ISSN : 2231-0711

Madhavi Gulhane et al | International Journal of Computer Science Engineering and Technology(IJCSET) | April 2014 | Vol 4, Issue 4,114-120

Data Center Transmission Control Protocol an Efficient Packet Transport for the Commoditized Data Center

Madhavi Gulhane

Computer Science& Engg.

Prof.Ram Meghe Institute of Technology & Research
Badnera-Amravati(Maharashtra)
INDIA

Prof. Dr.Sunil R.Gupta

Computer Science& Engg.

Prof.Ram Meghe Institute of Technology & Research
Badnera-Amravati(Maharashtra)
INDIA

Abstract-Cloud data centers host diverse applications, mixing in the same network a plethora of workflows that require small predictable latency with others requiring large sustained throughput. In this environment, today's state-of-the-art TCP protocol falls short. We present measurements of a 6000 server production cluster and reveal network impairments, such as queue buildup, buffer pressure, and incast, that lead to high application latencies. Using these insights, propose a variant of TCP, DCTCP, for data center networks. DCTCP leverages Explicit Congestion Notification (ECN) and a simple multibit feedback mechanism at the host. We evaluate DCTCP at 1 and 10Gbps speeds, through benchmark experiments and analysis. In the data center, operating with commodity,shallow buffered switches, DCTCP delivers the same or better throughput than TCP, while using 90% less buffer space. Unlike TCP, it also provides hight burst tolerance and low latency for short flows. While TCP's limitations cause our developers to restrict the traffic they send today, using DCTCP enables the applications to handle 10X the current background traffic, without impacting foreground traffic. Further, a 10X increase in foreground traffic does not cause any timeouts, thus largely eliminating incast problems.

1. INTRODUCTION

In recent years, data centers have transformed computing, with large scale consolidation of enterprise IT into data center hubs, and with the emergence of cloud computing service providers like Amazon, Microsoft and Google. A consistent theme in data center design has been to build highly available, highly performant computing and storage infrastructure using low cost, commodity components [18,5]. A corresponding trend has also emerged in data center networks. In particular, low-cost switches are common at the top of the rack, providing up to 48 ports at 1Gbps, at a price point under $2000 — roughly the price of one data center server. Several recent research proposals envision creating economical, easy-to-manage data centers using novel architectures built atop these commodity switches [3,14,17]

Is this vision realistic? The answer depends in large part on how well the commodity switches handle the traffic of real data center applications. In this paper, we focus on soft real-time applications, such as web search, retail, advertising, and recommendation systems that have driven much of the data center construction. We find that these applications generate a diverse mix of short and long flows, and require three things from the data center network: low latency for short flows, high burst tolerance, and high utilization for long flows.

The first two requirements stem from the Partition/Aggregate workflow pattern that many of these applications use. The soft real-time deadlines for end results translate into latency targets for the individual tasks in the workflow. These targets vary from 10ms to 100ms, and tasks not completed before their deadline are cancelled, affecting the final result. Thus, application requirements for low latency directly impact the quality of the result returned and thus revenue. Reducing network latency allows application developers to shift more cycles to the algorithms that improve relevance and end user experience.

The third requirement, high utilization for large flows, stems from the need to continuously update internal data structures of these applications, as the freshness of this data also affects the quality of results. High throughput for long flows that update the data is thus as essential as low latency and burst tolerance.

In this paper, we make two major contributions:

1. We measure and analyze production data center traffic that uses commodity switches (>150TB of compressed data), collected over the course of a month from 6000 servers (x2), extracting application patterns and needs (in particular, low latency needs). Impairments that hurt performance are identified, and linked to properties of the traffic and the switches.
2. We propose a TCP variant, DCTCP, which addresses these impairments to meet the needs of applications (x3). DCTCP uses Explicit Congestion Notification (ECN), a feature already available in modern commodity switches. We evaluate DCTCP at 1 and 10Gbps speeds on ECN-capable commodity switches (x4). We find DCTCP successfully supports 10X increases in application foreground and background traffic in our benchmark studies.

The measurements reveal that the data center's traffic consists of query traffic (2KB to 20KB), delay sensitive short messages (100KB to 1MB), and throughput sensitive long flows (1MB to 100MB). We find that the query traffic

experiences the incast impairment, discussed in [32, 15] in the context of storage networks. However, the data also reveal new impairments unrelated to incast: query and delay-sensitive short messages experience long latencies due to long flows consuming some or all of the available buffer in the switches. Our key learning from these measurements is that to meet the requirements of such a diverse mix of short and long flows, switch buffer occupancies need to be persistently low, while maintaining high throughput for the long flows. DCTCP is designed to do exactly this.

DCTCP combines Explicit Congestion Notification (ECN) with a novel control scheme at the sources. It extracts multi-bit

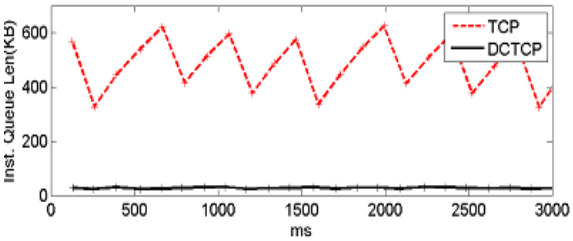


Figure 1: Queue length measured on a Broadcom Triumph switch. Two long flows are launched from distinct 1Gbps ports to a common 1Gbps port. Switch has dynamic memory management enabled, allowing flows to a common receiver to dynamically grab up to 700KB of buffer.

feedback on congestion in the network from the single bit stream of ECN marks. Sources estimate the fraction of marked packets, and use that estimate as a signal for the extent of congestion. This allows DCTCP to operate with very low buffer occupancies while still achieving high throughput. Figure 1 illustrates the effectiveness of DCTCP in achieving full throughput while taking up a very small footprint in the switch packet buffer, as compared to TCP.

While designing DCTCP, a key requirement was that it be implementable with mechanisms in existing hardware — meaning our evaluation can be conducted on physical hardware, and the solution can be deployed to our data centers. The industry reality is that after years of debate and consensus building, a very small number of mechanisms, such as basic RED and ECN, are realized in hardware.

We deliberately concentrate on the data center environment, and on TCP (which makes up 99 :91% of the traffic in our data centers). Our solution confronts the many differences between the data center environment and wide area networks (WANs), where most of the prior work on TCP has focused (x5). For example, we observe empty queue Round Trip Times (RTTs) to be consistently under 250 s. Further, applications have simultaneous needs for extremely high bandwidths and very low latencies, and often there is little statistical multiplexing: a single flow can dominate a particular path.

At the same time, we leverage luxuries not available in the WAN. The data center environment is largely homogeneous and under a single administrative control. Thus, backward compatibility, incremental deployment and fairness to legacy protocols are not major concerns. Connectivity to the external Internet is typically managed through load balancers and application proxies that effectively separate internal traffic from external, so issues of fairness with conventional TCP are irrelevant.

The TCP literature is vast, and there are two large families of congestion control protocols that also attempt to control queue lengths: (i) Implicit delay-based protocols use increases in RTT measurements as a sign of growing queueing delay, and hence of congestion. These protocols rely heavily on accurate RTT measurement, which is susceptible to noise in the very low latency environment of data center

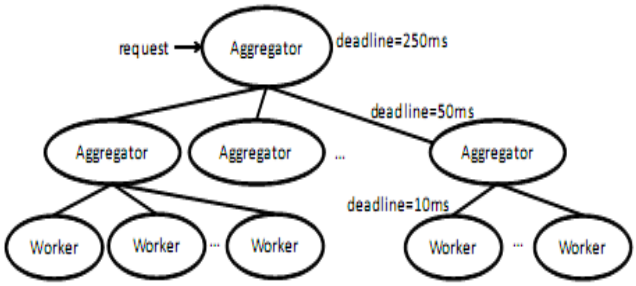


Figure 2: The partition/aggregate design pattern

Small noisy fluctuations of latency become indistinguishable from congestion and the algorithm can over-react. (ii) Active Queue Management (AQM) approaches use explicit feedback from congested switches. The algorithm we propose is in this family. Other approaches for obtaining short latencies include QoS and dividing network traffic into classes. However, QoS requires application developers to agree on how traffic is prioritized in a dynamic multi-application environment.

Having measured and analyzed the traffic in the cluster and associated impairments in depth, we find that DCTCP provides all the benefits we seek. DCTCP requires only 30 lines of code change to TCP, and the setting of a single parameter on the switches.

2. COMMUNICATIONS IN DATA CENTERS

To understand the challenges facing data center transport protocols, we first describe a common application structure, Partition/Aggregate, that motivates why latency is a critical metric in data centers. We then measure the synchronized and bursty traffic patterns that result, and we identify three performance impairments these patterns cause.

2.1 Partition/Aggregate

The Partition/Aggregate design pattern shown in Figure 2 is the foundation of many large scale web applications. Requests from higher layers of the application are broken into pieces and farmed out to workers in lower layers. The responses of these workers are aggregated to produce a result. Web search, social network content composition, and advertisement selection are all based around this application design pattern. For interactive, soft-real-time applications like these, latency is the key metric, with total permissible latency being determined by factors including customer impact studies[21] . After subtracting typical Internet and rendering delays, the "backend" part of the application is

typically allocated between 230-300ms. This limit is called an all-up SLA.

Many applications have a multi-layer partition/aggregate pattern workflow, with lags at one layer delaying the initiation of others. Further, answering a request may require iteratively invoking the pattern, with an aggregator making serial requests to the workers below it to prepare a response. (1 to 4 iterations are typical, though as many as 20 may occur.) For example, in web search, a query might be sent to many aggregators and workers, each responsible for a different part of the index. Based on the replies, an aggre-gator might refine the query and send it out again to improve the relevance of the result. Lagging instances of partition/aggregate can thus add up to threaten the all-up SLAs for queries. Indeed, we found that latencies run close to SLA targets, as developers exploit all of the available time budget to compute the best result possible.

To prevent the all-up SLA from being violated, worker nodes are typically assigned tight deadlines, usually on the order of 10-100ms. When a node misses its deadline, the computation continues without that response, lowering the quality of the result. Further, high percentiles for worker latencies matter. For example, high latencies at the 99 :9 percentile mean lower quality results or long lags (or both) for at least 1 in 1000 responses, potentially impacting large numbers of users who then may not come back. Therefore, percentiles are typically tracked to 99 :9 percentiles, and deadlines are associated with high percentiles. Figure 8 shows a screen shot from a production monitoring tool, focusing on a 5ms issue.

With such tight deadlines, network delays within the data center play a significant role in application design. Many applications find it so hard to meet these deadlines using stateof-the-art TCP that they often take on enormous amount of complexity to get around it. For example, our application reduces the amount of data each worker sends and employs jitter. Facebook, reportedly, has gone to the extent of developing their own UDP-based congestion control [29]

2.2 Workload Characterization

We next measure the attributes of workloads in three production clusters related to web search and other services. The measurements serve to illuminate the nature of data center traffic, and they provide the basis for understanding why TCP behaves poorly and for the creation of benchmarks for evaluating DCTCP.

We instrumented a total of over 6000 servers in over 150 racks. The three clusters support soft real-time query traffic, integrated with urgent short message traffic that coordinates the activities in the cluster and continuous background traffic that ingests and organizes the massive data needed to sustain the quality of the query responses. We use these terms for ease of explanation and for analysis, the developers do not separate flows in simple sets of classes. The instrumentation passively collects socket level logs, selected packetlevel logs, and app-level logs describing latencies – a total of about 150TB of compressed data over the course of a month.

Each rack in the clusters holds 44 servers. Each server connects to a Top of Rack switch (ToR) via 1 Gbps Ethernet. The ToRs are shallow buffered, shared-memory switches; each with 4 MB of buffer shared among 48 1 Gbps ports and two 10Gbps ports.

Query Traffic. Query traffic in the clusters follows the Partition/Aggregate pattern. The query traffic consists of very short, latency-critical flows, following a relatively simple pattern, with a high-level aggregator (HLA) partitioning queries to a large number of mid-level aggregators (MLAs) that in turn partition each query over the 43 other servers in the same rack as the mid-level aggregator. Servers act as both MLAs and workers, so each server will be acting as an aggregator for some queries at the same time it is acting as a worker for other queries. Figure 3(a) shows the CDF of time between arrivals of queries at mid-level aggregators. The size of the query flows is extremely regular, with queries from MLAs to workers being 1.6 KB and responses from workers to MLAs being 1.6 to 2 KB.

Background Traffic. Concurrent with the query traffic is a complex mix of background traffic, consisting of both large and small flows. Figure 4 presents the PDF of background flow size, illustrating how most background flows are small, but most of the bytes in background traffic are part of large flows. Key among background flows are large, 5KB to 50MB, update flows that copy fresh data to the workers and time-sensitive short message flows, 50KB to 1MB in size, that update control state on the workers. Figure 3(b) shows the time between arrival of new background flows. The interarrival time between background flows reflects the superposition and diversity of the many different services supporting the application: (1) the variance in interarrival time is very high, with a very heavy tail; (2) embedded spikes occur, for example the 0ms inter-arrivals that explain the CDF hugging the y-axis up to the 50 percentile; and (3) relatively large numbers of outgoing flows occur periodically, resulting from workers periodically polling a number of peers looking for updated files.

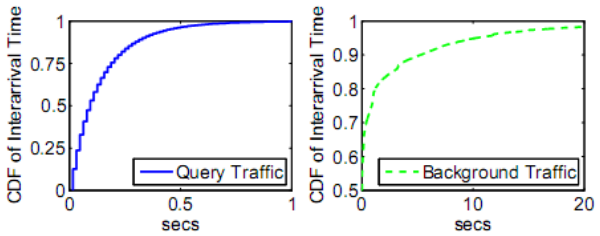


Figure 3: Time between arrival of new work for the Aggregator (queries) and between background flows between servers (update and short message).

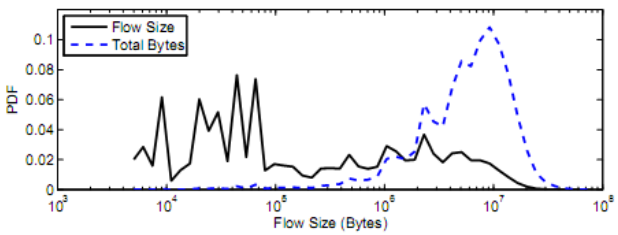


Figure 4: PDF of flow size distribution for background traffic. PDF of Total Bytes shows probability a randomly selected byte would come from a flow of given size.

Flow Concurrency and Size. Figure 5 presents the CDF of the number of flows a MLA or worker node participates in concurrently (defined as the number of flows active during a 50 ms window). When all flows are considered, the median

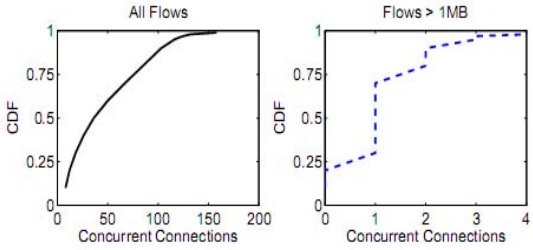


Figure 5: Distribution of number of concurrent connections.

number of concurrent flows is 36, which results from the breadth of the Partition/Aggregate traffic pattern in which each server talks to 43 other servers. The 99.99th percentile is over 1,600, and there is one server with a median of 1,200 connections.

When only large flows (>1MB) are considered, the degree of statistical multiplexing is very low — the median number of concurrent large flows is 1, and the 75th percentile is 2. Yet, these flows are large enough that they last several RTTs, and can consume significant buffer space by causing queue buildup.

In summary, throughput-sensitive large flows, delay sensitive short flows and bursty query traffic, co-exist in a data center network. In the next section, we will see how TCP fails to satisfy the performance requirements of these flows.

3. THE DCTCP ALGORITHM

The main goal of DCTCP is to achieve high burst tolerance, low latency, and high throughput, with commodity shallow buffered switches. To this end, DCTCP is designed to operate with very small queue occupancies, without loss of throughput.

DCTCP achieves these goals primarily by reacting to congestion in proportion to the extent of congestion. DCTCP uses a very simple marking scheme at switches that sets the Congestion Experienced (CE) codepoint of packets as soon

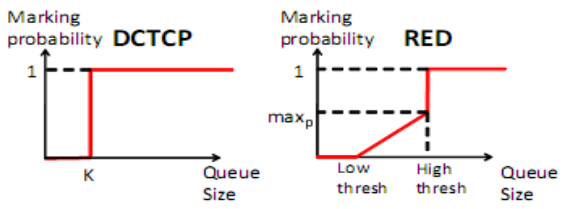


Figure 6: DCTCP's AQM scheme is a variant of RED: Low and High marking thresholds are equal, and marking is based on the instantaneous queue length.

as the buffer occupancy exceeds a fixed small threshold. The DCTCP source reacts by reducing the window by a factor that depends on the fraction of marked packets: the larger the fraction, the bigger the decrease factor.

It is important to note that the key contribution here is not the control law itself. It is the act of deriving multi-bit feedback from the information present in the single-bit sequence of marks. Other control laws that act upon this information can be derived as well. Since DCTCP requires the network to provide only single-bit feedback, we are able to re-use much of the ECN machinery that is already available in modern TCP stacks and switches.

We note that the idea of reacting in proportion to the extent of congestion is also used by delay-based congestion algorithms [6,31]Indeed, one can view path delay information as implicit multi-bit feedback. However, at very high data rates and with low-latency network fabrics, sensing the queue buildup in shallow-buffered switches can be extremely noisy. For example, a 10 packet backlog constitutes 120s of queuing delay at 1 Gbps, and only 12 sat 10 Gbps. The accurate measurement of such small increases in queueing delay is a daunting task for today's servers.

The need for reacting in proportion to the extent of congestion is especially acute in the absence of large-scale statistical multiplexing. Standard TCP cuts its window size by a factor of 2 when it receives ECN notification. In effect, TCP reacts to presence of congestion, not to its extent Dropping the

window in half causes a large mismatch between the input rate to the link and the available capacity. In the high speed data center environment where only a small number of flows share the buffer, this leads to buffer underflows and loss of throughput.

3.1 Algorithm

The DCTCP algorithm has three main components:

(1) Simple Marking at the Switch: DCTCP employs a very simple active queue management scheme, shown in Figure 10. There is only a single parameter, the marking threshold, K. An arriving packet is marked with the CE codepoint if the queue occupancy is greater than Kupon its arrival. Otherwise, it is not marked. The design of the DCTCP marking scheme is motivated by the need to minimize queue buildup. DCTCP aggressively marks packets when a queue overshoot is sensed. This allows sources to be notified of the queue overshoot as fast as possible.

Figure 6 shows how the RED marking scheme (implemented by most modern switches) can be re-purposed for DCTCP. We simply need to set both the low and high thresholds to K, and mark

based on instantaneous, instead of average queue length.

(2) ECN-Echo at the Receiver: The only difference between a DCTCP receiver and a TCP receiver is the way information in the CE codepoints is conveyed back to the sender. RFC 3168 states that a receiver sets the ECN-Echo flag in a series of ACK packets until it receives confirmation from the sender (through the CWR flag) that the congestion notification has been received. A DCTCP receiver, however, tries to accurately convey the exact sequence of marked packets back to the sender. The simplest way to do this is to ACK every packet, setting the ECN-Echo flag if and only if the packet has a marked CE codepoint

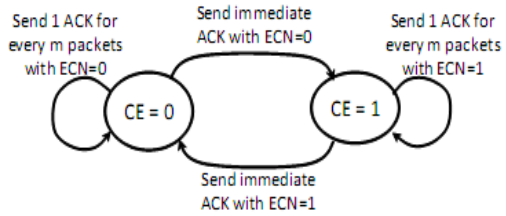


Figure 7 : Two state ACK generation state machine.

However, using Delayed ACKs is important for a variety of reasons, including reducing the load on the data sender. To use delayed ACKs (one cumulative ACK for every m consecutively received packets the DCTCP receiver uses the trivial two state state-machine shown in Figure 11 to determine whether to set ECN-Echo bit. The states correspond to whether the last received packet was marked with the CE codepoint or not. Since the sender knows how many packets each ACK covers, it can exactly reconstruct the runs of marks seen by the receiver.

(3) Controller at the Sender: The sender maintains a running estimate of the fraction of packets that are marked, called , which is updated once for every window of data (roughly one RTT) as follows:

(1 g) + g F (1) where Fis the fraction of packets that were marked in the

last window of data, and 0 <g<1 is the weight given to new samples against the past in the estimation of .

Note that is a real number between 0 and 1. Given that the sender receives marks for every packet when the queue length is higher than Kand does not receive any marks when the queue length is below K, the formula shown above implies that is the probability that the queue is greater than K. Essentially, close to 0 indicates low, and close to 1 indicates high levels of congestion.

Prior work on congestion control in the small buffer regime have argued that at high line rates, queue size fluctuations become so fast that you cannot control the queue size, only its distribution[25,20]. The physical significance of is well aligned with this intuition: it represents a single point of the queue size distribution at the bottleneck link.

The only difference between a DCTCP sender and a TCP sender is in how each reacts to receiving an ACK with the ECN-Echo flag set. All other features of TCP such as slow start, additive increase in congestion avoidance, or recovery from packet lost are left unchanged. While TCP always cuts its window size by a factor of 2 in response a marked ACK, DCTCP uses to cut its window size as follows

$$cwnd \leftarrow cwnd \times (1 - \alpha/2)$$

Thus, when is near 0 (low congestion), the window is slightly reduced. In other words, DCTCP senders start gently reducing their window as soon as the queue exceeds K. This is how DCTCP maintains low queue length, while still ensuring high throughput. When congestion is high (α=1), DCTCP cuts its window in half, just like TCP.

4. BENCHMARK TRAFFIC

In the sections that follow, we evaluate how DCTCP would perform under the traffic patterns found in production clusters (x2.2). For this test, we use 45 servers connected to a Triumph top of rack switch by 1Gbps links. An additional server is connected to a 10Gbps port of the Triumph to act as a stand-in for the rest of the data center, and all inter-rack traffic is directed to/from this machine. This aligns with the actual data center, where each rack connects to the aggregation switch with a 10Gbps link.

We generate all three types of traffic found in the cluster: query, short-message, and background. Query traffic is created following the Partition/Aggregate structure of the real application by having each server draw from the interarrival time distribution and send a query to all other servers in the rack, each of which then send back a 2KB response ,

(45 2KB 100KB total response size). For the shortmessage and background traffic, each server draws independently from the interarrival time and the flow size distributions, choosing an endpoint so the ratio of inter-rack to intra-rack flows is the same as measured in the cluster.

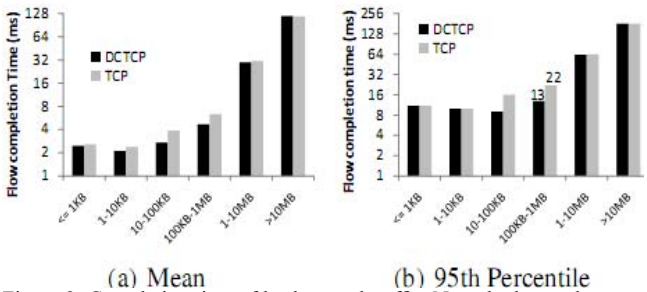


Figure 8: Completion time of background traffic. Note the log scale on the Y axis.

carry out these experiments using TCP and DCTCP, with RTO set to 10ms in both. For DCTCP experiments, K was set to 20 on 1Gbps links and to 65 on the 10Gbps link. Dynamic buffer allocation was used in all cases. We

generate traffic for 10 minutes, comprising over 200,000 background flows and over 188,000 queries.

Both query and short-message flows are time critical, their metric of interest is completion time. The RTT (i.e. queue length) and timeouts affect this delay the most. For large flows in the background traffic (e.g., updates), the throughput of the network is the main consideration.

Figure 8 shows the mean and 95th percentile of completion delay for background traffic, classified by flow sizes. The 90% confidence intervals of the mean are too tight to be shown. Short-messages benefit significantly from DCTCP, as flows from 100KB-1MB see a 3ms/message benefit at the mean and a 9ms benefit at 95th percentile. The background traffic did not suffer any timeouts with either protocol in this experiment. Thus, the lower latency for short-messages is due to DCTCP's amelioration of queue buildup.

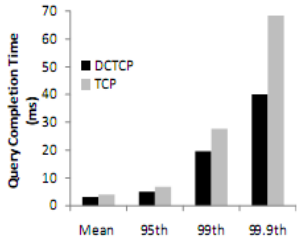


Figure 9 : Completion time Query traffic

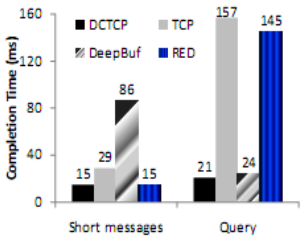


Figure 10 :95th percentile completion time 10x background & 10x query

Figure 9 shows query completion time statistics. DCTCP performs better than TCP, especially at the tail of the distribution. The reason is a combination of timeouts and high queueing delay. With TCP, 1.15% of the queries suffer from timeout(s). No queries suffer from timeouts with DCTCP.

Scaled traffic: The previous benchmark shows how DCTCP performs on today's workloads. However, as explained in x2.3, the traffic parameters we measured reflect extensive optimization conducted by the developers to get the existing system into the tight SLA bounds on response time. For example, they restrict the size of query responses and update frequency, thereby trading off response quality for response latency. This naturally leads to a series of "what if" questions: how would DCTCP perform if query response sizes were larger? Or how would DCTCP perform if background traffic characteristics were different? We explore these questions by scaling the traffic in the benchmark, while keeping the structure unchanged.

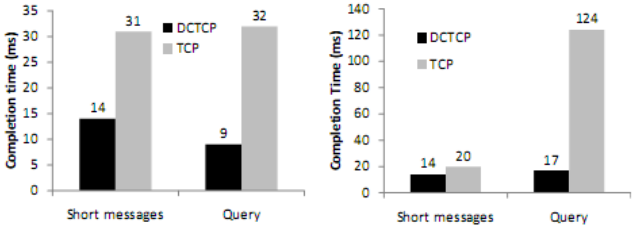
We begin by asking if using DCTCP instead of TCP would allow a 10X increase in both query response size and background flow size without sacrificing performance. We use the same testbed as before. We generate traffic using the benchmark, except we increase the size of update flows larger than 1MB by a factor of 10 (most bytes are in these flows, so this effectively increases the volume of background traffic by a factor of 10). Similarly, we generate queries as before, except that the total size of each response is 1MB (with 44 servers, each individual response is just under 25KB). We conduct the experiment for both TCP and DCTCP.

Additionally, for TCP, we tried two ways of fixing its performance. First, we replaced the shallow-buffered Triumph switch with the deep-buffered CAT4948 switch.

Second, instead of drop tail queues, we used RED with ECN.

It was as difficult to tune RED parameters at 1Gbps as it was previously at 10Gbps: after experimentation, we found that setting min th= 20 ; maxth= 60 and using for the remaining parameters gave the best performance.

Figure 10 shows the 95th percentile of response times for the short messages (100KB-1MB) and the query traffic (mean and other percentiles are qualitatively similar). The results show DCTCP performs significantly better than TCP for both update and query traffic. The 95th percentile of completion time for short-message traffic improves by 14ms, while query traffic improves by 136ms. With TCP, over 92% of the queries suffer from timeouts, while only 0.3% suffer from timeouts with DCTCP.



(a) 10x background size (b) 10x query size

Figure 10: 95th percentile of completion time

In fact, short message completion time for DCTCP is essentially unchanged from baseline (Figure 8(b)) and, even at 10X larger size, only 0.3% of queries experience timeouts under DCTCP: in contrast TCP suffered 1.15% timeouts for the baseline.

Thus, DCTCP can handle substantially more traffic without any adverse impact on performance.

Deep buffered switches have been suggested as a fix for TCP's incast problem, and we see this is true: on the CAT4948 less than 1% of the queries suffer from timeout with TCP, and the completion time is comparable to DCTCP. However, if deep buffers are used, the short-message traffic is penalized: their completion times are over 80ms, which is substantially higher than TCP without deep buffers (DCTCP is even better)

The reason is that deep buffers cause queue buildup.

We see that RED is not a solution to TCP's problems either: while RED improves performance of short transfers by keeping average queue length low, the high variability (see Figure 16) of the queue length leads to poor performance

for the query traffic (95% of queries experience a timeout). Another possible factor is that RED marks packets based on average queue length, and is thus slower to react to bursts of traffic caused by query incast.

These results make three key points: First, if our data center used DCTCP it could handle 10X larger query responses and 10X larger background flows while performing better than it does with TCP today. Second, while using deep buffered switches (without ECN) improves performance of query traffic, it makes performance of short transfers worse, due to queue build up. Third, while RED improves performance of short transfers, it does not improve the performance of query traffic, due to queue length variability.

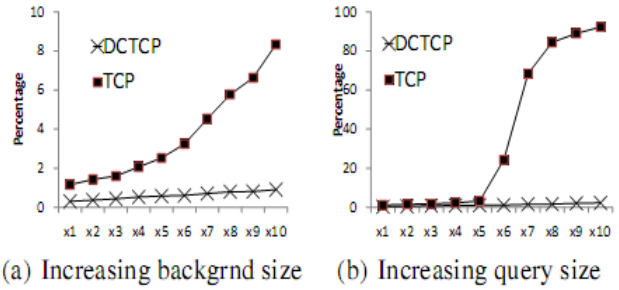


Figure 12: Fraction of queries that suffer at least one timeout

Benchmark variations:

The intensity of our benchmark traffic can be varied either by increasing the arrival rate of the flows or by increasing their sizes. We explored both dimensions, but the results are similar, so we report primarily on increases in flow sizes. Specifically, we report on the two corners: scaling the background traffic while holding query traffic to the original benchmark size, and vice versa.

Figure 11(a) shows that increasing the size of background traffic hurts the performance of both short messages and query traffic. As big flows cause both queue buildup delays and buffer pressure, which DCTCP mitigates. Figure 12(a) shows how increasing background traffic causes buffer pressure that causes query traffic timeouts, but the impact on TCP is greater than DCTCP.

Figure 11(b) shows that increasing the size of the query responses by a factor of 10 severely degrades the latency of query traffic, with TCP. However, DCTCP handles the increased traffic without significant impact on the performance (compare Fig. 11(b) to Fig.9). The reason is DCTCP reducing incast timeouts: Figure 12(b) shows how for TCP the fraction of queries that suffer timeouts grows quickly with response size. After the response size exceeds 800KB, almost all queries suffer from timeout

5. CONCULSION

In this paper, we provided detailed traffic measurements from a 6000 server data center cluster, running production soft real time applications, and linked these to the behavior of the commodity switches in use in the cluster. We found that to meet the needs of the observed diverse mix of short and long flows, switch buffer occupancies need to be persistently low, while maintaining high throughput for the long flows. A wide set of detailed experiments at 1 and 10Gbps speeds showed that DCTCP does exactly this. DCTCP relies on Explicit Congestion Notification (ECN), a feature now available on commodity switches rolling out in 2010. DCTCP succeeds through use of the multi-bit feedback derived from the series of ECN marks, allowing it to react early to congestion. We recommend enabling ECN and deploying DCTCP, to bring us one step closer to economical, high performance data center networks.

REFERENCE

[1] P. Agarwal, B. Kwan, and L. Ashvin. Flexible buffer allocation entities for traffic aggregate containment. US Patent 20090207848, August 2009.
[2] A. Aggarwal, S. Savage, and T. Anderson. Understanding the performance of TCP pacing. In IEEE Infocom 2000, 2000.
[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In SIGCOMM, 2008.
[4] G. Appenzeller et al. Sizing router buffers. In SIGCOMM, 2004.
[5] L. A. Barroso and U. Holzle. The Datacenter as a Computer - an introduction to the design of warehouse-scale machines. Morgan & Claypool, 2009.
[6] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques forcongestion detection and avoidance. In SIGCOMM, 1994.
[7] S.Floyd. RED: Discussions of setting parameters.http://www.icir.org/floyd/REDparameters.txt.
[8] S. Floyd. Rfc 3649: Highspeed TCP for large congestion windows.
[9] S. Floyd et al. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. Technical report, ACIRI, 2001.
[10] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. IEEE/ACM Trans. Netw., 1(4):397–413, 1993.
[11] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. IEEE/ACM ToN, 1993.
[12] S. Floyd and V. Jacobson. The synchronization of periodic routing messages.IEEE/ACM ToN, 1994.
[13] S. Gorinsky, A. Kantawala, and J. Turner. Link buffer sizing: A new look at the old problem. In IEEE ISCC '05, 2005.
[14] A. Greenberg et al. Vl2: A scalable and flexible data center network. In SIGCOMM, 2009.
[15] R. Griffith, Y. Chen, J. Liu, A. Joseph, and R. Katz. Understanding TCP incast throughput collapse in datacenter networks. In WREN Workshop, 2009.
[16] Y. Gu, D. Towsley, C. Hollot, and H. Zhang. Congestion control for small buffer high bandwidth networks. In INFOCOM, 2007.
[17] C. Guo et al. Bcube: High performance, server-centric network architecture for data centers. In SIGCOMM, 2009.
[18] J. Hamilton. On designing and deploying internet-scale services. In USENIX LISA, 2007.
[19] C. V. Hollot, V. Misra, D. Towsley, and W.-B. Gong. On designing

improved controllers for AQM routers supporting TCP flows. In Infocom, April 2001.
[20] F. Kelly et al. Stability and fairness of explicit congestion control with small buffers. SIGCOMM Comput. Commun. Rev., 38(3):51–62, 2008.
[21] R. Kohavi et al. Practical Guide to Controlled Experiments on the Web: Listen to Your Customers not to the HiPPO. KDD, 2007.
[22] D. Leith, R. Shorten, and G. McCullagh. Experimental evaluation of cubic-TCP. In Proc. Protocols for Fast Long Distance Networks 2007, 2007.
[23] Y.-T. Li, D. Leith, and R. N. Shorten. Experimental evaluation of TCP protocols for high-speed networks. IEEE/ACM Trans. Netw., 15(5):1109–1122, 2007.
[24] R. Z.-S. Nandita Dukkipati, Masayoshi Kobayashi and N. McKeown. Processor sharing flows in the internet. In IWQOS, 2006.
[25] G. Raina, D. Towsley, and D. Wischik. Part ii: control theory for buffer sizing.SIGCOMM Comput. Commun. Rev., 35(3):79–82, 2005.
[26] K. Ramakrishnan, S. Floyd, and D. Black. Rfc 3168: The addition of explicit congestion notification (ECN) to IP.
[27]K. K. Ramakrishnan and R. Jain. A binary feedback scheme for congestion avoidance in computer networks. ACM TRANSACTIONS ON COMPUTER SYSTEMS, 8:158–181, 1990.
[28] B. P. A. L. Rong Pan. QCN: Quantized congestion notification an overview.
[29] J. Rothschild. High performance at massive scale: Lessons learned at facebook. mms://video-jsoe.ucsd.edu/calit2/JeffRothschildFacebook.wmv.
[30] I. R. Sangtae Ha and L. Xu. Cubic: A new TCP-friendly high-speed TCP variant. SIGOPS-OSR, July 2008.
[31] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In INFOCOM, 2006.
[32] V. Vasudevan et al. Safe and effective fine-grained TCP
[33]D. X. Wei, C. Jin, S. H. Low, and S. Hegde. Fast TCP: motivation, architecture, algorithms, performance. TON, Dec. 2006.