DCTCP Notes, 6.033 2020

Background

What environment is DCTCP designed for? What makes this environment different than, say the Internet? What applications use datacenters?

Datacenters. Datacenters are owned and operated by a single company, making it easy to broadly implement a new protocol. That's why datacenters can implement many novel ideas that are impossible to implement in the world-wide Internet.The traffic types are also different, and follow a partition/aggregate workflow pattern. Web search, retail, and advertising queries require real-time responses, and the system is designed to ignore late workers (i.e., stragglers) results (which will lower the quality of the result).

The paper mentions different traffic types. What are they, and what metrics are they sensitive to?

- Query Traffic: Latency-sensitive. ~20KB
- Background Traffic:
    - Large update flows that copy fresh data to workers (throughput-sensitive). ~50MB
    - Short message flows that update control state of workers (latency-sensitive). ~1MB

To summarize: short, latency-sensitive messages; query traffic (incast bursts); throughput-sensitive traffic.

What are the three performance impairments to DCTCP?

1. **Incast:** synchronized small flows collide causing packet drops. Packet drops cause TCP to timeout and retransmit, which adds to the delay.

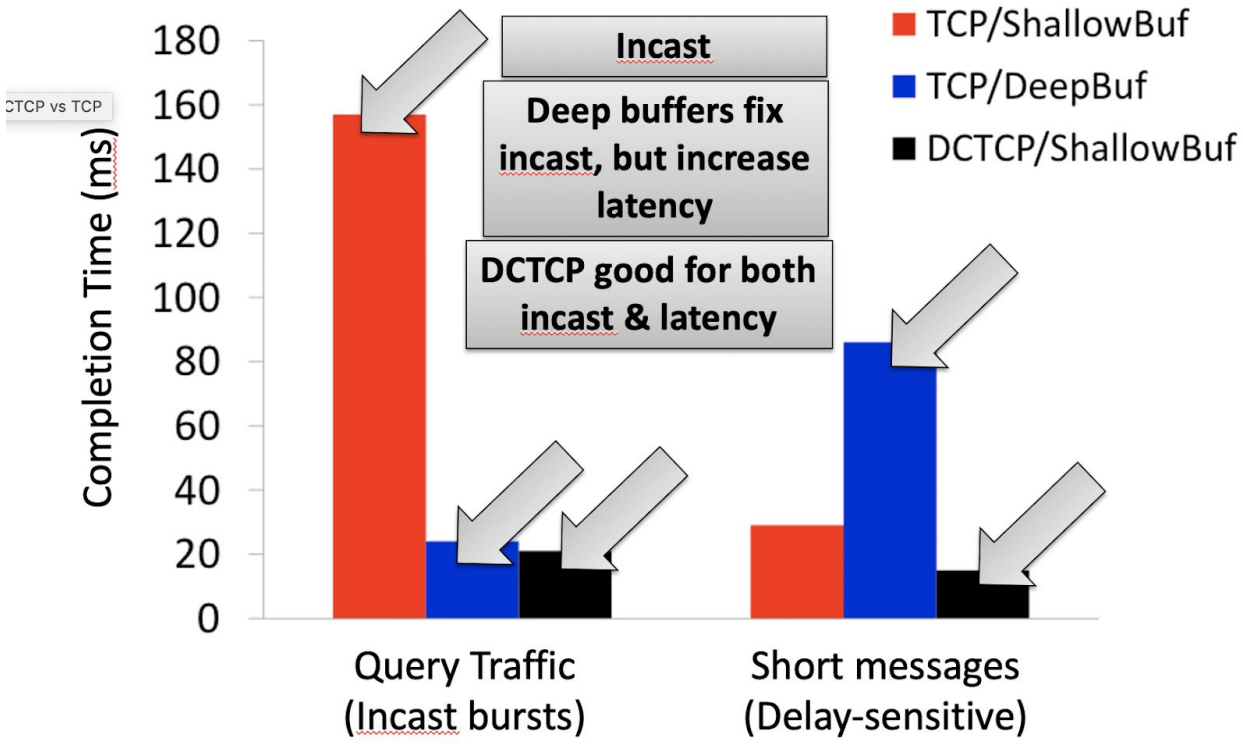
2. **Queue Buildup:** Big flows build up queues increasing latency for short flows (by order of magnitude: 1ms -> 10ms). With conventional TCP, we can have either deep buffers or shallow buffers. Deep buffers fix incast (because they have room to take packets) but increase latency for latency-sensitive packets. Shallow buffers help with latency-sensitive flows but they drop a lot of packets at incast and cause timeout.

3. **Buffer pressure:** due to shared resources. The datacenter has shallow packet buffers that form a shared pool.

What we want is a way to notify the senders "early" using ECN.

Conventional TCP "fills" the buffer and only reacts "after" it's too late and we start seeing packet loss. This means TCP's retransmission timeout will be triggered for partition/aggregate traffic patterns.

The main idea in DCTCP is to use ECN as an early notification mechanism to reduce the window size based on a fraction of marked packets. This way DCTCP keeps the buffer mostly empty to allow headroom for partition/aggregate traffic pattern.



Source: Mohammad Alizadeh

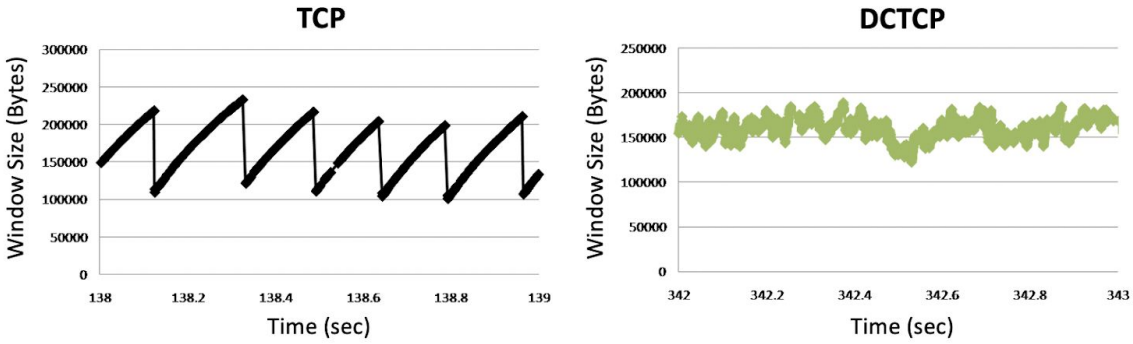
What are the goals of DCTCP?

High burst tolerance, low latency, and high throughput. So we're trying to help the small flows, but not at the expense of the big ones. DCTCP tries to keep the queue occupancy low, though non-zero.

DCTCP: Main Idea

- Extract multi-bit feedback from single-bit stream of ECN marks
  - Reduce window size based on **fraction** of marked packets.

ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	Cut window by 50%	Cut window by 40%
0 0 0 0 0 0 0 0 0 1	Cut window by 50%	Cut window by 5%



Source: Mohammad Alizadeh

DCTCP Algorithm

What does the algorithm do, at a high level?

Uses ECN to provide multi-bit feedback to end hosts; end hosts modify behavior based on this. They decrease their window in proportion to the congestion that is observed. In practice, DCTCP senders will decrease their window before a TCP sender will.

1. Mark packets if queue length > K on packet arrival

2. ACK every packet setting ECN-Echo flag if packet was marked

3. Sender maintains estimate of fraction of packets marked
   a = (1-g)a + g*F (0 < g < 1; F = fraction marked at last window)
   a = 0, no congestion; a = 1, high congestion
   cwnd = cwnd * (1 - a/2)

a is a running average of the fraction of packets marked.

So DCTCP starts gently reducing the window as soon as the queue length exceeds K.

Why does it work?

1. DCTCP reacts earlier to congestion than TCP. It also marks based on instantaneous queue lengths, which results in faster feedback to sources.
2. DCTCP also reacts in proportion to the extent of congestion, not just to its presence.
3. DCTCP keeps queues small (low latency, see fig. 19)
4. Large buffer headroom since throttling occurs when queue length > K. So DCTCP can still absorb bursts

What does DCTCP trade off for this low latency/high throughput environment?

Convergence time, i.e., the time it takes for an existing flow to grab its bandwidth. However, RTTs are so short in datacenters, that a higher convergence time is still a really small convergence time.

Trading off generality for performance

DCTCP was designed for datacenter networks. What are wide-area networks like?

- Flows are not as structured (partition/aggregate)
- Feedback is much slower (can't use instantaneous queue length)
- RTTs are higher; convergence time is more important
- High statistical multiplexing
- Care about backwards-compatibility, incremental deployment, fairness to legacy protocols

It’s not clear that DCTCP would work on a wide area network. Nor do the authors make any claim that it would.

Since DCTCP backs off less aggressively than standard AIMD, it would get a greater share of bandwidth than legacy flows. Also, the latency benefits of DCTCP are less important in a WAN.

Some additional thoughts from Manya Ghobadi: Moreover, if a router does not implement ECN, it may drop the ECN marked packets it receives from other routers, creating a huge loss. In fact, this was a case at a datacenter in 2018: we found that although DCTCP was disabled, one router was marking ECN bits but other routers were dropping the ECN marked packets which

led into unexplained packet loss in the network (we were seeing packet loss but the buffers were not nearly full). We fixed this issue by disabling ECN all across the datacenter.

What do you think about the tradeoff between performance and generality? Should we develop a different version of TCP for every environment? What happens when those different versions interact? Will a generalizable protocol -- for TCP or otherwise -- necessarily result in worse performance?

If we're going to make specialized protocols, should they run at such a low layer? Typically we think about getting more specialized as we go higher up the stack.

No real “correct” answer here, these are just things to think about.