Data Center Transport

Lecture 10
Srinivas Narayana
http://www.cs.rutgers.edu/~sn624/553-S23

Parts of this lecture were heavily adapted from slides by Mohammad Alizadeh
DC Transport Requirements

High throughput, low latency, burst tolerance
100Kbps–100Mbps links
~100ms latency

10–100Gbps links
~10–100μs latency

Transport inside the DC
Transport inside the DC

INTERNET

Fabric

Interconnect for distributed compute workloads

request → Aggregator

deadline = 250ms

Aggregator

Aggregator

deadline = 50ms

Aggregator

deadline = 10ms

Worker Worker ...

Worker Worker ...

web app cache
Data center workloads

• Mice and Elephants

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

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

Coexistence creates some performance impairments...

Source with B bytes to send

Delay D

Rate R

Completion time $D + B/R$
(1) Incast

Worker 1

Worker 2

Worker 3

Worker 4

Aggregator

• Synchronized fan-in congestion

RTO_{min} = 300 ms

TCP timeout

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

One option: reduce RTO to mitigate this
Another option: Jittering to avoid sync

- Jittering switched off around 8:30 am.

Jittering trades off median for high percentiles
Head of line blocking: Queue buildup

- Resource contention occurs in the core of the network.
- Congestion control will react, but may be too little & too late:
  - Congestion control can’t prevent packet drops “now”
  - Congestion control won’t prevent high-sending-rate flows from inflicting large delays or recurring drops
Queue buildup on a port increases delays for all flows using that port (head of the line blocking).

Reducing RTO doesn’t help latency when there are no drops.
(3) Shared memory buffering

- Where should the packets not currently serviced wait?
- Input-queued vs. output-queued (preferable design)
- Buffer management: how to put packets into the buffer
- Scheduling: how to schedule packets leaving the buffer
Shared memory buffering means that queues building up on a different port can also impact flows.
Need to keep queues small. Use delay-based CC?

• Keep just a few packets in queues by observing delays

\[
\text{queue\_use} = cwnd - \text{BWE} \times RTT_{\text{noLoad}} = cwnd \times (1 - \frac{\text{RTT}_{\text{noLoad}}}{\text{RTT}_{\text{actual}}})
\]

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

\[\alpha \leq \text{queue\_use} \leq \beta\]

• RTT estimates need to be very accurate and precise
  • Difficult in low-RTT data centers.
  • Challenges: Software queueing & scheduling delays. Timer tick res
Data Center TCP (DCTCP)

Design of the congestion control algorithm
Review: TCP congestion control

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

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

• Main question for this lecture:
  • (How) should this design change for data centers?
Behavior of Additive Increase

Say \( \text{MSS} = 1 \text{ KByte} \)
Default \( \text{ssthresh} = 64\text{KB} = 64 \text{ MSS} \)

- Loss occurs at \( \text{cwnd} = 54\text{K} \)
- Set \( \text{ssthresh} \) to 27 MSS
- Additive increase
- Loss occurs at \( \text{cwnd} = 40\text{K} \)
- Set \( \text{ssthresh} \) to 20 MSS

Diagram:
- Slow start
- Additive increase
- Packet drops
- RTO
Additive Increase/Multiplicative Decrease

Say $\text{MSS} = 1$ KByte
Default $\text{sssthresh} = 64$KB = 64 $\text{MSS}$
Explicit Congestion Notification

Additive Increase: 
\[ W \rightarrow W + 1 \text{ per round-trip time} \]

Multiplicative Decrease: 
\[ W \rightarrow W/2 \text{ per drop or ECN mark} \]

Queue size > K, mark the packet.

Example of Active Queue Management (RED, PI, etc.)

ECN = Explicit Congestion Notification

Sender 1

Sender 2

Receiver

Window Size (Rate) vs. Time

ECN Mark (1 bit)
ECN set on the IP header by routers

- **00** – Not ECN-Capable Transport, Not-ECT
- **01** – ECN Capable Transport(1), ECT(1)
- **10** – ECN Capable Transport(0), ECT(0)
- **11** – Congestion Experienced, CE.

Dropped if TCP sender is not ECN enabled
Explicit Congestion Notification

Additive Increase:
W → W+1 per round-trip time

Multiplicative Decrease:
W → W/2 per drop or ECN mark

Receiver’s ACK echoes mark in TCP header

ECN = Explicit Congestion Notification
## ECN on the TCP header

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Octet</th>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>448</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TCP segment header**

<table>
<thead>
<tr>
<th>Octet</th>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Source port</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>Sequence number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>Acknowledgment number (if ACK set)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td>Data offset</td>
<td>Reserved</td>
<td>C</td>
<td>E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>W</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>128</td>
<td>Checksum</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>160</td>
<td>Urgent pointer (if URG set)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>448</td>
<td>Options (if data offset &gt; 5. Padded at the end with &quot;0&quot; bits if necessary.)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>1011110111</td>
<td>Cut window by 50%</td>
<td>Cut window by 40%</td>
</tr>
<tr>
<td>0000000001</td>
<td>Cut window by 50%</td>
<td>Cut window by 5%</td>
</tr>
</tbody>
</table>

**Graphs:**
- TCP: Window Size (Bytes) vs. Time (sec)
- DCTCP: Window Size (Bytes) vs. Time (sec)
DCTCP algorithm

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

Sender side:
• Maintain running average of fraction of packets marked (α).

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

• Adaptive window decreases: \( W \leftarrow (1 - \frac{\alpha}{2})W \)
  • Note: decrease factor between 1 and 2.

Reacting to and controlling queue size distribution
Delayed ACKs

• Not every packet is ACKnowledged by receiver

• Too many ACKs: increase packet processing load
  • Typical policy: ACK every m packets, or after sender has paused transmitting for a delayed ACK timeout

• How to allow the sender to see the full stream of ECN marks?
Efficient and “lossless” ACK generation

Send 1 ACK for every m packets with ECN=0

Send immediate ACK with ECN=0

CE = 0

Send immediate ACK with ECN=1

CE = 1

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

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

DCTCP mitigates Incast by creating a large buffer headroom
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?

Ø Need to quantify queue size oscillations (Stability).

\[ K > \frac{1}{7} C \times RTT \]

for TCP:

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

**Background Flows**

<table>
<thead>
<tr>
<th>Flow Size</th>
<th>DCTCP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>10-100KB</td>
<td>9</td>
<td>16</td>
</tr>
<tr>
<td>100KB-1MB</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>1-10MB</td>
<td>63</td>
<td>64</td>
</tr>
<tr>
<td>&gt;10MB</td>
<td>182</td>
<td>182</td>
</tr>
</tbody>
</table>

**Query Flows**

<table>
<thead>
<tr>
<th>Metric</th>
<th>DCTCP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>95th</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>99th</td>
<td>19</td>
<td>28</td>
</tr>
<tr>
<td>99.9th</td>
<td>40</td>
<td>68</td>
</tr>
</tbody>
</table>
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

![TCP Convergence Time](image)

![DCTCP Convergence Time](image)
CC evaluation: many aspects to consider

- Throughput, delays, flow completion times
- Fairness, convergence times
- Specific impairments:
  - incast (many to one, all to all)
  - Queue buildup
  - Buffer pressure
  - Collateral damage from incast
- Multi-hop versus single-hop bottlenecks
- Comparison against existing TCPs ad AQMs
- How deployable: app awareness, hardware compatibility, …
CC Deployment Concerns

Life isn’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: no reactive scheme can work if buffers are so small that drop in one RTT

• Enabling appropriate options at senders, receivers, and routers
  • Non “ECN-capable” flagged packets will be dropped when Q > K
  • … including the SYN packets of any connection

• Receive-side buffer tuning
  • Reacting to increasing buffer demands at the endpoints takes time
  • Static: Usually, receive buffer must be at least BDP; also influenced significantly by queueing