Implementing the 'Prague Requirements' in TCP for L4S

Bob Briscoe (Independent & CableLabs)
Koen De Schepper (Nokia Bell-Labs)
Olivier Tilmans (Nokia Bell-Labs)
Mirja Kuehlewind (ETHZ)
Joakim Misund (Uni Oslo & Simula Research Lab)
Olga Albisser née Bondarenko (Simula Research Lab)
Asad Sajjad Ahmed (Simula Research Lab & Uni Oslo)
## The trick: scalable congestion control

<table>
<thead>
<tr>
<th></th>
<th>① Today (typical)</th>
<th>② Today (at best)</th>
<th>③ Unacceptable</th>
<th>④ L4S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottleneck</td>
<td>Bloated drop-tail buffer</td>
<td>AQM</td>
<td>Shallow AQM</td>
<td>Immediate AQM</td>
</tr>
<tr>
<td>Sender CC</td>
<td>Classic</td>
<td>Classic</td>
<td>Classic</td>
<td>Scalable (tiny saw-teeth)</td>
</tr>
</tbody>
</table>

- **Today (typical)**:
  - **Bottleneck**: Bloated drop-tail buffer
  - **Sender CC**: Classic
  - TCP saw-teeth seeking capacity
  - Link utilization: Full utilization; insensitive to target
  - Buffer occupancy: Less buffer; still enough for bursts
  - **Time**: Less buffer but poor utilization

- **Today (at best)**:
  - **Bottleneck**: AQM
  - **Sender CC**: Classic
  - TCP saw-teeth seeking capacity
  - Link utilization: Shallower target
  - Buffer occupancy: Shallower AQM
  - **Time**: Shallower target; no delay

- **Unacceptable**:
  - **Bottleneck**: AQM
  - **Sender CC**: Classic
  - TCP saw-teeth seeking capacity
  - Link utilization: Shallower AQM
  - Buffer occupancy: Shallower target
  - **Time**: Shallower target; no delay

- **L4S**:
  - **Bottleneck**: Immediate AQM
  - **Sender CC**: Scalable (tiny saw-teeth)
  - TCP saw-teeth seeking capacity
  - Link utilization: Full utilization; insensitive to target
  - Buffer occupancy: Even less buffer; consistently low queueing delay
  - **Time**: Even less buffer; consistently low queueing delay
Implementation status
pasted from https://riteproject.eu/dctth/#code

Source Code

- Dual Queue Coupled AQM
  - with PI2: Linux repo
  - With Curvy RED (TBA)
- TCP Prague
  - Linux repo
- QUIC Prague
  - General repo (should work for Linux, FreeBSD, Windows)
- SCReAM (Self-Clocked Rate Adaptation for Multimedia) a mobile optimised congestion control algorithm for real-time interactive media, with support for L4S
  - General repo

+DOCSIS 3.1
(next slide)

Component parts
- Accurate ECN TCP Feedback (included in TCP Prague above)
  - Linux repo and Linux repo without AccECN TCP Option
- Paced Chirping
  - for Linux (initial proof-of-concept research code)
- Data Centre TCP (DCTCP) for
  - Linux (in the mainline kernel)
  - FreeBSD (in the mainline kernel)
  - ns2 patch

particular thanks to Olivier Tilmans
for pulling together TCP Prague and the Hackathon team
Low Latency DOCSIS 3.1

- Low Latency measures mandatory from Jan’19
  - upstream (Cable Modem) & downstream (CMTS)
    - DOCSIS 3.1 MAC and Upper Layer Protocols i/f (MULPI) Spec (i17+)
    - Cable Modem Operations Support System Interface Spec (i14+)
    - CCAP Operations Support System Interface Specification (i14+)

- Cuts 2 main sources of delay
  - MAC: Request-grant loop
  - Queuing: Mandatory L4S support

- White paper: Low Latency DOCSIS: Technology Overview
  - Also translated into ASCII: draft-white-tsvwg-lld (Informational)

- Certification test plans nearing completion
- Implementation in progress
TCP Prague

- new Linux congestion control module
  - with mandatory use of certain improvements to base TCP
- based off DCTCP scalable congestion controller
  - some improvements available for DCTCP as well
- usable for testing, but still a work in progress
  - Available from: https://github.com/L4STeam/tcp-prague (tcp_prague branch)
  - patch submitted to Linux netdev list this week
  - To load & enable
    sudo modprobe tcp_prague
    sudo sysctl -w net.ipv4.tcp_congestion_control=prague
The 'Prague L4S requirements'

- for scalable congestion controls over Internet
  - Assuming only partial deployment of either FQ or DualQ Coupled AQM isolation for L4S
  - Jul 2015 Prague IETF, ad hoc meeting of ~30 DCTCP folks
    - categorized as safety (mandatory) or performance (optional)
- not just for TCP
  - behaviour for any wire protocol (TCP, QUIC, RTP, etc)
- evolved into IETF conditions for setting ECT(1) in IP

<table>
<thead>
<tr>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4S-ECN Packet Identification: ECT(1)</td>
</tr>
<tr>
<td>Accurate ECN TCP feedback</td>
</tr>
<tr>
<td>Reno-friendly on loss</td>
</tr>
<tr>
<td>Reno-friendly if Classic ECN bottleneck</td>
</tr>
<tr>
<td>Reduce RTT dependence</td>
</tr>
<tr>
<td>Scale down to fractional window</td>
</tr>
<tr>
<td>Detecting loss in units of time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optimizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECN-capable TCP control packets</td>
</tr>
<tr>
<td>Faster flow start</td>
</tr>
<tr>
<td>Faster than additive increase</td>
</tr>
</tbody>
</table>
TCP Prague: status against Prague L4S requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>base TCP</th>
<th>DCTCP</th>
<th>TCP Prague</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4S-ECN Packet Identification: ECT(1)</td>
<td></td>
<td>module option</td>
<td>mod opt default</td>
</tr>
<tr>
<td>Accurate ECN TCP feedback</td>
<td>sysctl option</td>
<td>?</td>
<td>mandatory</td>
</tr>
<tr>
<td>Reno-friendly on loss</td>
<td>inherent?</td>
<td>inherent</td>
<td></td>
</tr>
<tr>
<td>Reno-friendly if classic ECN bottleneck</td>
<td></td>
<td>TBA if nec.</td>
<td></td>
</tr>
<tr>
<td>Reduce RTT dependence</td>
<td></td>
<td>simulated</td>
<td></td>
</tr>
<tr>
<td>Scale down to fractional window</td>
<td>in progress</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Detecting loss in units of time</td>
<td>default RACK</td>
<td>default RACK</td>
<td>mandatory</td>
</tr>
</tbody>
</table>

Optimizations

| ECN-capable TCP control packets                  | sysctl option off | default on | default off → on later |
| Faster flow start                                 | in progress |             |                             |
| Faster than additive increase                    | in progress |             |                             |
Set ECT(1) in IP header

- the L4S packet identifier
  - IPv4 & IPv6
- Added module option to TCP Prague, and DCTCP could duplicate it
  - TCP Prague: might replace DCTCP in a private DC
    sudo modprobe tcp_prague prague_ect=X
  - DCTCP: private DC might need to control which codepoint\(^1\)
    sudo modprobe tcp_dctcp dctcp_ect=X

<table>
<thead>
<tr>
<th>Codepoint</th>
<th>IP-ECN bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-ECT</td>
<td>00</td>
<td>Not ECN-Capable Transport</td>
</tr>
<tr>
<td>ECT(0)</td>
<td>10</td>
<td>Classic ECN-Capable Transport</td>
</tr>
<tr>
<td>ECT(1)</td>
<td>01</td>
<td>L4S ECN-Capable Transport</td>
</tr>
<tr>
<td>CE</td>
<td>11</td>
<td>Congestion Experienced</td>
</tr>
</tbody>
</table>

\(^1\) only once the patch fixing dctcp's response to loss is accepted and response to loss is not disabled
Accurate ECN TCP feedback

- TCP Prague, like DCTCP, needs extent of congestion, not just existence
  - Classic ECN [RFC3168] limits TCP feedback to one ECN-event per RTT
  - DCTCP redefines TCP flags, but no negotiation and unreliable

- AccECN: negotiated reliable f/b of extent of congestion
  - updates the base TCP stack – independent of TCP Prague
  - but TCP Prague depends on both ends having negotiated AccECN

- For testing, if AccECN negotiation fails, can force
  a classic ECN peer to echo each ECN mark once
  - not for production – unreliable delivery of congestion feedback
  
  ```
sudo sysctl -w net.ipv4.tcp_force_peer_unreliable_ece=1
  ```

See Mirja's netdev 2.2 talk: AccECN

[RFC7560], [ietf-AccECN], [AccECN-netdev]
Fall back to Reno-Friendly on loss

- Linux DCTCP bug for last 4 years had no ssthresh reduction on fast re-xmt
  - fixed in TCP Prague, and patch submitted for DCTCP
- Multiplicative decreases of ECN & loss in 'same' RTT?
  - EWMA of ECN, $\alpha$ continues independent of losses
  - each response to a loss episode contrived
to give a compound decrease of $\approx 1/2$
  (ECN reduction) * (loss reduction) $\approx 1/2$
  $(2-\alpha)/2 \times (2+\alpha)/4 \approx 1/2$

EWMA of ECN marking fraction, $\alpha$

ECN rounds
\[\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6\]
loss rounds
\[\times, \times, \times, \times\]
Fall back to Reno-Friendly on Classic ECN

- All academic ECN studies over the years (incl. 2017, 2019) have found virtually no ECN marking
- Would expect to see CE marking as FQ-CoDel deploys

- Are any FIFO Classic ECN routers enabled?
  - Mar 2017: Apple found at least 1 CE /12hr
  - FQ or FIFO?
    - Digging into Apple data
    - Devised FQ v FIFO test
Reduce RTT Dependence

• Why worry?
  1) all TCP's are RTT-dependent anyway
  2) large & small RTT flow in one bottleneck is doubly rare
     - multiple long-running flows in one bottleneck are rare
     - large RTT flows are rare

• Because
  • RTT dependence only appears as queues shrink
  • today's rare traffic mix could be tomorrow's killer app

• Solutions simulated
  • not yet implemented in Linux
Problem

- TCP's min window:
  - 2 segments (due to delayed ACKs)
- with deep queue: no issue
- with shallow AQM target:
  - TCP becomes unresponsive
  - inflates queue to keep 2 pkts in flight / flow
  - ignores AQM's attempts to hold down queue
Scale down to fractional window

Solution

- fractional window implemented in base TCP
  - uses pacing (with ACK clock) for packet conservation
  - keeps AI below MD by scaling to log of ssthresh
- still debugging DCTCP-specific addition:
  - to adjust clocking of DCTCP EWMA
- once mature, will be added to TCP Prague
  - initially as default-off option

- testbed config of above example
  - 20 TCP Reno flows, SMSS 1448B
  - bottleneck link rate: 200Mb/s; base RTT 300µs
  - AQM: RED 0-10% over 1-3ms
Detecting loss in units of time

- like RACK (Recent ACKnowledgements)
- mandatory for L4S
  - L4S queues will know all sources tolerate reordering < x µs
  - softens a hard constraint on hi-speed link design
- TCP Prague implementation currently uses RACK as-is
  - including RACK's 3 DupACK bootstrap at start of flow – in units of packets :
  - alternative approaches to modify RACK bootstrap for TCP Prague
    - bootstrap with SRTT/8 (say)
      using SRTT from 3WHS, dst cache and/or TFO cookie
    - ensure initial window is paced over the RTT,
      so duration of 3 DupACK is invariant as flow-rate scales

[RACK-netdev], [ietf-RACK]
Set IP-ECN to ECT on TCP control packets (ECN++)

- Loss protection for performance-critical pkts
- Already default-on in DCTCP
  - otherwise high background ECN-marking leads to high loss of not-ECT control packets
- Some fall-backs in IETF ECN++ spec still TBA
  - ECN++ default-off while fall-back code matures
  - to enable:
    ```bash
    sudo sysctl -w net.ipv4.tcp_ecn_plus_plus=1
    ```
- ECN++ patches the base TCP stack
  - TCP Prague depends on negotiating AccECN
  - So ECT(1) is allowed on every packet

<table>
<thead>
<tr>
<th>TCP packet type</th>
<th>RFC3168 sender</th>
<th>ECN++ sender</th>
<th>RFC3168 f/b negotiated</th>
<th>AccECN f/b negotiated</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
<td>not-ECT</td>
<td>ECT</td>
<td>not-ECT</td>
<td></td>
</tr>
<tr>
<td>SYN-ACK</td>
<td>not-ECT</td>
<td>ECT</td>
<td>ECT</td>
<td></td>
</tr>
<tr>
<td>Pure ACK</td>
<td>not-ECT</td>
<td>ECT</td>
<td>not-ECT</td>
<td></td>
</tr>
<tr>
<td>Window probe</td>
<td>not-ECT</td>
<td>ECT</td>
<td>ECT</td>
<td></td>
</tr>
<tr>
<td>FIN</td>
<td>not-ECT</td>
<td>ECT</td>
<td>ECT</td>
<td></td>
</tr>
<tr>
<td>RST</td>
<td>not-ECT</td>
<td>ECT</td>
<td>ECT</td>
<td></td>
</tr>
<tr>
<td>Re-XMT</td>
<td>not-ECT</td>
<td>ECT</td>
<td>ECT</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td>ECT</td>
<td>ECT</td>
<td>ECT</td>
<td></td>
</tr>
</tbody>
</table>

1 AccECN feedback ‘requested’ in the case of a SYN

'ECT' means whichever ECN-Capable Transport codepoint is appropriate, whether ECT(0) or ECT(1)
Over-strict ECN negotiation 'bug'

- If a SYN requests ECN at the TCP layer and is already ECN-capable at the IP layer
  - Linux TCP listeners currently disable ECN for the connection
- Rationale: RFC3168 says “A host MUST NOT set ECT on SYN...”
  - so ECT on a SYN could be a symptom of network mangling
  - 'bug': it could also be a symptom of a new standard (it is now: ECN++)
- ECN++ client deployment cannot get started – it disables ECN :( 

- Recent tiny patch for back-porting to all TCP listeners
  - identifies an ECN set-up SYN that's ECN-capable in IP by:
    - flag bits 4-9 == 0b000011
    - not just
    - flag bits 8-9 == 0b11
  - This can distinguish an RFC3168 ECN setup SYN from something newer that allows ECT on a SYN, such as an AccECN setup SYN, which uses
    - flag bits 4-9 == 0b111
Faster flow start without overshoot

- DCTCP is slow to get started
  - alone: hard not to exceed the shallow ECN threshold
  - pushing in: frequent ECN marks hit new flow early
  - early exit from slow start into additive increase

- Why not respond to extent not existence?
  - TCP Prague sender doesn't know if marks are L4S

- Plan to use paced chirping, once stable
  - only a few packets of overshoot
  - typically much faster increase than slow-start
  - primarily delay-based, so universal

- Until paced-chirping is more mature
  - download separately: github.com/JoaikimMisund/PacedChirping
Faster than additive increase

- DCTCP and TCP Prague can exploit high freq ECN
  - rapidly detect when it stops...

- ...then probe for capacity
  - plan to use paced chirping

- Why not use paced chirping all the time (like TCP Rapid)?
  - tried that – unnecessarily raises the noise floor
Faster than AI

- NS-3 simulation
  - Base RTT: 100ms
  - Capacity 50Mbps → 100Mbps

<table>
<thead>
<tr>
<th></th>
<th>RTTs to 95%</th>
<th>Overshoot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cubic</td>
<td>150</td>
<td>20ms</td>
</tr>
<tr>
<td>DCTCP + Pc'd Chirping</td>
<td>7</td>
<td>&lt;1ms</td>
</tr>
<tr>
<td>Requirements</td>
<td>base TCP</td>
<td>DCTCP</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>----------</td>
<td>-------</td>
</tr>
<tr>
<td>L4S-ECN Packet Identification: ECT(1)</td>
<td></td>
<td>module option</td>
</tr>
<tr>
<td>Accurate ECN TCP feedback</td>
<td>sysctl option</td>
<td>?</td>
</tr>
<tr>
<td>Reno-friendly on loss</td>
<td>inherent?</td>
<td>inherent</td>
</tr>
<tr>
<td>Reno-friendly if classic ECN bottleneck</td>
<td></td>
<td>TBA if nec.</td>
</tr>
<tr>
<td>Reduce RTT dependence</td>
<td></td>
<td>simulated</td>
</tr>
<tr>
<td>Scale down to fractional window</td>
<td>in progress</td>
<td></td>
</tr>
<tr>
<td>Detecting loss in units of time</td>
<td>default RACK</td>
<td>default RACK</td>
</tr>
</tbody>
</table>

**Optimizations**

<table>
<thead>
<tr>
<th>Optimizations</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ECN-capable TCP control packets</td>
<td>sysctl option off</td>
<td>default on</td>
<td>default off → on later</td>
</tr>
<tr>
<td>Faster flow start</td>
<td>in progress</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Faster than additive increase</td>
<td>in progress</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance (briefly)

- Low higher percentile delay important
  - for low latency real-time delivery
  - have to measure delay of every packet

- median Q delay: 100-200μs
- 99%ile Q delay: 1-2ms
- ~10x lower delay than best 2nd gen. AQM
  - at all percentiles

- when really hammering each AQM
  - long-running TCPs: 1 ECN 1 non-ECN
  - web-like flows @ 300/s ECN, 300/s non-ECN
    - exponential arrival process
    - file sizes Pareto distr. α=0.9 1KB min 1MB max
  - 120Mb/s 10ms base RTT
summary

• L4S
  • Frequent L4S-ECN markings give leaps in performance
    Low Latency
    Low Loss
    Scalable throughput
  • a set of incremental changes to network & hosts

• TCP Prague
  • a set of incremental changes to TCP or DCTCP
  • with radical results
  • most are desirable in themselves
Implementing the 'TCP Prague' Requirements for L4S

Q&A

and spare slides
FAQs

Q. Why is IP-ECN=X1 classified into L4S queue?
   This will classify CE as L4S, which could have started as ECT(0) (non-L4S) or ECT(1) (L4S) but been marked as CE by an upstream AQM.

A. Reordering a few packets with lower delay is the safe way round.

Q. Isn't the square-root only for TCP Reno, not Cubic?
A. Typical scenarios: Cubic still in Reno-compatibility mode (see next slide).

Q. Isn't an ECN-capable SYN over the Internet a flooding attack vulnerability?
A. No. ECN AQMs disable ECN marking under persistent overload (IETF requirement).

Q. Does fq_CoDel fit into the picture?
A. Yes. As-is, fq_CoDel only gives low latency to low bandwidth flows.
   But it could include the last ECN bit in its flow classifier, and implement a simple immediate AQM in any ECT(1) queue.

Q. What about BBR?
A. BBR & L4S have complementary goals, and could be combined into one compound congestion control.
   TCP Prague gives much lower delay than BBR, and is much more responsive to other traffic, but where there is no L4S support in the network TCP Prague is merely no worse than Reno.
   BBR addresses the latter case, where there is no AQM in the network. So if L4S could fall back to BBR when it detected no L4S support in the network, the user would see lower delay everywhere, and still get much better performance where there was network support for L4S.

<table>
<thead>
<tr>
<th>Codepoint</th>
<th>IP-ECN bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-ECT</td>
<td>00</td>
<td>Not ECN-Capable Transport</td>
</tr>
<tr>
<td>ECT(0)</td>
<td>10</td>
<td>Classic ECN-Capable Transport</td>
</tr>
<tr>
<td>ECT(1)</td>
<td>01</td>
<td>L4S ECN-Capable Transport</td>
</tr>
<tr>
<td>CE</td>
<td>11</td>
<td>Congestion Experienced</td>
</tr>
</tbody>
</table>
FAQs

• What about the SCE proposal for ECT(1)?
  • Caveat: reverse engineered from unclear initial draft

• L4S incremental deployment:
  • ECN used as classifier to isolate low latency flows
  • works with DualQ or per-flow Q
  • CE → L4S even if originally ECT(0)

• SCE incremental deployment:
  • ECN not used as classifier
  • need another classifier to isolate low latency flows
  • so only works with per-flow Q
ECN transitions

- RFC3168 & RFC8311
  - ECT(0) → CE
  - ECT(1) → CE

- RFC6040 added support for RFC6660
  - ECT(0) → ECT(1)

- Many encapsulations will still be pre-RFC6040
  - decap will revert ECT(1)

- Ambiguity of CE
  - ECT(0) → CE early on path
    CE → L4S queue later on path
  - 5 unlikely scenarios have to coincide
to cause an occasional spurious re-xmt

<table>
<thead>
<tr>
<th>incoming inner</th>
<th>incoming outer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-ECT</td>
<td>Not-ECT</td>
</tr>
<tr>
<td>ECT(0)</td>
<td>ECT(0)</td>
</tr>
<tr>
<td>ECT(1)</td>
<td>ECT(0)</td>
</tr>
<tr>
<td>CE</td>
<td>CE</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outgoing header (RFC4301 \ RFC3168)

<table>
<thead>
<tr>
<th>incoming inner</th>
<th>incoming outer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-ECT</td>
<td>Not-ECT</td>
</tr>
<tr>
<td>ECT(0)</td>
<td>ECT(0)</td>
</tr>
<tr>
<td>ECT(1)</td>
<td>ECT(1)</td>
</tr>
<tr>
<td>CE</td>
<td>CE</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outgoing header (RFC6040)
(bold = change for all IP in IP)
## Distinguishing features of L4S & SCE

<table>
<thead>
<tr>
<th></th>
<th>L4S</th>
<th>SCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of queues (see prev slide)</td>
<td>• either dual-queue or per-flow queue</td>
<td>• per-flow queue only</td>
</tr>
<tr>
<td>Legacy flow coexistence at:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• non-ECN bottleneck</td>
<td>• simple to fall back to classic</td>
<td>• simple to fall back to classic</td>
</tr>
<tr>
<td>• FQ classic ECN bottleneck</td>
<td>• no prob</td>
<td>• no prob</td>
</tr>
<tr>
<td>• 1Q classic ECN bottleneck</td>
<td>• requires heuristic to fall back to classic</td>
<td>• simple to fall back to classic</td>
</tr>
<tr>
<td>ECN TCP feedback req’d from recvr</td>
<td>• sndr can force classic ECN rcvr to give fine-grained feedback (unreliable delivery)</td>
<td>• No ECT(1) feedback in TCP without new AccECN TCP option</td>
</tr>
<tr>
<td>Additional link benefits</td>
<td>• Enables links to optimize for new sender behavior (e.g. out-of-order tolerance)</td>
<td></td>
</tr>
<tr>
<td>Congestion control status</td>
<td>• DCTCP - widespread deployment experience</td>
<td>• A high-level idea</td>
</tr>
</tbody>
</table>
Low delay *and* high throughput

- ~10x lower delay than state-of-the-art AQMs
- with capacity-seeking behaviour – TCP-like and adaptive real-time
- for all packets, not just a 'low delay' class

- IETF's experimental track L4S architecture
  
  Low Latency
  Low Loss
  Scalable throughput

- Also adopted in DOCSIS 3.1

[RFC8311], [ietf-l4s-arch], [LLDOCSIS-spec], [LLDOCSIS-overview]
Ultra-low latency for every application

- Not only non-queue-building traffic
  - DNS, gaming, voice, SSH, ACKs, HTTP requests, etc
- Capacity-seeking traffic as well
  - TCP, QUIC, RMCAT for WebRTC
  - web, HD video conferencing, interactive video, cloud-rendered virtual reality, augmented reality, remote presence, remote control, interactive light-field experiences, ...
DualQ Coupled AQM
latency isolation, but bandwidth pooling

- L4S-ECN: senders set ECT(1) → classifies into L4S queue

<table>
<thead>
<tr>
<th>Codepoint</th>
<th>IP-ECN bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-ECT</td>
<td>00</td>
<td>Not ECN-Capable Transport</td>
</tr>
<tr>
<td>ECT(0)</td>
<td>10</td>
<td>Classic ECN-Capable Transport</td>
</tr>
<tr>
<td>ECT(1)</td>
<td>01</td>
<td>L4S ECN-Capable Transport</td>
</tr>
<tr>
<td>CE</td>
<td>11</td>
<td>Congestion Experienced</td>
</tr>
</tbody>
</table>

see Olga's later talk: DualPI² qdisc

[RFC8311], [ietf-ecn-l4s-id], [dualpi2-netdev], [ietf-dualq-aqm]
DualQ Coupled AQM

latency isolation, but bandwidth pooling

- flow rate 'fairness' across the two queues
  ① classic congestion control (TCP & QUIC): rate depends on the square root of the drop level
  ② counterbalanced by the squaring

$ r_L \propto \frac{1}{p_L} $  
$ r_C \propto \frac{1}{\sqrt{p_C}} $  

- no flow ID inspection, no bandwidth priority

- $ r_L \propto \frac{1}{p_L} $  
  - page rate per flow
- $ p_L $: drop or marking probability

- $ r_C \propto \frac{1}{\sqrt{p_C}} $  
  - page rate per flow
- $ p_C $: drop or marking probability

- $ L $: L4S AQM ECN marking
- $ C $: Classic AQM drop/marking

- L4S Classifier
- Classic Classifier
- Scalable sender
- Classic sender

- Conditional priority scheduler

- see Olga's later talk: DualPI² qdisc

- [RFC8311], [ietf-ecn-l4s-id], [dualpi2-netdev], [ietf-dualaqm]
Potential Minor change to FQ-CoDel to support L4S

- Don't need to couple AQMs if you have FQ
- FQ-CoDel target used to be lower for ECN-capable flows
- A simple patch could use a shallow target for ECT(1) flows
  - L4S AQM is stateless: just immediate shallow threshold ECN marking
- Open question
  - Whether/how to schedule L4S flows with latency priority but not bandwidth priority