencies, respectively. As expected, D^{2}TCP's latencies are significantly lower than those of DCTCP and D^3, resulting in fewer missed deadlines. Overall, the latencies for all the schemes closely track the fraction of missed deadlines in Figure 10.

We now examine our earlier claims about D^3's shortcomings. Recall that D^3's greedy approach may allocate bandwidth to far-deadline requests arriving slightly ahead of near-deadline

Table 1: Priority inversion for D^3

Fan-in Degree	Low-Var.	Med. Var.	Hi. Var.
20	31.9	26.3	24.1
25	33.2	28.7	24.6
30	35.7	30.8	28.6
35	41.9	33.4	31.5
40	48.6	40.5	33.1

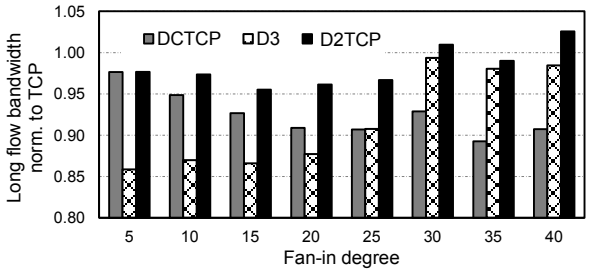

Figure 13: Bandwidth of background flows

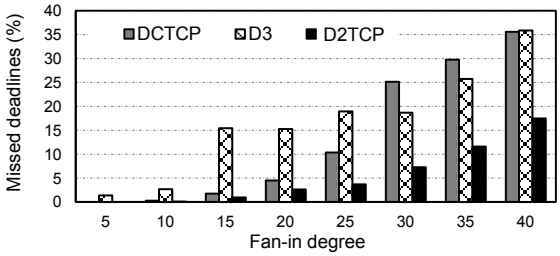

Figure 14: Missed deadlines under background flows

requests. This race condition causes D^3 to frequently invert priorities of congested flows. To confirm these claims, we compute the percentage of requests that are denied while later-deadline requests have been granted. This percentage is a measure of priority inversion in D^3. Table 1 shows this percentage for D^3 under all three variance cases, for various fan-in degrees. From the table, we see that even in a favorable setting (high variance deadlines with a fan-in degree of 20), D^3 incurs priority inversion for nearly 25% of all flows. Also note that the priority inversions worsen both as the fan-in degree increases, and as the variance in deadlines gets tighter causing more burstiness.

4.2.3 Background flows

To test whether long-lived, non-OLDI flows achieve high bandwidth even as short-lived OLDI flows come and go, we replace one leaf-to-parent flow in each OLDI tree with a long-lived background flow. This background flow has an exponential arrival with mean of 300 ms and sends 1 MB of data.

In Figure 13, we show the background flows' bandwidth for DCTCP, D^3, and D^{2}TCP, normalized to that for TCP on the Y axis as we vary the fan-in degree on the X axis. Background flows give up bandwidth to OLDI flows only for the short duration of fan-in-burst-induced congestion under all the schemes. Consequently, all the schemes perform well, achieving 85% or more of the bandwidth achieved by TCP. D^{2}TCP is slightly better than DCTCP which throttles background flows to make room unnecessarily for far-deadline, OLDI flows. Overall, D^{2}TCP achieves 95% or more of the bandwidth achieved by TCP.

To confirm that the background flows do not take bandwidth away from the OLDI flows, we show in Figure 14 the fraction of

Table 2: Long-flow b/w when D^{2}TCP & TCP coexist

Fan-in degree	Long flow bandwidth (Mbps)		
	All TCP	Mix #1	Mix #2
15	90	90	90
20	86	86	86

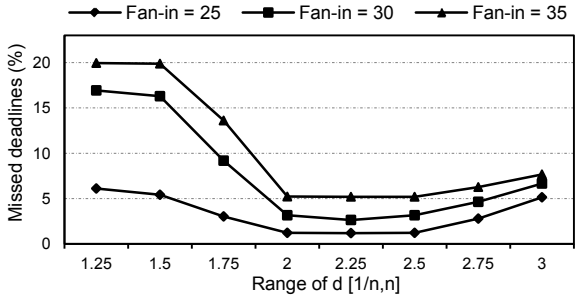

Figure 15: Missed deadlines while varying cap on d

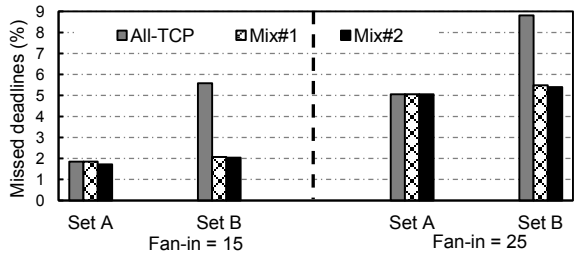

Figure 16: Missed deadlines when D^{2}TCP & TCP coexist

missed deadlines for the OLDI flows in the presence of the background flows. For all the schemes, the fraction remains similar to that in the absence of the background flows (Figure 10).

4.2.4 Varying the cap on d – deadline imminence

Recall from Section 3.2.3, that because d appears in the exponent of the gamma-correction function, extremely low and high values may cause undesirable behavior. Therefore we cap the value of d to be within the range $(n, 1/n)$. In all our experiments we set $n = 2.0$. In this section we evaluate the effects of varying n. In Figure 15, we show D^{2}TCP's percentage of missed deadlines for various fan-ins as we vary n between 1.25 and 3.0. As expected, when n is close to 1.0, D^{2}TCP's behavior matches DCTCP and the fraction of missed deadlines is high. As n increases to 2.0, the fraction drops dramatically but then levels off. At $n = 3.0$ and beyond we see that the fraction starts to increase slowly as the larger n allows near-deadline flows to increasingly ignore congestion feedback.

4.2.5 Coexisting with TCP

To demonstrate that D^{2}TCP can coexist with TCP without hurting bandwidth or deadlines, we use the same production benchmark but use a mix of D^{2}TCP and TCP for the various network traffic. Because we are specifically interested in TCP performance here, we restrict this experiment to fan-in degrees where TCP performance is acceptable in Figure 10 (i.e., missed deadline fraction of 5% or less, and fan-in degree of 15 and 20).

We run three experiments gradually adding D^{2}TCP traffic to a TCP datacenter. Imagine that the set of five OLDI applications in our workload are divided into two sets: set A consist of three OLDIs, and set B consists of the other two OLDIs. We start with

Table 3: Deadlines achieved by D^3 and D^{2}TCP for similar fraction of missed deadlines

Fan-in degree	D3's missed deadlines (%)	D^{2}TCP's missed deadlines (%)	D^{2}TCP's tighter deadline (%)
10	0.71	0.84	55
15	3.61	3.49	45
20	4.7	4.88	35

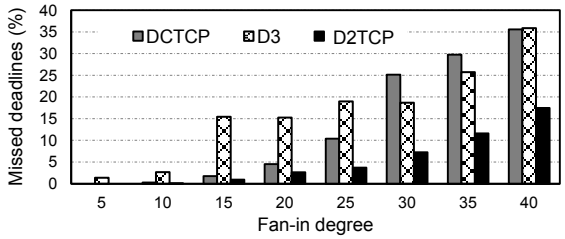

Figure 17: Missed deadlines under tighter deadlines

all five OLDIs and all long flows running on TCP. We first "upgrade" set B to D^{2}TCP, while set A continues to run on TCP. Next, we upgrade the background long-flow traffic to D^{2}TCP as well. In summary, the three setups are:

- All-TCP: 5x OLDIs + long flows; no D^{2}TCP.
- Mix#1 TCP: 3x OLDIs + long flows; D2TCP: 2x OLDIs.
- Mix#2 TCP: 3x OLDIs; D^{2}TCP: 2x OLDI + long flows.

In Figure 16, we show the fraction of missed deadlines (Y axis) for sets A and B in the three runs as we vary the fan-in degree (X axis). We separately analyze long flows below. For set A, comparing All-TCP and Mix#1 shows that TCP's (i.e., set A's) missed-deadline fraction is not worsened by sharing the network with D^{2}TCP traffic (set B in Mix#1). Furthermore, set B sees a reduction in missed deadlines upon migrating from TCP (All-TCP) to D^{2}TCP (Mix#1). Going back to set A, comparing Mix#1 and Mix#2 we see that TCP's (i.e., set A's) missed-deadline fraction is not hurt by background flows upgrading to D^{2}TCP. Similarly, set B's) missed-deadline fraction stays the same between Mix#1 and Mix#2, showing that background flows using D^{2}TCP do not hurt the OLDIs using D^{2}TCP.

In Table 2, we show the long-flow throughput achieved in the three runs. Going from All-TCP to Mix#1, the throughput does not degrade, showing that upgrading some OLDIs to D^{2}TCP does not hurt long TCP flows. Comparing Mix#1 and Mix#2 shows that D^{2}TCP's long-flow throughput is similar to that of TCP.

4.2.6 Tighter deadlines

To show that D^{2}TCP performs well over a range of deadlines, we evaluate D^{2}TCP under tighter deadlines than our default (Section 4.2.1). Because we found that deadlines tightened by 10% or 20% lead to similar behavior, we show results only for the 20% case in Figure 17. As expected, the tighter deadlines here result in more deadlines being missed under all the schemes than those missed in Figure 10. Nevertheless, D^{2}TCP maintains its advantage over DCTCP and D^3 under the tighter deadlines.

While the above results show the fractions of missed deadlines under tighter deadlines, it may be important to determine the inverse (i.e., how much tighter can the deadlines be for a target fraction of missed deadlines). For instance, this question is typically of interest to datacenter operators who would like to maintain the fraction of missed deadlines within an acceptable threshold and wish to know how much the communication deadlines can be tightened to allow more time for computation and improve response quality. In Table 3, we show the tightness of deadlines supported by D^{2}TCP as compared to D^3 for a target fraction of missed deadlines. We limit the study to a reasonable fraction of missed deadlines (i.e., 5%). From the table, we see that D^{2}TCP achieves deadlines that are tighter by 35-55%, which would make sizable room for computation.

5. RELATED WORK

There is an abundance of past work that deals with the subjects of congestion control, network scheduling, and reducing latencies. There are many schemes that build on top of TCP, while others are novel protocols altogether. A comprehensive review of all such work is beyond the scope of this paper, but we summarize some of the most relevant work here.

Earliest Deadline First (EDF) [17] is one of the earliest packet scheduling algorithms and is provably optimal when deadlines are associated with individual packets. When deadlines are associated with flows, however, applying EDF to individual packets as they arrive at the switch is not only suboptimal but can worsen the congestion in the network [26].

In [24], the authors show that having finer grain system clocks allow for a faster response to TCP timeouts, and help in reducing the net latency of TCP flows.

Rate Control Protocol (RCP) [5] can achieve 10-fold improvement in the completion times of small- to medium-sized flows in the Internet, particularly downloads representative of typical web browsing. RCP is provably optimal when minimizing overall completion times is the metric. RCP replaces TCP's slow start phase with an allocation equal to the fair share available at the bottleneck route. Like D^3, RCP requires hardware modification to the routers.

Live multimedia traffic also has a soft-real-time nature, and both proactive bandwidth reservation [6] and reactive [23] [16] schemes exist. TCP-RTM [16] observes that TCP always favors reliability over timeliness, and proposes extensions that improve performance of multimedia applications by allowing minimal amount of packet re-ordering and loss in the TCP stack.

Active Queue Management schemes like RED [8] and E-TCP [11] inject early warnings of congestion to TCP endhosts by randomly dropping packets when switch buffer occupancy is high. Because senders back-off before full on congestion, these schemes allow TCP to operate in the high throughput, fast-retransmit mode, instead of degrading to full back-off.

High-speed TCP [7], CUBIC [22], and XCP [15] all successfully improve the performance of TCP in high bandwidth-delay-product networks. They exploit the large degree of statistical multiplexing present, and also mitigate TCP's drastic reaction to packet losses. XCP shares some common design details with D^3 in that senders request bandwidth via a congestion header, and the switches populate their responses in this header.

Re-feedback [3] addresses the problem of fairness and stability in the Internet when untrusted senders may act selfishly in the face of congestion. Re-feedback incentivizes senders to populate packet headers with honest information about congestion situation so the network may schedule accordingly.

QCN [20] proposes to improve Ethernet performance in datacenters via multibit feedback from the switches to endhosts. By utilizing smarter switches and hardware-based reaction logic in the endhost NICs, QCN dramatically reduces recovery time during congestions, thus improving flow completion times. However, QCN cannot span beyond L2 domains limiting its scope

of application. VCP [27] is another similar scheme that relies on ECN-like feedback via elaborate processing at the switches.

6. CONCLUSION

Online, data-intensive applications (OLDI) in datacenters (e.g., Web search, online retail and advertisement) achieve good user experience by controlling latency using soft-real-time constraints which translate to deadlines for network communication within the applications. Further, OLDI applications typically employ tree-based algorithms which, in the common case, result in fan-in burst of children-to-parent traffic with tight deadlines. Previous work on datacenter network protocols is either deadline-agnostic, or is deadline-aware but suffers under bursts due to race conditions.

We proposed Deadline-Aware DataCenter TCP (D^2TCP) which:

- prioritizes near-deadline flows over far-deadline flows in the presence of fan-in-burst-induced congestion;
- achieves high bandwidth for background flows even as the short-lived OLDI flows come and go;
- requires no changes to the switch hardware; and
- coexists with legacy TCP.

D^2TCP uses a distributed and reactive approach for bandwidth allocation that fundamentally enables D^2TCP's properties. D^2TCP's key mechanism is a novel congestion avoidance algorithm, which uses ECN feedback and deadlines to modulate the congestion window via a gamma-correction function.

Using small-scale real implementation, and at-scale simulations, we showed that D^2TCP

- reduces the fraction of missed deadlines compared to both DCTCP and D^3 by 75% and 50%, respectively;
- achieves nearly as high bandwidth as TCP for background flows without degrading OLDI performance;
- meets deadlines that are 35-55% tighter than those achieved by D^3 for the same reasonable fraction of missed deadlines (i.e., 5%), giving OLDIs more time for actual computation; and
- coexists with TCP flows without degrading their performance.

D^2TCP has significant performance and practical advantages. On the performance side, by reducing the number of missed deadlines, D^2TCP improves OLDI applications' response quality, and hence user experience. Further, by meeting tighter deadlines, D^2TCP allows more time for computation in OLDI applications and thereby further enhances OLDI response quality and user experience. Given that OLDI applications are likely to scale up in size to accommodate ever-growing data on the Web, D^2TCP's tighter deadlines may fundamentally enable this scale-up without degrading OLDI response quality. On the practical side, by requiring no changes to the switch hardware, D^2TCP can be deployed by merely upgrading the TCP and RPC stacks. Our prototype implementation of D^2TCP amounted to only 100 lines of kernel code. Finally, by being able to coexist with TCP, D^2TCP is amenable to incremental deployment, a key requirement for datacenter network protocols in the real world. The growing importance of OLDI applications implies that these significant advantages make D^2TCP an important ingredient for datacenters.

ACKNOWLEDGMENTS

We thank the SIGCOMM reviewers, and our shepherd David Maltz, for their insightful comments which helped us significantly improve the paper. We also thank Sridhar Raman and Abdul Kabbani for their help with real implementations of DCTCP and D^2TCP, and Gwendolyn Voskuilen for reviewing the paper.

REFERENCES

[1] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In Proc. *SIGCOMM*, 2010.

[2] Charles A. Poynton (2003). Digital Video and HDTV: Algorithms and Interfaces. Morgan Kaufmann. pp. 260, 630. ISBN 1558607927.

[3] B. Briscoe et. al. Policing Congestion Response in an Internetwork using Re-feedback. In Proc. *SIGCOMM* 2005.

[4] Datacenter TCP, http://www.stanford.edu/~alizade/Site/DCTCP.htm

[5] Nandita Dukkipati. RCP: Congestion Control to Make Flows Complete Quickly. PhD Thesis, Department of Electrical Engineering, Stanford University, October 2006.

[6] D. Ferrari, A. Banerjea, and H. Zhang. Network support for multimedia: A discussion of the tenet approach. In Proc. *Computer Networks and ISDN Systems*, 1994.

[7] S. Floyd. RFC 3649: HighSpeed TCP for large congestion windows.

[8] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[9] S. Floyd and V. Jacobson. The synchronization of periodic routing messages. *IEEE/ACM Transactions on Networking*, 2(2):122-136, 1994.

[10] R. Griffith, Y. Chen, J. Liu, A. Joseph, and R. Katz. Understanding TCP incast throughput collapse in datacenter networks. In *WREN Workshop*, 2009.

[11] Y. Gu, D. Towsley, C. Hollot, and H. Zhang. Congestion control for small buffer high bandwidth networks. In Proc. *INFOCOM*, 2007.

[12] Urs Hoelzle, Jeffrey Dean, and Luiz André Barroso. Web Search for A Planet: The Architecture of the Google Cluster, In *IEEE Micro Magazine*, April 2003.

[13] T. Hoff. Latency is Everywhere and it Costs You Sales - How to Crush it, July 2009. http://highscalability.com/blog/2009/7/25/latency-iseverywhere-and-it-costs-you-sales-how-to-crush-it.html.

[14] S. Iyer et. al. Analysis of a memory architecture for fast packet buffers. In IEEE *HPSR* Workshop, 2001.

[15] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In Proc. *SIGCOMM*, 2002.

[16] Sam Liang and David Cheriton. TCP-RTM: Using TCP for Real Time Applications. In Proc. *ICNP*, 2002.

[17] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1), 1973.

[18] D. Meisner, C. M. Sadler, L. A. Barroso, W. Weber and T. F. Wenisch. Power Management of Online Data-Intensive Services. In Proc. *ISCA*, June 2011.

[19] The ns-3 discrete-event network simulator. http://www.nsnam.org/

[20] R. Pan, B. Prabhakar, and A. Laxmikantha. QCN: Quantized congestion notification an overview. http://www.ieee802.org/1/files/public/docs2007/au_prabhakar_qcn_overview_geneva.pdf

[21] K. Ramakrishnan, S. Floyd, and D. Black. RFC 3168: The addition of explicit congestion notification (ECN) to IP.

[22] I. R. Sangtae Ha and L. Xu. Cubic: A new TCP-friendly high-speed TCP variant. In Proc. *SIGOPS-OSR*, 2008.

[23] V. Tsaoussidis and C. Zhang. 2002. TCP-Real: receiver-oriented congestion control. *The International Journal of Computer and Telecommunications Networking*. 40(4), 2002.

[24] V. Vasudevan et al. Safe and effective fine-grained TCP retransmissions for datacenter communication. In Proc. *SIGCOMM*, 2009.

[25] C. Wilson, H. Ballani, T. Karagiannis, A. Rowstron. Better Never Than Late: Meeting Deadlines in Datacenter Networks. In Proc. *SIGCOMM*, 2011.

[26] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better never than late: Meeting deadlines in datacenter networks. Technical Report MSR-TR-2011-66, Microsoft Research, May 2011.

[27] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman. One more bit is enough. In Proc. *SIGCOMM*, 2005.