TCP Prague
a prototype for L4S Congestion Control

Bob Briscoe, Independent <ietf@bobbriscoe.net>

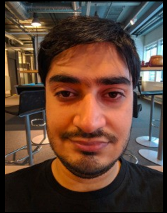
Koen De Schepper, NOKIA Bell Labs <koen.de_schepper@nokia.com>

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:

	Bottleneck AQM	Sender CC
Classic	100 to 200 ms (worst-case RTT)	0
L4S	0	0 to RTT timescale **(each flow's own RTT)**

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:

	Bottleneck AQM	Sender CC
Classic	100 to 200 ms (worst-case RTT)	0
L4S	0	0 to RTT timescale **(each flow's own RTT)**

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 EWMA (α) of F

Each RTT: $$F = \frac{no.\,of\ marked\ ACKs}{total\ no.\,of\ ACKs}$$

$$\alpha \leftarrow gF + (1\text{-}g)\alpha$$

where g is gain [$^1/_{16}$]

- Congestion window (W) reduction: $\alpha W/2$
- Compared to Reno: $W/2$

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 α 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)

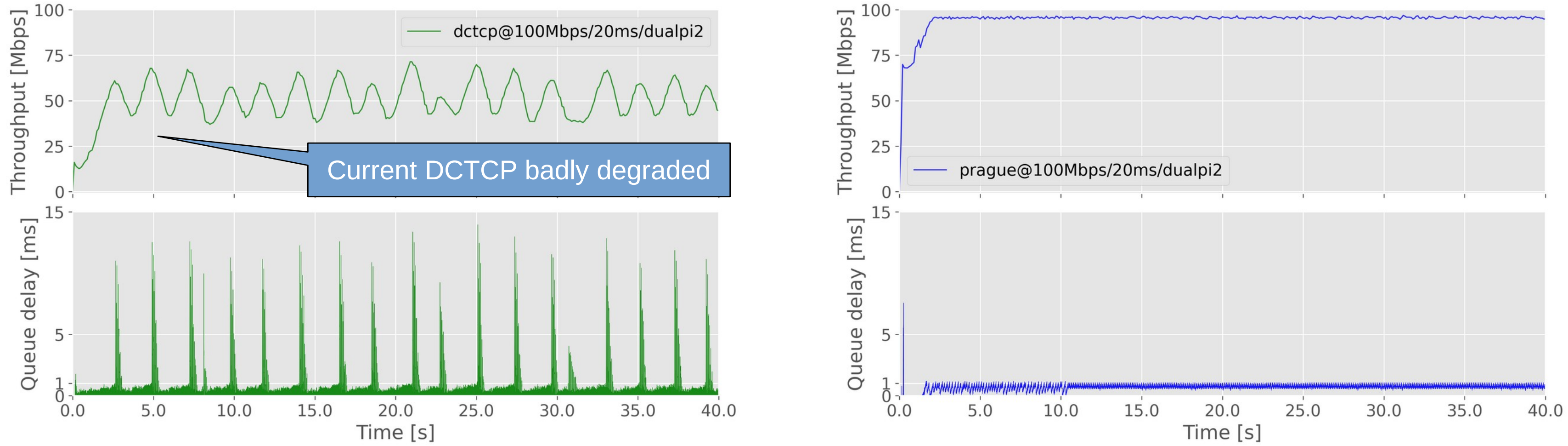
Linux code:	none	none (simulated)	research private	research opened	L4Steam git	**Linux RFC**	**Linux mainline**

Requirements	base TCP	TCP Prague
L4S-ECN Packet Identification: ECT(1)		**mandatory**
Accurate ECN TCP feedback	**sysctl option**	**mandatory**
Reno-friendly on loss		**inherent**
Reno-friendly if classic ECN bottleneck		default off → on later
Reduce RTT dependence (low RTT dominance)		default on
Scale down to fractional window	research git	research git
Detecting loss in units of time	**default RACK**	**default RACK**
Performance Improvements		
ECN-capable TCP control packets	**module option off**	**default off→on later**
Faster flow start	research git	research git
Faster than additive increase		in progress
Continuous additive increase		default on
Reduce RTT dependence (high RTT weakness)		Todo
Burst avoidance for TSO sizing & pacing (<1ms)		default
Performance-bug fixes		
integer scaling & fractional carry (alpha, cwnd, etc)		fixed
PRR undershoot spike		fixed

Prague Benefits today

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

100Mbps 20ms



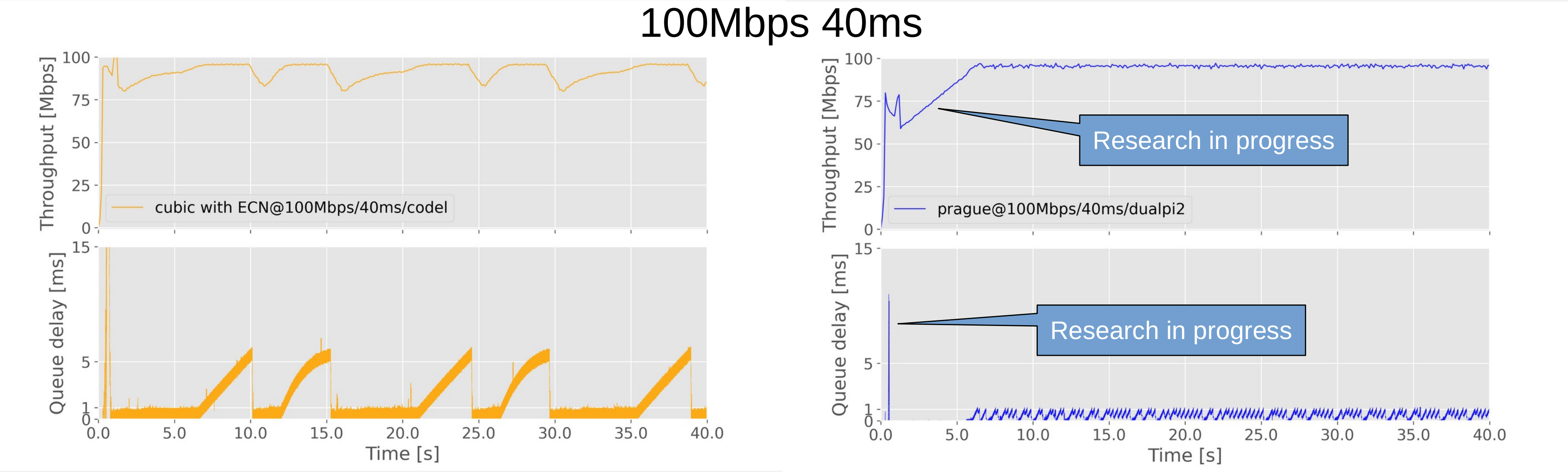
Adapted pacing / TSO and bug-fixed integer-roundings / PRR / partial-AI

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

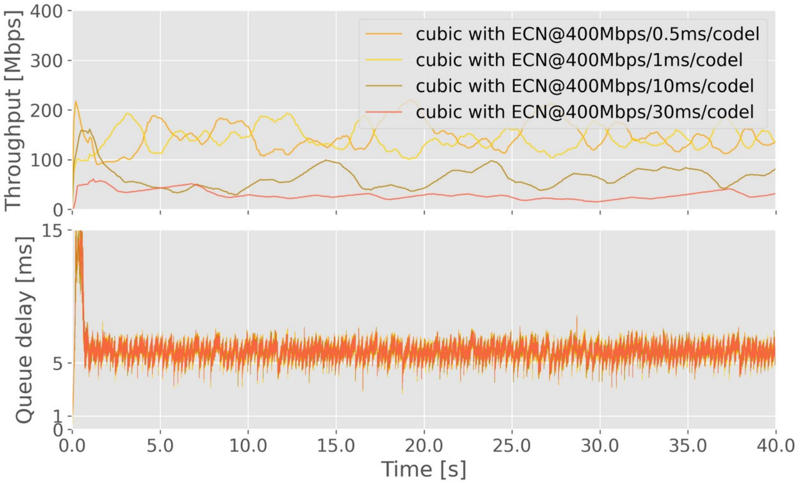


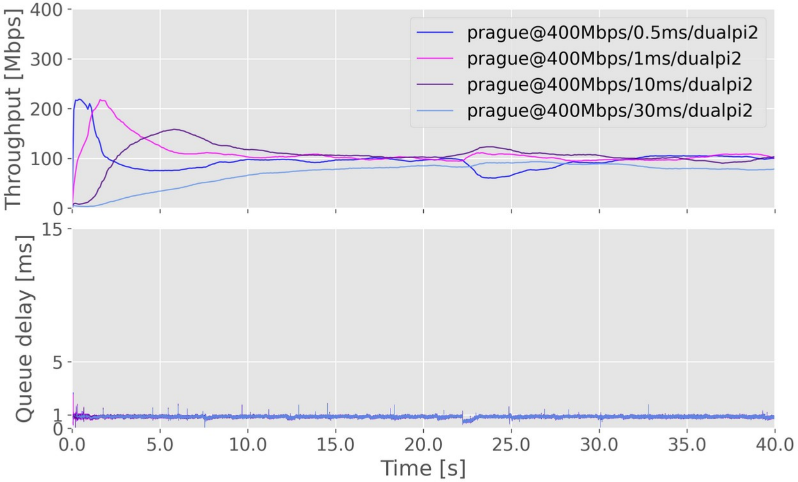
Smoother Throughput, smaller Queue

Prague Benefits today

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

400Mbps 0.5, 1, 10, 30ms





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

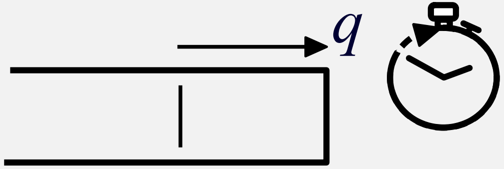
- L4S landing page
  - https://riteproject.eu/dctth/
- TCP Prague mailing list
  - https://www.ietf.org/mailman/listinfo/tcpprague
- Open source code from L4S team
  - Linux kernel code, testbed scripts and GUI visualizer, Prague virtual machine, …
  - https://github.com/L4Steam
  - https://l4steam.github.io/
- 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



if (q > Q) ECN-mark

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

[DCTCP-stability] Alizadeh, M., Javanmard, A., and B. Prabhakar, "Analysis of DCTCP: Stability, Convergence, and Fairness", ACM SIGMETRICS 2011 , June 2011,

Event Cycles

Typical DCTCP implementation