A blog on Data Center TCP (DCTCP).

Shubham Pramod Singh.

[singhsp@uwm.edu](mailto:singhsp@uwm.edu)

The University of Wisconsin Milwaukee.

Instructor: Michael Denzien.

**Abstract**

A TCP congestion control scheme for data-center traffic. DCTCP extends the Explicit Congestion Notification (ECN) processing to estimate the fraction of bytes that encounter congestion rather than simply detecting that some congestion has occurred. DCTCP then scales the TCP congestion window based on this estimate.

This technique uses shallow buffer switching to produce high burst tolerance, low latency, and high throughput. This document also examines the lack of a negotiation mechanism between the sender and recipient, deployment challenges linked to the coexistence of DCTCP and traditional TCP, and potential mitigations.

1. **Introduction**

Large data centers must have many network switches to connect their numerous servers. Therefore, by utilizing inexpensive switches, a data center can significantly lower its capital investment. However, these inexpensive switches frequently have low queue capacity, making them more prone to congestion-related packet loss.

In a data center, network traffic frequently consists of both short- and long-term flows, with the short-term flows requiring low latencies and the long-term flows requiring large throughputs. Bursts of traffic from multiple servers to a single server at once also happen in data centers. For instance, this traffic pattern results naturally from the MapReduce workload: the worker nodes finish roughly at the same time and respond to the master node simultaneously.

These elements place a switch's queue occupancy under various competing demands:

The queue needs to be short enough so that short flows are not subjected to significant latency.

The queue must hold enough data in buffers for long flows to fill the path's capacity.

Incast bursts must be handled by the queue with little packet loss.

1. **Background**

Build highly available, highly performant computing and storage infrastructure using low-cost, commodity components. Authors focus on soft real-time applications, supporting web search, retail, advertising, and recommendation systems that have driven much data center construction. These applications generate a diverse mix of short and long flows. In this paper, the authors make two major contributions measuring and analyzing production traffic, extracting application patterns and needs, and identifying impairments that hurt performance propose Data Center TCP (DCTCP), which addresses these impairments to meet the needs of applications. The authors first describe a common application structure in order to understand the challenges facing data center transport protocols. Authors measure the synchronized and bursty traffic patterns that result from these application structures. Identify three performance impairments these patterns cause.

1. **Design**

3.1 **Partition/Aggregate**:

Case study: Microsoft Bing Measurements passively gathers socket-level logs, chosen packet-level logs, and app-level logs characterizing latencies from a production cluster of 6000 servers, over 150TB of compressed data in a single month, and 99.91% of traffic in the Microsoft data center is TCP traffic. Web search, social network creation, ad selection, and other large-scale web applications are all built on this basis. For instance, Facebook. Partition/Aggregate uses Multiget, which uses Web servers as aggregators and Memcached servers as workers. Consider Workloads that include Aggregate/Partition within the size of [2KB - 20KB] are time-sensitive. Short messages (Coordination, Control status) are delay-sensitive and range from 50 KB to 1 MB. Large flows (from 1 MB to 50 MB) (Data update) are sensitive to throughput.

* 1. **Incast**:

When there are too many storage servers transmitting data to a client for an Ethernet switch to handle, a catastrophic TCP throughput collapse known as TCP Incast takes place. When using a clustered file system, for instance, a client program stripes data blocks across multiple storage servers, only requesting the next block when each server has finished responding with its piece (Figure 1). This synchronized request demand may cause packets to overflow the switch's client port buffers, causing numerous losses. When there is a significant amount of packet loss, TCP may time out for a minimum of 200 milliseconds, as specified by the TCP minimum retransmission timeout (RTOmin). Incast degrades both performance and, more importantly, user experience. A response that incurs incast will almost certainly miss the aggregator deadline and be left out of the final results.

Diagram, engineering drawing, schematic

Description automatically generated

Figure: 1.1

* Incast degrades both performance and, more importantly, user experience.
* A response that incurs incast will almost certainly miss the aggregator deadline and be left out of the final results.

Diagram

Description automatically generated

Figure 1.2: Shows a straightforward cluster-based storage architecture where one client synchronized reads data from several servers.

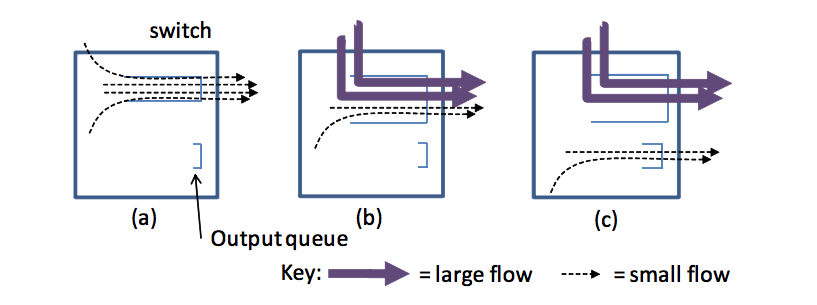
* 1. **Queue Buildup**:

There is a queue accumulation impairment: the short flows experience higher latency even when no packets are lost since they are in the queue after packets from the large flows. This traffic pattern occurs frequently because each worker in the cluster handles both query traffic and background traffic (huge flows required to update the workers' data structures). When long and short flows traverse the same queue, two impairments occur packet loss on the short flows can cause incast problems, as described before there is a *queue buildup* impairment: even when no packets are lost, the short flows experience in- creased latency as they are in the queue behind packets from the large flows

Measurements in Bing cluster

**For 90% packets: RTT < 1ms**

**For 10% packets: 1ms < RTT < 15ms**

Figure 2: Three ways in which flows interact on a multi-ported switch that result in performance problems.

* 1. **Buffer pressure:**

Buffer space is a shared resource, the queue build-up reduces the amount of buffer space available to absorb bursts of traffic from Partition/Aggregate traffic result is packet loss and timeouts, as in incast, but without requiring synchronized flows. On their interfaces, the long, greedy TCP flows create queues. The queue buildup decreases the amount of buffer space available to absorb bursts of traffic from Partition/Aggregate traffic because buffer space is a shared resource (shallow buffered switches). Data Centre transport requirements are Low latency for shorter flows, High burst tolerance (manages incast issues brought on by partition/aggregation), and High flow rates with extended throughput. Switch buffer occupancies must consistently be low while long flows require high throughput.

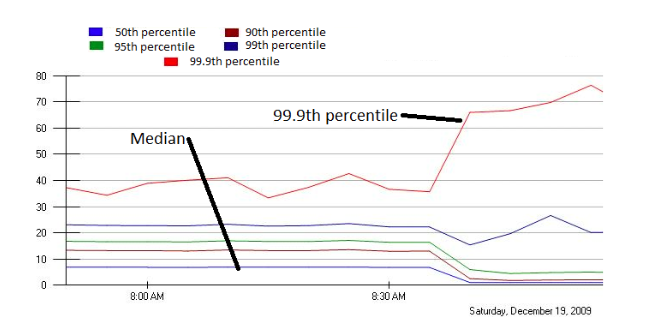


Figure 3: Shows the percentiles of response times for a production application with incast traffic. Before 8:30 am, when jittering was turned off, forwarded requests were intentionally delayed across a 10ms timeframe. The 99.9th percentile doubles, while the 95th and lower percentiles decrease by a factor of 10.

* 1. **Switches**:

Shared memory *switches* that aim to exploit statistical multiplexing gain through the use of logically common packet buffers available to all switch ports.

If the interface has hit its maximum memory allocation or the shared pool itself is depleted, then the packet is dropped.

Building large multi-ported memories are very expensive, so most cheap switches are shallowly buffered, with packet buffer being the scarcest resource.

1. **DCTCP Algorithm**:

The DCTCP algorithm consists of these three elements: Congestion is detected by the switches (or other network intermediaries), which then sets the Congestion Encountered (CE) codepoint in the IP header. Using the ECN-Echo (ECE) signal in the TCP header, the receiver relays the congestion information back to the sender.  In response, the sender adjusts the TCP congestion window (cwnd) by computing a congestion estimate. With widely available shallow buffered switches, DCTCP aims to provide high burst tolerance, low latency, and high throughput. DCTCP is made to function with low queue occupancies without throughput loss. As soon as the buffer occupancy rises above a predetermined modest threshold, DCTCP uses a straightforward marking mechanism at switches to set the Congestion Experienced (CE) codepoint of packets.

4.1 **Simple Marking at the Switch:**

In response, the DCTCP source shrinks the window by a factor that changes according to the percentage of marked packets: the higher the percentage, the larger the marking threshold, and K, the only parameter, is present. When a packet arrives and the queue occupancy is more than K, it is indicated with the CE codepoint. If not, it is unmarked. With this system, sources are promptly informed of queue overshoots.

4.2 **ECN-Echo at the Receiver:**

The DCTCP receiver uses the trivial two state state-machine shown in Figure 4 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.

Diagram

Description automatically generated

Figure 4: Two-state ACK generation State Machine.

4.3 **Controller at the Sender**:

* a <- (1-g)×a+g×F (1.1)
  + F is the *fraction* of packets that were marked in the last window of data
  + 0 < g < 1 is the weight given to new samples against the past in the estimation of **a**
  + **a** estimates the probability that the queue size is greater than K
  + **a** close to 0 indicates low, and a close to 1 indicates high levels of congestion

cwnd <- cwnd × (1-a/2) (1.2)

4.4 **Advantages of the DCTCP Algorithm**:

**Queue Buildup**:

DCTCP senders begin responding when the queue length on an interface exceeds K.

Decreases queueing delays on busy switch ports, which lessens the effect of long flows on the time it takes for minor flows to complete.

**Buffer pressure**:

The length of the line in a crowded port does not increase much.

**Incast**:

Drops are frequently caused by bursts in successive RTTs.

The amount of follow-up bursts is controlled by the marks that DCTCP sources get during the first one or two RTTs.

4.5 **Algorithm analysis**:

* Consider N infinitely long-lived flows with identical round-trip times RTT, sharing a single bottleneck link of capacity C.
* N flows are synchronized
* The queue size at time t is given by
  + Q(t) = NW(t) – C×RTT (1.3)

W (t) is the window size of a single source

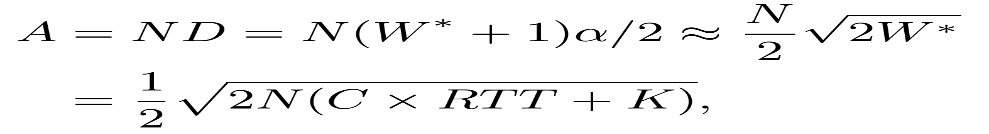
* S (W1, W2) denotes the number of packets sent by the sender, while its window size increases from W1 to W2 > W1.
* Screen Shot 2014-10-27 at 11.42.38 pm.png (1.4)

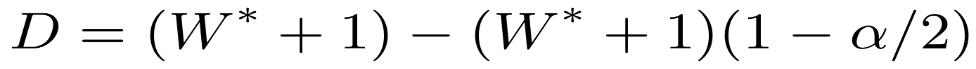
Screen Shot 2014-10-27 at 11.42.51 pm.png (1.5)

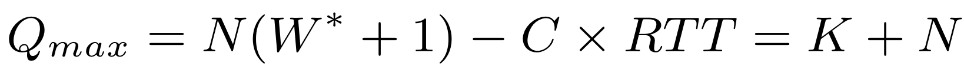
Screen Shot 2014-10-27 at 11.42.59 pm.png(1.6)

D is the amplitude of oscillation in the window size of a single flow

 (1.6)

(1.7)

 (1.8)

(1.9)

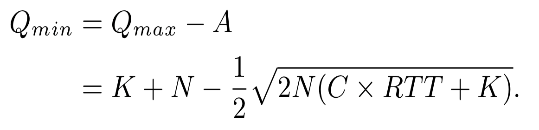
* Authors have evaluated the accuracy of the above results using NS-2 simulations in a variety of scenarios
  + For large N, as in the N = 40 case, de-synchronization of the flows leads to smaller queue variations than predicted by the analysis

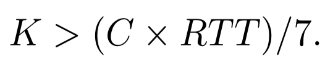
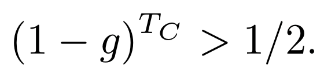
Graphical user interface

Description automatically generated with medium confidence

Figure 5: Comparison between queue size process with NS2 simulations where k = 40 and g =1/16.

* Marking Threshold: The minimum value of the queue occupancy sawtooth is given by:

(2.0)

* To find a lower bound on K, we minimize it over N, and choose K so that this minimum is larger than zero
* (2.1)
* Estimation Gain: g must be chosen small enough to ensure the exponential moving average “spans” at least one congestion event
* (2.2)
* Plugging in expression for Tc with the worst-case value N = 1
* (2.3)

1. **Review**

5.1 **AQM is not enough**:

Red and PI: Active Queue Management (AQM)

AQM does not alter the congestion control method of TCP.

Work poorly when traffic is sporadic and statistical multiplexing is low.

Must compromise between throughput and latency.

With few flows (5), PI experiences queue underflows and a loss of utilization, while with numerous flows (20), queue oscillations worsen and cause latency for time-critical flows.

Due to the significant level of queue length unpredictability in RED, query traffic times out.

Both high throughput and low delay are attained using DCTCP.

* 1. **Convergence and Synchronization**:

DC RTT: 0.1ms, which is negligible in comparison to Internet RTT. DCTCP only takes 2 or 3 times longer to converge than TCP.

The majority of DC are microbursts, which are too small and/or too brief to converge.

Over their lengthy lifetimes, large flows can tolerate a slight convergence delay.

* DCTCP’s reaction to congestion is not severe, so it is less critical to avoid synchronization.
* In reality, hosts tend to send more bursts (30-40 packets): must make allowances for such bursts when selecting the value of K.
* K = 20 for 1Gbps (K > (C×RTT)/7).
* K = 65 for 10Gbps to avoid loss of throughput.
* DCTCP sacrifices convergence time: the time required for a new flow to grab its share of the bandwidth from an existing flow with large window size.
* Because DCTCP source must make incremental adjustments to its window size based on the accumulated multi-bit feedback in alpha.
  1. **Performance**:

DCTCP queue length is stable at around 20 packets, while the TCP queue length is 10X larger and varies widely. Figure 7 and 8.

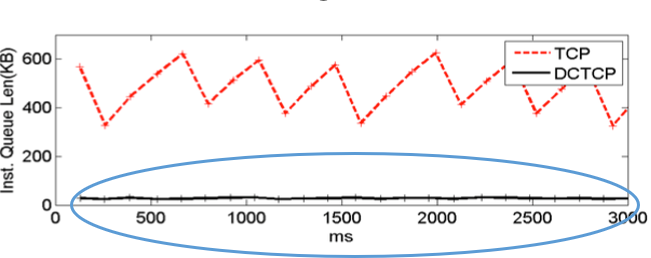


Figure 6: Shows Queue length measures on Broadcom triumph switch. Two long flows are launched from a distinct 1 GBPS port to a common 1 GBPS port. The switch has dynamic memory management enabled, allowing flows to a dynamically grab up to 700KB of a buffer.

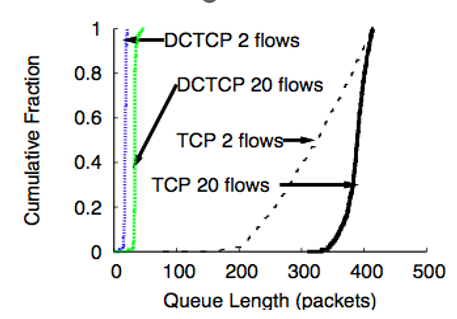
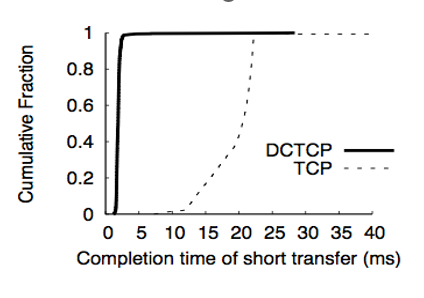


Figure 7: Short transfers see a low delay with DCTCP Figure 8: Queue length (Queue Buildup CDF)

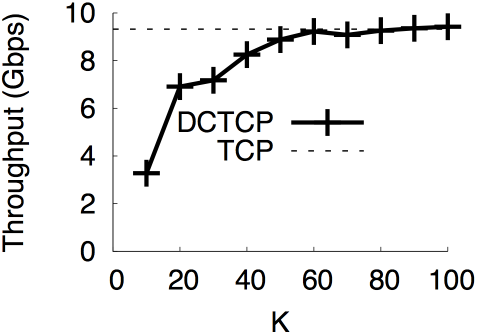
* 1. **Throughput**:
* With a 1Gbps link, performance is insensitive to the value of K.
* With 10Gbps link, K>65, DCTCP gets the same throughput as TCP. 

Figure 9: Shows Throughput for DCTCP VS TCP.

* 1. **Fairness and convergence**:
* DCTCP converges quickly, and all flows achieve their fair share.
* TCP throughput is fair on average but has much higher variation.
* 1 receiver and 5 senders connected via 1Gbps links to the Triumph switch.
* K=20.
* One sender starts a single long-lived flow, and then sequentially starts and then stops the other senders, spaced by 30 seconds
  1. **Incast:**
* DCTCP performs better than TCP and then converges at 35 senders.
* TCP begins to suffer timeouts when the number of servers exceeds 10, while DCTCP only suffers timeouts once the number of senders is large enough so that the static buffer runs out.

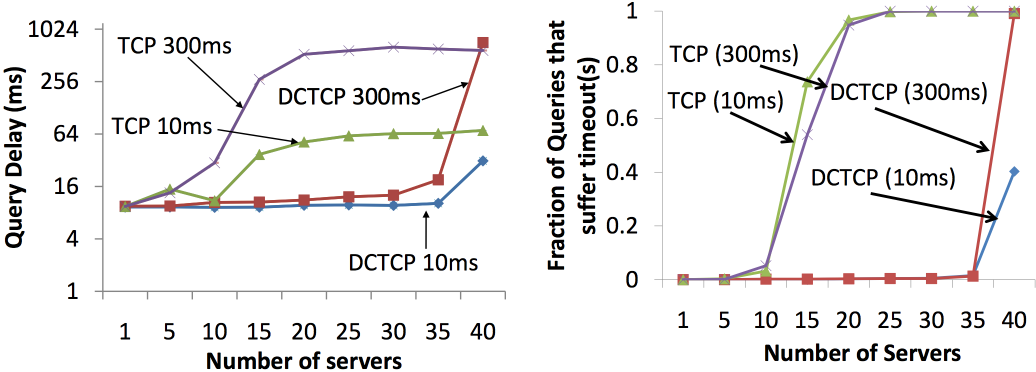


Figure 10: Shows Query delay and Fraction of Queries in the y-axis and number of servers IN x-axis for DCTCP VS TCP.

* **With dynamic buffering**:
  + DCTCP no longer suffers incast timeouts.
  + TCP continues to suffer, even with a 700KB buffer size.
* **Results**:
  + DCTCP has less response time than TCP.
  + DCTCP suffers no timeouts, while for TCP 55% of the queries suffer from timeout.
  + DCTCP’s completion time (median delay < 1 millisecond) is much lower than TCP (median delay 19 millisecond).
  + DCTCP improves latency caused by the big flows (queue build-up).
  + TCP: long flows use up the shared buffer space, so there’s less headroom to absorb incast bursts.
  + DCTCP should alleviate this problem by limiting the queue build up on the interfaces used by long flows.
  + DCTCP performs better than TCP because of low queuing delay.
  + Due to DCTCP’s amelioration of queue buildup, short-messages has lower latency.
  + TCP: 1.15% of queries suffer timeouts.
  + DCTCP: No queries suffer from timeouts.

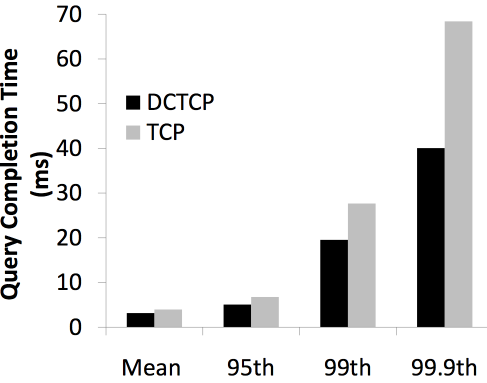


Figure 11: Shows Query completion and short message flow Time.

* + No timeout (both TCP and DCTCP from background flows).
  + Flows from 100KB to 1MB benefit most from DCTCP from background flows.

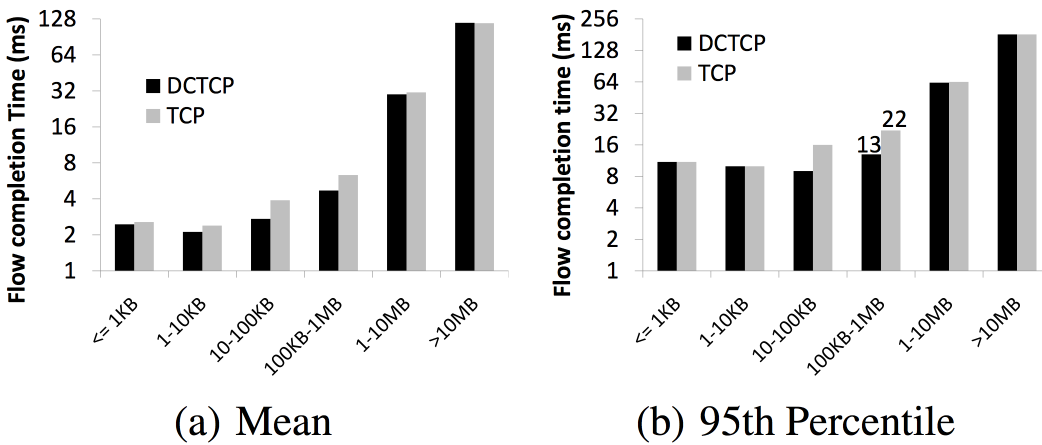


Figure 12: Shows Benchmark Traffic.

1. **Summary**

The data center would function better than it does now with TCP if it switched to DCTCP since it could accommodate 10X larger query answers and 10X larger background traffic.

Second, while employing deep buffered switches enhances query traffic performance, it degrades short-transfer performance owing to queue accumulation.

Third, due to queue length unpredictability, while RED enhances the performance of short transfers, it does not enhance the performance of query traffic.

1. **Conclusion**

Congestion control in TCP causes a number of performance issues.

Switch buffer occupancies must be consistently low while maintaining sufficient throughput for the long flows in order to accommodate the needs of the observed varied mixture of short and long flows.

With the use of multi-bit feedback obtained from a succession of Explicit Congestion Notification (ECN) markings, DCTCP, which relies on ECN, can respond quickly and avoid congestion.

1. **Reference**:

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan.

CCR October 2010, Data Center TCP (DCTCP**),**

<http://doi.acm.org/10.1145/1851275.1851192>.

## 