Low Latency, Low Loss, Scalable Throughput (L4S)

Problem Statement
draft-briscoe-tsvwg-aqm-tcpm-rmcat-l4s-problem

Bob Briscoe*
Koen De Schepper†
Marcelo Bagnulo Braun§
July 2016

* Simula Research Laboratory
† NOKIA Bell Labs
§ Universidad Carlos III de Madrid
The application performance problem

- increasingly *all* of a user's apps at one time require low delay
  - online gaming
  - voice
  - conversational video, interactive video
  - virtual reality, augmented reality
  - instant messaging
  - interactive Web, Web services
  - remote desktop, cloud-based apps

- Need a new service for *all* Internet traffic to transition to

- caches have cut base (propagation) delay, where they can
- queuing remains a major component of delay, albeit intermittent
  - under load, delay roughly doubles,
    even with state-of-the-art queue management tuned for your current base RTT
The deployment incentive problem

• Significantly better – not just incremental
  – worth the deployment hassle
  – enable valuable new products and services
The technology problem

- More access bandwidth?
  - does not address queuing delay

- Differentiated services (Diffserv)?
  - only cuts delay for some packets at the expense of others

- Per-flow queuing?
  - isolates one flow from the delay of another, but not from its own
  - requires L4 header inspection and significant processing expense

- Active Queue Management (AQM)?
  with Explicit Congestion Notification (ECN)?
  - on the right track, but the root problem is beyond AQM control...

- …'Classic' TCP (not the network) determines queue delay
  - to go faster, its saw-teeth get bigger (unscalable)

New word: Classic TCP =
Reno congestion control [RFC5681] & friends: Cubic, SCTP, QUIC, etc.
The Classic TCP dilemma: delay vs. utilization

Today (typical)
TCP on end-systems
Drop-tail buffers

TCP saw-teeth
seeking capacity

buffer occupancy

AQM operating point

Today (at best)
TCP on end-systems
AQM at bottlenecks

TCP saw-teeth
seeking the operating point

buffer size

shallower operating point

A Scalable TCP
Resolves dilemma
Have to change TCP

buffer kept for bursts

queuing delay consistently lower

more smaller saw-teeth

full line utilisation

cuts delay, but poorer line utilization

sawtooth amplitude:
~1 'typical' base RTT (round trip time)
Actually, it's a Hexlemma

- Three impairments:
  - Queuing delay
  - Under-utilization
  - Packet loss

- Three wider issues:
  - Layering violation
  - Unscalable
  - RTT-unfairness

- If AQM reduces one, TCP increases the others

New word: hexlemma = like a dilemma, but between six things
Fine saw-teeth are only feasible...

1) if drop is not used as the congestion signal
   • drop would be too frequent
   • need explicit congestion notification (ecn)
     – and not “the same as drop” [RFC3168], otherwise coarse saw-teeth

2) if the 'coexistence problem' is solved
   • one 'Scalable' flow with frequent sawteeth looks like many 'Classic' flows to a 'Classic' TCP flow
   • so the Classic flow starves itself
Problem: very high level summary

- Problem: Classic TCP is the elephant in the room
- Solution: build another room without the elephant
Solution: very high level summary

- Problem: Classic TCP is the elephant in the room
- Solution: build another room without the elephant
Coexistence: Solution Architecture

- DualQ Coupled AQMs: a 'semi-permeable membrane' that:
  - partitions latency (separate queues for L4S & Classic)
  - but pools bandwidth (shared by apps/transport, not by network)

- per 'site' (home, office, campus or mobile device)
  - typically one access bottleneck in each direction
  - deploying DualQ here should give nearly all the benefit
Coexistence: Solution Architecture

- DualQ Coupled AQMs: a 'semi-permeable membrane' that:
  - partitions latency (separate queues for L4S & Classic)
  - but pools bandwidth (shared by apps/transport, not by network)

$$r \propto \frac{1}{p}$$
$$r \propto \frac{1}{\sqrt{p}}$$

$r$: packet rate
$p$: drop/mark probability
Framework for Diverse Solutions

- The DualQ Coupled AQM draft is structured as a framework
  - pseudocode of concrete examples in the appendices

OK to include some unresponsive VoIP, DNS, etc. in L4S

Scalable sender

Classic sender

L4S: [X1]

ECN Classifier

Coupling

marker

drop or marking

priority scheduler

many scalable algorithms already:

DCTCP, Relentless, S-SCREAM, etc.
(not yet with safety features–see later)

Classify at IP layer or lower

2 different classic AQMs already implemented:

Curvy RED & PIE

2 different schedulers already implemented:

strict priority & MEDF
very high level summary

- problem: Classic TCP is the elephant in the room
- solution: build another room without the elephant
Q&A

large saw teeth can ruin the quality of your experience
Why is performance so much better? Immediate signalling

- Today's AQMs defer drop for \(~100\text{ms}\)
  1) to allow time for a worst-case RTT response
     because: the network doesn't know each packet's RTT
  2) to avoid drop unless the queue proves persistent
     because: drop is an impairment as well as a signal

- Using ECN for L4S makes it feasible to signal immediately
  - because ECN is a signal but not an impairment

Problem with the Classic approach: a flow with RTT=5ms gets no signal for 20 round-trips
related problems L4S also addresses

- incremental deployment of low delay DCTCP
  - within & between data centres with no unified control
- near-zero congestion loss
  - for short flows, loss translates to timeout and retransmit delay
- incremental deployment of scalable congestion controls
  - 'Scalable' = invariant recovery time
  - TCP Reno [RFC5681]: unscalable
  - TCP Cubic: less unscalable

![Graph showing recovery time and window size comparison](image)

Better