TCP Prague
a prototype for L4S Congestion Control

Bob Briscoe, Independent <ietf@bobbriscoe.net>
Koen De Schepper, NOKIA Bell Labs <koen.de_schepper@nokia.com>
Olivier Tilmans, NOKIA Bell Labs <olivier.tilmans@nokia-bell-labs.com>
Asad Sajjad Ahmed, Independent <me@asadsa.com>
Joakim Misund, Uni Oslo <joakim.misund@gmail.com>

iccrg, IETF-109, Nov 2020
Invitation to collaborate

- In all the early work on L4S, DCTCP in v3.19 Linux gave ultra-low delay
- For 3 years L4S team was focused on AQM products (& CC safety aspects)
  - while, with later kernels, DCTCP was no longer performing
- Prague git repo now tracks the current kernel – with performance restored
  - Updating involved tracing interrelated problems between the kernel and the DCTCP module
- Time for a relaunch
  - TCP Prague & AccECN codebase now usable for others to build on
  - Deployments of network part pending codepoint assignment
Brief DCTCP tutorial
Smoothing Congestion Signals

- Classic AQMs filter out queue fluctuations to avoid unnecessary drops
- DCTCP uses ECN so it can shift responsibility for smoothing from AQM to sender
- Smoothing adds the following feedback delay:

<table>
<thead>
<tr>
<th></th>
<th>Bottleneck AQM</th>
<th>Sender CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic</td>
<td>100 to 200 ms (worst-case RTT)</td>
<td>0</td>
</tr>
<tr>
<td>L4S</td>
<td>0</td>
<td>0 to RTT timescale (each flow’s own RTT)</td>
</tr>
</tbody>
</table>
Smoothing Congestion Signals

- Classic AQMs filter out queue fluctuations to avoid unnecessary drops
- DCTCP uses ECN so it can shift responsibility for smoothing from AQM to sender
- Smoothing adds the following feedback delay:

<table>
<thead>
<tr>
<th></th>
<th>Bottleneck AQM</th>
<th>Sender CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic</td>
<td>100 to 200 ms (worst-case RTT)</td>
<td>0</td>
</tr>
<tr>
<td>L4S</td>
<td>0</td>
<td>0 to RTT timescale (each flow’s own RTT)</td>
</tr>
</tbody>
</table>

if (inst_qDelay > threshold) ECN-mark;

can choose not to smooth, e.g. during flow start
DCTCP Sender: Differences from Reno

- Congestion window reduction depends on extent of congestion
  - not just existence
- Maintain fraction ($F$) of marked pkts, then
  \[ F = \frac{\text{no. of marked ACKs}}{\text{total no. of ACKs}} \]

Each RTT:

- Congestion window ($W$) reduction:
- Compared to Reno:

  \[ \alpha \leftarrow gF + (1-g)\alpha \]
  where $g$ is gain \([\frac{1}{16}]\)

  \[ \alpha W/2 \]
  \[ W/2 \]
Headroom for short flows

- Short flows (and bursts) are effectively unresponsive
- The classic approach (incl. BBR) is for short flows to burst into the buffer
- Long-running DCTCP senders leave headroom for the recent level of short flows and bursts
  - which they learn by maintaining an EWMA $\alpha$ of ECN feedback

- An ongoing trend of short flows fits in the headroom below the Q threshold
  - but occasional short flows can still queue above the threshold
Prague Developments
### TCP Prague Linux Reference Code: Status (Nov'20)

<table>
<thead>
<tr>
<th>Requirements</th>
<th>base TCP</th>
<th>TCP Prague</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4S-ECN Packet Identification: ECT(1)</td>
<td>mandatory</td>
<td></td>
</tr>
<tr>
<td>Accurate ECN TCP feedback</td>
<td>sysctl option</td>
<td>mandatory</td>
</tr>
<tr>
<td>Reno-friendly on loss</td>
<td>inherent</td>
<td></td>
</tr>
<tr>
<td>Reno-friendly if classic ECN bottleneck</td>
<td>default off → on later</td>
<td></td>
</tr>
<tr>
<td>Reduce RTT dependence (low RTT dominance)</td>
<td>default on</td>
<td></td>
</tr>
<tr>
<td>Scale down to fractional window</td>
<td>research git</td>
<td>research git</td>
</tr>
<tr>
<td>Detecting loss in units of time</td>
<td>default RACK</td>
<td>default RACK</td>
</tr>
</tbody>
</table>

### Performance Improvements

<table>
<thead>
<tr>
<th>Performance Improvements</th>
<th>module option off</th>
<th>default off → on later</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECN-capable TCP control packets</td>
<td>research git</td>
<td>research git</td>
</tr>
<tr>
<td>Faster flow start</td>
<td></td>
<td>in progress</td>
</tr>
<tr>
<td>Faster than additive increase</td>
<td></td>
<td>default on</td>
</tr>
<tr>
<td>Continuous additive increase</td>
<td></td>
<td>Todo</td>
</tr>
<tr>
<td>Reduce RTT dependence (high RTT weakness)</td>
<td></td>
<td>default</td>
</tr>
<tr>
<td>Burst avoidance for TSO sizing &amp; pacing (&lt;1ms)</td>
<td></td>
<td>default</td>
</tr>
</tbody>
</table>

### Performance-bug fixes

<table>
<thead>
<tr>
<th>Performance-bug fixes</th>
<th>fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer scaling &amp; fractional carry (alpha, cwnd, etc)</td>
<td></td>
</tr>
<tr>
<td>PRR undershoot spike</td>
<td></td>
</tr>
</tbody>
</table>

Linux code: none | none (simulated) | research private | research opened | L4Steam git | Linux RFC | Linux mainline
Prague Benefits today

- DCTCP on a STEP AQM
- Prague on a STEP AQM

Adapted pacing / TSO and bug-fixed integer-roundings / PRR / partial-AI
Smoother steady state additive increase

- With marks expected in every RTT, no time for additive increase

- Like Classic TCPs, DCTCP suppresses AI in the RTT after MD
  - Unnecessary variation, to force periods with no marking and more marking
  - Worse RTT dependence, as longer RTTs can’t increase

- Prague increases on every ACK except on echoed ECN marks
Prague Benefits today

- ECN-Cubic on a CoDel AQM
- Prague on a STEP AQM

100Mbps 40ms

Smoother Throughput, smaller Queue
Prague Benefits today

- Cubic on a CoDel AQM
- Prague on a STEP AQM

Better RTT-independent fairness
Invitation to collaborate
Research opportunities

- Congestion control exploiting high fidelity ECN markings
  - relevant beyond L4S

- Possible topics to work on:
  - As L4S moves from testbed to Internet
  - Both Pragmatic deployment facilitators and longer term research
    - Single-ended TCP deployment
    - Integration with BBR/Cubic instead of Reno
    - Improved flow startup
    - Faster tacking of available capacity, e.g. over 5G mmWave links
    - Improved detection of RFC 3168 marking behaviour
      - Distinguishing FIFO from FQ
    - Burstiness introduced by network (multiple access)

- What would a relaunch need to look like, for you to want to get involved?
Evaluation and reproducibility

• Reference test-cases
  • RFC7928; draft-ietf-rmcat-eval-criteria ?

• Common metrics

• Comparable visualizations

• Reusable tools
How to Get Started

- L4S landing page
  - https://riteproject.eu/dctth/
- TCP Prague mailing list
  - https://www.ietf.org/mailman/listinfo/tcpprague
- Open source code from L4S team
  - Linux kernel code, testbed scripts and GUI visualizer, Prague virtual machine, ...
  - https://github.com/L4Steam
  - https://l4steam.github.io/
- Pete Heist’s L4S evaluation testbed scripts
  - https://github.com/heistp/l4s-tests and others
  - NYU Wireless fork for CloudLab-based deployment (https://github.com/ffund/sce-l4s-bakeoff)
- ns-3 simulation models (some in mainline, some out-of-tree)
  - Prague, AccECN, DualQ, FQ/CoDel/Cobalt/PIE with L4S support, scenario scripting
  - https://www.nsnam.org/wiki/L4S-support
Prague Congestion Control

Q&A

and spare slide
DCTCP AQM: Difference from Classic

- Immediate AQM – no smoothing
  - Simple ramp or step threshold
- Can configure $Q$ much lower than optimum in [DCTCP-stability]
  \[ Q \approx 0.17 \times RTT \]
  - Because utilization is fairly insensitive to non-optimal $Q$

Event Cycles

Typical DCTCP implementation

rcvr

sndr

time

update: $F$
update: $\alpha$

congestion window reduction

CWR state (1 RTT)
no further reductions

CWR state (1 RTT)
no further reductions

reduction triggered on first ECN mark after !CWR

$= ECN$ mark