Data Center TCP

Presented by Shreyas Shah (slshah@ucdavis.edu, 922938069)

Authors: M. Alizadeh et al.

UCDAVIS

The Problem

Problem:

- Data centers run apps with different needs: low latency vs. high throughput
- Standard TCP struggles in this environment
- Background traffic fills switch buffers → delays foreground traffic

Proposed Solution:

- DCTCP: a TCP-like protocol designed for data centers
- Uses ECN for better feedback on congestion
- Works well with shallow-buffered switches
- Improves latency and handles traffic bursts effectively

Background & Challenges

- Data centers use cheap, commodity switches
- Apps have mixed needs: short (low latency) & long (high throughput) flows
- Short flows miss deadlines if delayed
- Long flows fill buffers → slow down short traffic
- Need: low latency, burst tolerance, high throughput

DCTCP Solution

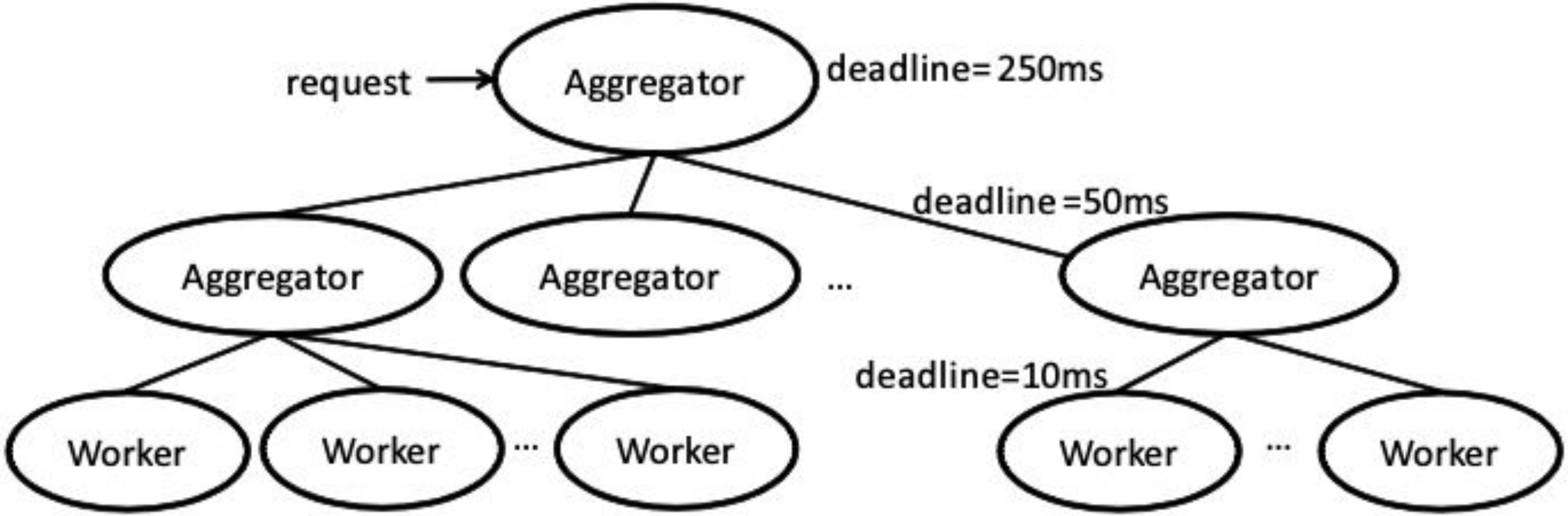
- Real data: 99.9% traffic is TCP, diverse sizes
- DCTCP keeps buffers low + throughput high
- Uses ECN for early congestion feedback
- Simple: ~30 lines of code change
- Designed for data center environment

Partition/Aggregate & Latency Challenges

- Partition/Aggregate = common design in large apps
- Requests split → many workers → results combined
- Tight latency budget: ~230–300 ms total
- Workers must respond in 10–100 ms
- Delays hurt result quality and user experience
- TCP struggles → custom fixes needed

UCDAVIS

Partition/Aggregate



Workload in Data Centers

- Study: 6000+ servers, 150TB data collected

- Three traffic types:
    - Query traffic: tiny, latency-critical flows (~1.6KB)
    - Background traffic: large updates (1–50MB) + small control messages
    - Concurrent flows: often 30+ active; some servers 1000+

- All types coexist → hard for TCP to perform well

Performance Problems in Data Centers

Because switches have small shared buffers, data centers face three recurring problems:

- Incast: many small responses at once overflow buffers.
- Queue buildup: long flows slow down short ones.
- Buffer pressure: heavy traffic on one port affects all others.

DCTCP: Goal & Key Idea

- Goal: high throughput, low latency, burst tolerance
- Works with shallow-buffered switches
- Reacts proportionally to congestion
- Uses ECN marking to signal congestion early

UCDAVIS

How the DCTCP Algorithm Works

- Marking (Switch): mark packets when queue > K

- Feedback (Receiver): reports exactly which packets were marked

- Control (Sender): estimates congestion (α) and adjusts sending rate

Formula: $cwnd = cwnd \times (1 - \alpha/2)$

UCDAVIS

Benefits of DCTCP

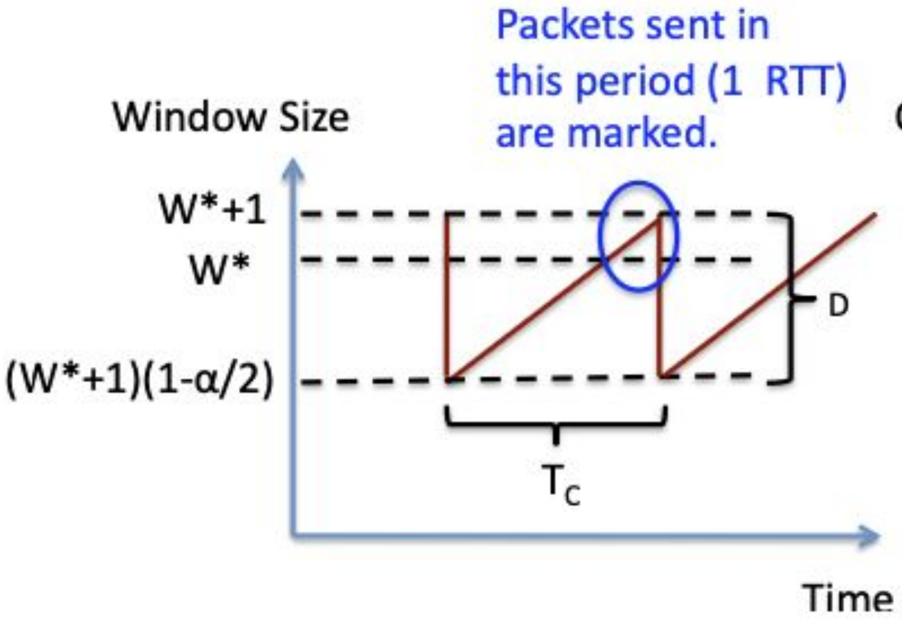
- Queue buildup: reacts early → keeps delays low
- Buffer pressure: prevents one port from hogging memory
- Incast: early marking reduces burst size → fewer drops/timeouts

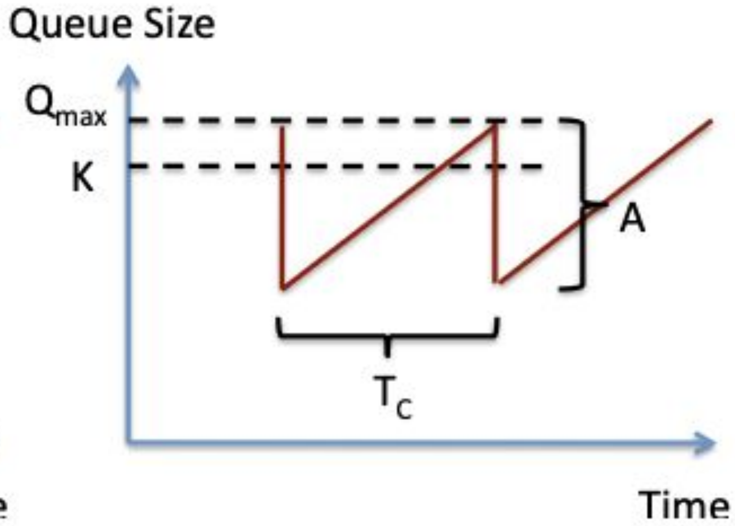
Steady-State Behavior of DCTCP

- Flows stabilize into a sawtooth pattern: queue grows, gets marked, then shrinks
- DCTCP reacts proportionally to congestion, not suddenly like TCP
- Keeps queue small and stable → low latency & fewer drops
- Works well with small buffers and few flows

UCDAVIS

Sawtooth Pattern





UCDAVIS

Choosing DCTCP Parameters

Marking Threshold (K):

- Point where switches start marking packets
- Must be high enough to avoid empty queues
- Must be low enough to keep latency low

Estimation Gain (g):

- Controls how quickly sender reacts to congestion
- Must be small for smooth, stable adjustments
- Prevents overreacting or reacting too slowly

UCDAVIS

Discussion & Practical Insights

AQM (Active Queue Management) alone isn't enough:

- Struggles with bursty traffic and few flows
- Tradeoff: high delay or low throughput

Convergence:

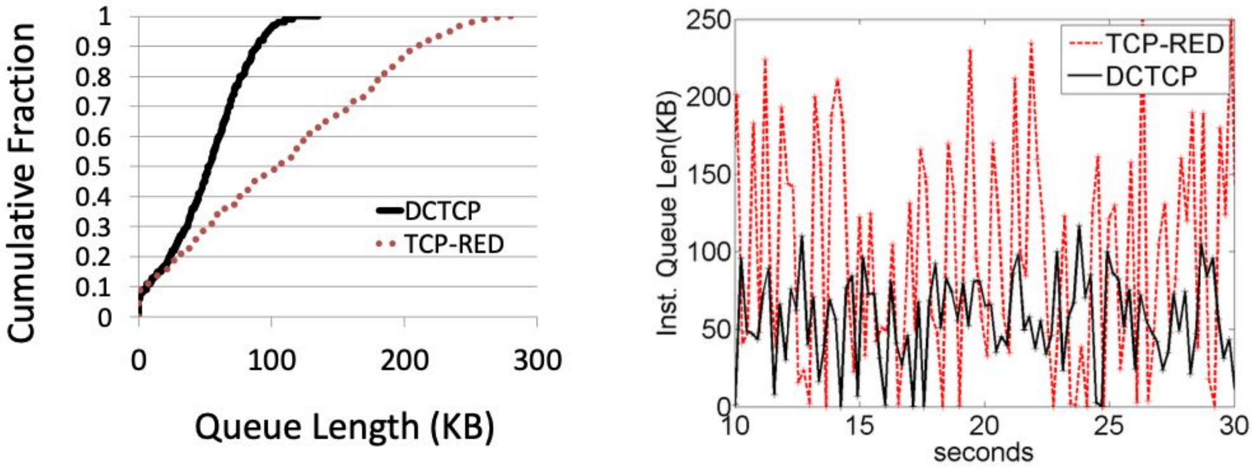
- DCTCP adjusts slowly but delay is small in data centers
- Microbursts don't need convergence, big flows can tolerate it

Practical notes:

- Real networks are bursty → use higher marking thresholds

UCDAVIS

DCTCP vs RED



(a) CDF of queue length (b) Time series of queue length

Figure 15: DCTCP versus RED at 10Gbps

DCTCP – Results Summary

Performance

- Matches TCP throughput (~100%)
- Keeps queues small and stable (~20 packets)

Fairness & Stability

- Quickly shares bandwidth fairly
- Works well even in multi-hop networks

Impairments Fixed

- Incast: Fewer timeouts, even with 40+ senders
- Queue buildup: Short flows complete much faster
- Buffer pressure: Less packet loss, better isolation

Real Traffic & Scale

- Eliminates query timeouts (0% vs 1.15%)
- Handles 10× more traffic with low latency
- Outperforms RED and deep buffers

UCDAVIS

Possible Questions

- Is it acceptable to sacrifice convergence speed for lower latency and stable queues?

- How could DCTCP's approach impact multi-tenant environments where fairness between applications matters?

- Could DCTCP principles apply to non-data center scenarios like IoT networks or edge computing?

- If deep-buffered switches solve some of TCP's issues, why not just use them everywhere instead of DCTCP?

UCDAVIS