Congestion Control for Data Centers

Lecture 21, Computer Networks (198:552)
Fall 2019

Material adapted from slides by Mohammad Alizadeh
Review: TCP congestion control

• Keep some in-flight (un-ACK’ed) packets: congestion window

• Adjust window based on several algorithms:
  • Startup: slow start
  • Steady state: AIMD
  • Loss: fast retransmission, fast recovery

• Main question for this lecture:
  • (How) should this design change for data centers?
DC Transport Requirements

High throughput, low latency, burst tolerance
INTERNET Servers

Fabric

100Kbps–100Mbps links
~100ms latency

Transport inside the DC

10–40Gbps links
~10–100μs latency

Servers
Transport inside the DC

INTERNET

Fabric

Interconnect for distributed compute workloads

Transport

web
app
cache

request → Aggregator

deadline = 250ms

deadline = 50ms

deadline = 10ms

Aggregator

Aggregator

Worker

Worker

Worker

Worker

Worker
Data center workloads

• Mice and Elephants

• Short messages
  (e.g., query, coordination)

• Large flows
  (e.g., data update, backup)

  → Low Latency

  → High Throughput
Incast

Worker 1

Worker 2

Worker 3

Worker 4

• Synchronized fan-in congestion

Aggregator

RTO_{\text{min}} = 300 \text{ ms}

TCP timeout

Vasudevan et al. (SIGCOMM’09)
Trace of a real incast event

Maybe, reduce RTO to mitigate this
Jittering to mitigate incast

Jittering trades of median for high percentiles
HOL Blocking and Buffer Pressure

Queue buildup increases latency for everyone
(Reducing RTO doesn’t help latency)
Another possibility: Delay-based CC

- Keep just a few packets in queues by observing delays

\[ \text{queue}_\text{use} = \text{cwnd} - \text{BWE} \times \text{RTT}_{\text{noLoad}} = \text{cwnd} \times (1 - \text{RTT}_{\text{noLoad}}/\text{RTT}_{\text{actual}}) \]

- Adjust window such that only a few packets are in queue

\[ \alpha \leq \text{queue}_\text{use} \leq \beta \]

- RTT estimates need to be very accurate and precise
  - Can be challenging in low-RTT data centers
Data Center TCP (DCTCP)
Design of the congestion control algorithm
Review: TCP algorithm

Additive Increase: 
$W \rightarrow W+1$ per round-trip time

Multiplicative Decrease: 
$W \rightarrow W/2$ per drop or ECN mark

ECN Mark (1 bit)

ECN = Explicit Congestion Notification
DCTCP: Main idea

• Extract multi-bit feedback from single-bit stream of ECN marks
  • Reduce window size based on fraction of marked packets
DCTCP: Main idea

<table>
<thead>
<tr>
<th>ECN Marks</th>
<th>TCP</th>
<th>DCTCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 1 1 0 1 1 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 40%</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0 1</td>
<td>Cut window by 50%</td>
<td>Cut window by 5%</td>
</tr>
</tbody>
</table>

**Window Size**

**TCP**

**DCTCP**

[Graphs showing window size changes over time for TCP and DCTCP]
DCTCP algorithm

Switch side:
• Mark packets when Queue Length > K.

Sender side:
• Maintain running average of fraction of packets marked ($\alpha$).

\[
\text{each RTT: } F = \frac{\text{# of marked ACKs}}{\text{Total # of ACKs}} \implies \alpha \leftarrow (1 - g)\alpha + gF
\]

• Adaptive window decreases: 
  \[
  W \leftarrow (1 - \frac{\alpha}{2})W
  \]
  • Note: decrease factor between 1 and 2.
Efficient and “lossless” ACK generation

- Send 1 ACK for every m packets with ECN=0
- Send immediate ACK with ECN=0
- Send 1 ACK for every m packets with ECN=1
- Send immediate ACK with ECN=1
DCTCP vs TCP

Experiment: 2 flows (Win 7 stack), Broadcom 1Gbps Switch

DCTCP mitigates Incast by creating a large buffer headroom

Buffer is mostly empty
Why it works

1. Low Latency
   ✓ Small buffer occupancies $\rightarrow$ low queuing delay

2. High Throughput
   ✓ ECN averaging $\rightarrow$ smooth rate adjustments, low variance

3. High Burst Tolerance
   ✓ Large buffer headroom $\rightarrow$ bursts fit
   ✓ Aggressive marking $\rightarrow$ sources react before packets are dropped
Setting parameters: A bit of analysis

• How much buffering does DCTCP need for 100% throughput?

• Need to quantify queue size oscillations (stability).

\[ \alpha = \frac{\text{# of pkts in last RTT of Period}}{\text{# of pkts in Period}} \]
Setting parameters: A bit of analysis

• How small can queues be without loss of throughput?

- Need to quantify queue size oscillations (Stability).

\[ K > \left( \frac{1}{7} \right) C \times RTT \]

for TCP:

\[ K > C \times RTT \]
Bing benchmark (baseline)

**Background Flows**

- Flow Completion Time (ms):
  - 10-100KB: DCTCP 9, TCP 16
  - 100KB-1MB: DCTCP 13, TCP 22
  - 1-10MB: DCTCP 63, TCP 64
  - >10MB: DCTCP 182, TCP 182

**Query Flows**

- Query Completion Time (ms):
  - Mean: DCTCP 3, TCP 4
  - 95th: DCTCP 5, TCP 7
  - 99th: DCTCP 19, TCP 28
  - 99.9th: DCTCP 40, TCP 68
Convergence time

- DCTCP takes at most ~40% more RTTs than TCP
  - “Analysis of DCTCP”, SIGMETRICS 2011
- Intuition: DCTCP makes smaller adjustments than TCP, but makes them much more frequently
CC evaluation: several aspects!

- Throughput, delays, flow completion times
- Fairness, convergence times
- Specific impairments:
  - incast (many to one, all to all)
  - collateral damage from incast
  - buffer pressure
- Impact on background traffic
- Multi-hop versus single-hop bottlenecks
CC Deployment Concerns

Life ain’t easy in the fast lane
Practical deployment concerns in DCs

• Coexistence with legacy protocols like TCP Cubic
  • Application code can’t be upgraded in one shot
• Minimum window size matters during heavy incast events
  • e.g., 2 packets versus 1 packet!
• Setting pkt flags appropriately at senders, receivers, and routers
  • Non “ECN-capable” flagged packets will be dropped when Q > K
  • … including the SYN packets of any connection
• Receive-buffer tuning
  • Receive buffer must be at least BDP, but what is the BDP?