‘Data Centre to the Home’: Deployable Ultra-Low Queuing Delay for All

Koen De Schepper[†] Olga Bondarenko[*] [‡] Ing-Jyh Tsang[†] Bob Briscoe[‡]
[†]Nokia Bell Labs, Belgium
[†]{koen.de_schepper|ing-jyh.tsang}@nokia.com
[‡]Simula Research Laboratory, Norway
[‡]{olgabo|bob}@simula.no

ABSTRACT

Traditionally, ultra-low queueing delay and capacity-seeking are considered mutually exclusive. We introduce an Internet service that offers both: Low Latency Low Loss Scalable throughput (L4S). Therefore it can incrementally replace best efforts as the default service. It uses ‘Scalable’ congestion controls, e.g. Data Centre TCP. Under a wide range of conditions emulated on a testbed using real residential broadband equipment, it proved hard not to get remarkably low (sub-millisecond) average queuing delay, zero congestion loss and full utilization. To realize these benefits we had to solve a hard problem: how to incrementally deploy controls like DCTCP on the public Internet. The solution uses two queues at the access link bottleneck, for Scalable and ‘Classic’ (Reno, Cubic, etc.) traffic. It is like a semi-permeable membrane, isolating their latency but coupling their capacity into a single resource pool. We implemented this ‘DualQ Coupled AQM’ as a Linux qdisc to test our claims. Although Scalable flows are much more aggressive than Classic, the AQM enables balance (TCP ‘fairness’) between them. However, it does not schedule flows, nor inspect deeper than IP. L4S packets are identified using the last ECN codepoint in the IP header, which the IETF is in the process of allocating.

CCS Concepts

•**Networks** → **Cross-layer protocols; Packet scheduling; Network performance analysis; Public Internet;** *Network resources allocation;*

*The first two authors contributed equally

Under submission. Please do not distribute.
© 2017, the authors/owners

ACM ISBN .
DOI:

Keywords

Internet, Performance, Queuing Delay, Latency, Scaling, Algorithms, Active Queue Management, AQM, Congestion Control, Congestion Avoidance, Congestion Signalling, Quality of Service, QoS, Incremental Deployment, TCP, Evaluation

1. INTRODUCTION

With increases in bandwidth, latency is becoming the critical performance factor for many, if not most, applications, e.g. Web, voice, conversational and interactive video, finance apps, online gaming, cloud-based apps, remote desktop. Latency is a multi-faceted problem that has to be tackled on many different fronts [9] and in all the different stages of application delivery—from data centres to access links and within end systems.

The aspect this paper addresses is the variable delay due to queuing. Even state-of-the art Active Queue Management (AQM) [38, 23] can only bring this down to roughly the same order as a typical base round-trip delay. This is because bottlenecks are typically in the most numerous edge access links where statistical flow multiplexing is lowest. And a single TCP flow will underutilize a link unless it can buffer about a round trip flight of data.

Queuing delay is intermittent, only occurring when a sufficiently long-running capacity-seeking flow (e.g. TCP) happens to coincide with interactive traffic [24]. However, intermittent delays dominate experience, and many real-time apps adapt their buffering to these intermittent episodes.

Our main contribution is to keep queueing delay extremely low (sub-millisecond) for *all* of a user’s Internet applications. A differentiated service (Diffserv) class such as EF [15] can provide low delay if limited to a fraction of the link’s traffic. Instead, we propose a new service that accommodates ‘greedy’ (capacity-seeking) applications that want both full link utilization and low queuing delay, so it can incrementally replace the default best efforts service. The new service effectively removes congestion loss as well, so it is called Low Latency, Low Loss, Scalable throughput (L4S).

L4S works because senders use one of the family of

'Scalable' congestion controls (§ 2.1 for the rationale). In contrast, we use the term 'Classic' for controls like TCP Reno and Cubic, where control becomes slacker as rate scales.

For evaluation we configure the host OS to use Data Centre TCP (DCTCP [1]), which is a widely available scalable control. We emphasize that the L4S service is not just intended for DCTCP, but also for a range of Scalable controls, e.g. Relentless TCP [34] and future scalable variants of QUIC, SCTP, real-time protocols, etc. In order to test one change at a time, we focus this paper on network-only changes, and use DCTCP, 'as is'. Our extensive experiments over a testbed using real data-centre and broadband access equipment and models of realistic traffic strengthen confidence that DCTCP would work very well over the public Internet.

However, DCTCP will need some safety (and performance) enhancements for production use, so a large group of DCTCP developers has informally agreed a list dubbed the 'TCP Prague' requirements (§ 5.2) to generalize from the otherwise confusing name.

Our second contribution is a solution to the deployability of Scalable controls like DCTCP. It is a common misconception that DCTCP is tailored for data centres, but the name merely emphasizes that it should not be deployed outside a controlled environment; it is too aggressive to coexist with existing 'Classic' traffic so a single admin is expected to upgrade all senders, receivers and bottlenecks at once.

We propose the 'Dual Queue Coupled AQM' that can be incrementally added at path bottlenecks to solve this 'coexistence' problem. It acts like a semi-permeable membrane. For delay it uses two queues to isolate L4S traffic from the Classic queue. But for throughput, the queues are coupled to appear as a single resource pool. So, for n aggressive L4S flows and m TCP-friendly Classic flows, each flow gets roughly $1/(n+m)$ of the capacity. The high-level idea of coupling is that the L4S queue emits congestion signals more aggressively to counterbalance the more aggressive response of L4S sources.

Balance between microflows should be a policy choice not a network default (§ 2.3), so we enable but do not enforce it. And coexistence between DCTCP and Classic flows is achieved without the network inspecting flows (no deeper than the IP layer). We have also tested that the L4S service can cope with a reasonable proportion of unresponsive traffic, just as best efforts copes with unresponsive streaming, VoIP, DNS etc.

The two queues are for transition, not scheduling priority. So low L4S delay is not at the expense of Classic performance and delay remains low even if a high load of solely L4S traffic fills the link.

Given access networks are invariably designed to bottleneck in one known location, the AQM does not have to be deployed in every buffer. Most of the benefit can be gained by deployment at the downstream queue into the access link, and home gateway deployment addresses the upstream. § 5 discusses how a Scalable control like DCTCP falls back to Reno if it encounters a non-L4S bottleneck. It also discusses wider deployment considerations, including other deployment scenarios such as coexistence between DCTCP and Classic TCP in heterogeneous or interconnected data centres.

L4S faces a very similar deployment problem to classic Explicit Congestion Notification (ECN [39]). However, we have learned from the ECN experience. To overcome the risk a first mover faces in kick-starting a multi-party deployment, we have attempted to ensure that the performance gain is dramatic enough to enable valuable new applications, not just a relatively marginal performance improvement.

The dramatic improvement of L4S has been demonstrated [7] by simultaneously running many apps that are both bandwidth-hungry and latency-sensitive over a regular 40Mb/s broadband access link. Two apps transmitted a user's physical movements (virtual reality goggles and pan/zoom finger gestures on a panoramic interactive video display) to cloud-based video servers over a broadband access (base delay 7 ms). The queuing delay of every packet was so low that the scenes that were generated on the fly and streamed back to the user seemed as if they were local and natural. Whereas without L4S, there was considerable lag and jerkiness. Other users were downloading streaming video, bulk files and running a gaming benchmark, all in the same queue, and mean per-packet queuing delay was around 500 μs.

Our third contribution is to ensure that the low queuing delay of L4S packets is preserved during overload from either L4S or Classic traffic, and neither can harm the other more than they would in a single queue.

Our fourth contribution is to ensure that the AQM can be deployed in any public Internet access network with zero configuration.

Our fifth contribution is extensive quantitative evaluation of the above claims: i) dramatically reduced delay and variability without increasing other impairments; ii) 'do no harm' to Classic traffic; iii) window balance between competing Scalable and Classic flows; and iv) overload handling (see § 4).

2. RATIONALE

2.1 Why a Scalable Congestion Control?

A congestion controller is defined as 'Scalable' if the rate of congestion signals per round trip, c, does not decrease as bandwidth-delay product (BDP or window) scales. The flow rate saw-tooths down whenever a congestion signal is emitted. So, by definition, the average sawtooth duration (a.k.a. recovery time) of a scalable control does not grow as rate scales.

TCP Cubic scales better than Reno, but it is still not fully scalable. For instance, for every 8-fold rate increase the average Cubic sawtooth duration doubles while its amplitude increases 8-fold, which is the cause of growing delay variation. For instance, between 100

and 800 Mb/s, Cubic's sawtooth recovery time expands from 250 round trips to 500 round trips (assuming base RTT=20 ms). In contrast, whatever the rate, the average recovery time of a DCTCP sawtooth remains invariant at just half a round trip.

We use a Scalable congestion control because, unlike Classic TCP algorithms, this implies:

1. control does not slacken as the window scales;
2. variation of queuing and/or under-utilization, need not increase with scale (Figure 1).

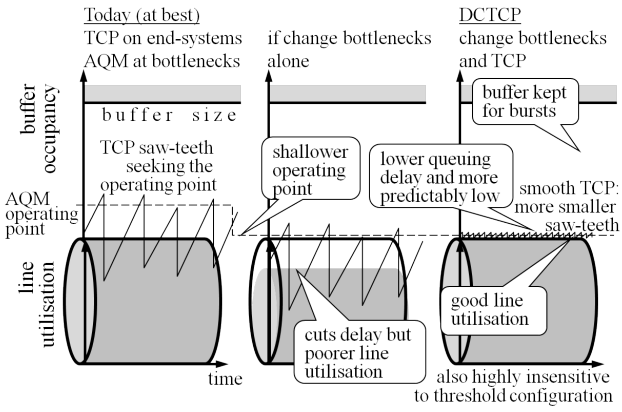

Figure 1: Data Centre TCP: Intuition

In the steady state, the number of signals per round is the product of segments per round W and the probability p that a segment carries a signal, i.e. $c = pW$. Formulas for the steady-state window, W, can be derived for each congestion controller (see § 3.2). Each formula is of the form $W \propto 1/p^B$, where B is a characteristic constant of the algorithm [4] (e.g. $B = 1/2$ for TCP Reno). So it is straightforward to state the above scalability condition in terms of B by substituting for p in the above formula for c:

$$c \propto W^{(1 - 1/B)}.$$

Therefore, $B \geq 1$ defines a control as Scalable.

For DCTCP, $B \geq 1$, and DCTCP with probabilistic marking has $B = 1$ (see § 3.2) so the steady-state signalling rate is scale-invariant. However, the dynamic behaviour of DCTCP is not scalable, e.g. DCTCP's window update algorithm is unscalable by the definition in [29]. However, our AQM supports any control with a steady state that is scalable, so we are confident that solutions to DCTCP's problems (e.g. [44]) will be able to evolve and co-exist with today's DCTCP, without a need for further network changes.

2.2 Why ECN?

Explicit Congestion Notification (ECN [39]) is purely a signal, whereas drop is both an impairment and a signal, which compromises signalling flexibility. ECN is essential to the L4S service, because:

1. A Scalable control's finer (more aggressive) sawteeth imply a higher signalling rate, which would be untenable as loss, particularly during high load;
2. If the queue grows, a drop-based AQM holds back from introducing loss in case it is just a sub-RTT burst, whereas it can emit ECN immediately, because it is harmless.

This last point significantly reduces typical signalling delay, because with drop, the network has to add smoothing delay but it does not know each flow's RTT, so it has to smooth over a worst-case (inter-continental) RTT, to avoid instability for worst-case RTT flows. Whereas, the sender knows its own RTT, which it can use as the appropriate time constant to smooth the network's unsmoothed ECN signals [2] (and it can choose to respond without smoothing, e.g. in slow start).

Therefore, we require that Scalable traffic is ECN-capable, which we can also use to classify Scalable packets into the L4S queue (see § 3).

Irrespective of L4S, ECN also offers the obvious latency benefit of near-zero congestion loss, which is of most concern to short flows [40]. This removes retransmission and time-out delays and the head-of-line blocking that a loss can cause when a single TCP flow carries a multiplex of streams.

2.3 Why Not Per-Flow Queues?

Superficially, it might seem that per-flow queuing (as in FQ-CoDel) would fully address queuing delay; it is designed to isolate a latency-sensitive flow from the delays induced by other flows. However, that does not protect a latency-sensitive flow from the saw-toothing queue that a Classic TCP flow will still inflict upon *itself*. This is important for the growing trend of interactive video-based apps that are both extremely latency-sensitive and capacity-hungry, e.g. virtual and augmented reality, remote presence.

It might seem that self-inflicted queuing delay should not count. To avoid delay in a dedicated remote queue, a sender would have to hold back the data, causing the same delay, just in a different place. It seems preferable to release the data into a dedicated network queue; then it will be ready to go as soon as the queue drains.

However, this logic applies i) if and only if the sender somehow knows that the bottleneck in question implements per-flow queuing and ii) only for non-adaptive applications. Modern applications, e.g. HTTP/2 [5] or the panoramic interactive video app described in § 1, suppress lower priority data, depending on the progress of higher priority data sent already. To adapt how much they send, they need to maintain their self-induced send-queue locally, not remotely; because once optional data is in flight, they cannot suppress it.

As well as not solving self-induced latency, there are further well-rehearsed arguments against per-flow scheduling: i) it cannot know whether flow rate variations are deliberate, e.g. complex video activity ; ii) it cannot know (without prohibitive complexity) whether a flow using more, or less, than an equal share of a user's own capacity is intentional, or even mission-critical; iii)

it needs to inspect transport layer headers (preventing transport evolution); and iv) it requires many more queues and supporting scheduling structures.

Therefore we aim to reduce queuing delay without per-flow queuing. That does not preclude adding a per-flow policer, as a separate policy option.

3. SOLUTION DESIGN

The solution will be explained in two passes. The first pass introduces the overall structure (§3.1). Then details of each aspect are given in a second pass, specifically: how coexistence between L4S and Classic flows is achieved (§3.2); how the low latency of the L4S service is isolated from Classic (§3.3); how overload is handled (§3.4); and an overview of the whole implementation in pseudocode (§3.5).

3.1 Solution Structure

L4S and Classic traffic have opposing delay requirements. The first design goal of L4S traffic is ultra-low queuing delay. Whereas, if the number of flows at the bottleneck is small, Classic congestion controllers (CCs) need a significant queue to avoid under-utilization. One queue cannot satisfy these opposing goals, so we use two separate queues.

Packets are classified between the two queues based on the 2-bit ECN field in the IP header. Classic sources set the codepoints 'ECT(0)' or 'Not-ECT' depending on whether they do or do not support standard ('Classic') ECN [39]. L4S sources ensure their packets are classified into the L4S queue by setting 'ECT(1)', which is an experimental ECN codepoint being redefined for L4S (see §5.1).

An L4S CC such as DCTCP achieves low latency, low loss and low rate variations by driving the network to give it frequent ECN marks. A Classic CC (TCP Reno, Cubic, etc.) would starve itself if confronted with such frequent signals.

So the second design goal is coexistence between Classic and L4S congestion controllers [26], meaning rough balance between their steady-state packet rates per RTT (a.k.a. TCP-fairness or TCP-friendliness). Therefore, we couple the congestion signals of the two queues and reduce the intensity for Classic traffic to compensate for its stronger response to each signal, in a similar way to the single-queue coupled AQM in [16].

Introducing two queues creates a new problem: how often to schedule each queue. We do not want to schedule based on the number of flows in each, which would introduce all the problems of per-flow queuing (§2.3). Instead, we allow the end-systems to 'schedule' themselves in response to the congestion signals from each queue. However, whenever there is contention we give the L4S queue priority, because L4S sources can tightly control their own delay. Nonetheless, to prevent Classic starving, priority is conditional on the delay difference between the two queues.

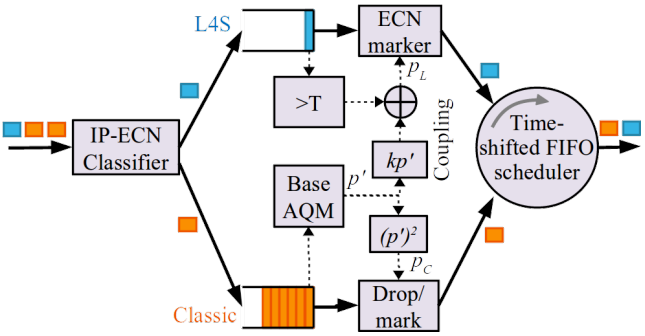

Figure 2: Dual Queue Coupled AQM

The schematic in Figure 2 shows the whole DualQ Coupled AQM. with the classifier and scheduler as the first and last stages. In the middle, each queue has its own native AQM that determines dropping or marking even if the other queue is empty.

3.2 Coupled AQM for Window Balance

To support co-existence between the Classic (C) and L4S (L) congestion control families, we start with the equations that characterize the steady-state window, W, of each as a function of the loss or ECN-marking probability p. Then, like [16], we set the windows to be equal to derive the coupling relationship between the congestion signals for C and L.

We use Reno and DCTCP for C and L. We use Reno because it is the worst case (weakest). We can ignore dynamics, so we use the simplified Reno equation from [35]. For L4S, we do not use the equation from the DCTCP paper [1], which is only appropriate for step marking. Instead, we use the DCTCP equation that is appropriate to our coupled AQM, where marking is probabilistic, as derived in Appendix A of [16]. For balance between the windows, $W_{\text{reno}} = W_{\text{dc}}$, which becomes (1) by substituting from each window equation. Then we rearrange into a generalized relationship for coupling congestion signals in the network (2):

$$\sqrt{\frac{3}{2p_{\text{reno}}}} = \frac{2}{p_{\text{dc}}} \qquad (1) \qquad\qquad p_C = \left(\frac{p_L}{k}\right)^2, \qquad (2)$$

where coupling factor $k = 2\sqrt{2/3} = 1.64$ for Reno.

Appendix A of [16] shows that TCP Cubic [22] will be comfortably within its Reno compatibility mode for the 'Data Centre to the Home' scenarios that are the focus of this paper. The coupling formula in (2) also applies when the Classic traffic is TCP Cubic in Reno mode ('CReno'), except it should use $k = 2/1.68 = 1.19$.

To avoid floating point division in the kernel we round to $k = 2$. In all our experiments this proves to be a sufficiently accurate compromise for any Reno-friendly CC. It gives a slight window advantage to Reno, and a little more to CReno. However, any L4S source gives itself a counter-advantage by virtue of its shallower queue. So L4S achieves a higher packet rate with the same window because of it lower RTT. We do not expend effort coun-

tering this rate imbalance in the network—the proper place to address this is to ensure L4S sources will be less RTT-dependent (see § 5.2).

The coupling is implemented by structuring the AQM in two stages (Figure 2). First what we call a 'Base AQM' outputs the internal probability p'. Then p' is transformed depending on which traffic it is applied to. For Classic traffic it is squared, $p_C = (p')^2$. Whereas for L4S traffic it is applied linearly, $p_L = k*p'$. Substituting for p' from the latter to the former proves that p_L and p_C will be coupling by equation (2) as required.

Diversity of Base AQMs is possible and encouraged. Two have been implemented and tested [17]: a variant of RED and a proportional integral (PI) AQM. Both control queuing time not queue size, given the rate of each queue varies considerably [33, 37]. This paper uses the latter, because it performs better.

[16] also couples two AQMs to enable coexistence of different CCs, but within one queue, not across two. It proves theoretically and experimentally that a PI controller is a robust base AQM. It directly controls a scalable control like DCTCP (rate proportional to $1/p'$). And it shows that squaring the output of a PI controller is a more effective, more principled and simpler way of controlling TCP Reno (rate proportional to $1/\sqrt{p'}$) than PI Enhanced (PIE [38]). It shows that the piecewise lookup table of scaling values used by PIE was just a heuristic way of achieving the same effect as squaring.

3.3 Dual Queue for Low Latency

Often, there will only be traffic in one queue, so each queue needs its own native AQM. The L4S queue keeps delay low using a shallow marking threshold (T), which has already been proven for DCTCP. T is set in units of time [33, 3] with a floor of two packets, so it auto-tunes as the dequeue rate varies. On-off marking may [13] or may not [32, §5] be prone to instability. But to test one change at a time we deferred this to future research.

If there is traffic in both queues, an L4S packet can be marked either by its native AQM or by the coupled AQM (see the OR symbol in Figure 2). However, the coupling ensures that L4S traffic generally only touches the threshold when it is bursty or if there is insufficient Classic traffic.

Note that the L4S AQM emits ECN marks immediately and the sender is expected to do any necessary smoothing. Whereas Classic congestion signals are subject to smoothing delay in the network.

We use what we call a time-shifted FIFO scheduler [36] to decide between the head packets of the two queues. It selects the packet with the earliest arrival timestamp, after subtracting a constant timeshift to favour L4S packets. Normally, this behaves like a strict priority scheduler, but an L4S packet loses its priority if the extra delay of the leading Classic packet exceeds the timeshift. This protects Classic traffic from unresponsive L4S traffic or long L4S bursts, even ensuring a new Classic flow can break into a standing L4S queue.

3.4 Overload Handling

Having introduced a priority scheduler, during overload we must ensure it does no more harm to lower priority traffic than a single queue would.

Unresponsive traffic below the link rate just subtracts from the overall capacity, irrespective of whether it classifies itself as low (L4S) delay or regular (Classic) delay. Then the coupled AQM still enables other responsive flows to share out the remaining capacity by inducing the same balanced drop/mark probability as they would in a single queue with the same capacity subtracted.

To handle excessive unresponsive traffic, we simply switch the AQM over to using the Classic drop probability for both queues once the L4S marking probability saturates at 100%. By equation (2) this occurs once drop probability reaches $(100\%/k)^2$, which is 25% if $k = 2$. When a DCTCP source detects a drop, it already falls back to classic behaviour, so balance between flow rates is preserved.

The native L4S AQM also continues to ECN-mark packets whenever its queue exceeds the threshold, so any responsive L4S traffic maintains the ultra-low queuing delay of the L4S service.

If there are no packets in the Classic queue, the base AQM continues to evolve p' using the L4S queue. As soon as something starts to overload the L4S queue, this ensures the correct level of drop, given L4S sources fall back to a Classic response on detecting a drop. Nonetheless, with solely normal L4S sources, the L4S queue will stay shallow and drive the contribution from the base AQM ($k * p'$) to zero.

3.5 Linux qdisc Implementation

Algorithm 1 Enqueue for Dual Queue Coupled AQM

```
1: STAMP(pkt)                                  ▷ Attach arrival time to packet
2: if LQ.LEN() + CQ.LEN() >L then
3:     DROP(pkt)                               ▷ Drop packet if Q is full
4: else
5:     if LSB(ECN(pkt))==0 then                ▷ Not ECT or ECT(0)
6:         CQ.ENQUEUE(pkt)                     ▷ Classic
7:     else                                    ▷ ECT(1) or CE
8:         LQ.ENQUEUE(pkt)                     ▷ L4S
```

Algorithm 2 Dequeue for Dual Queue Coupled AQM

```
1: while LQ.LEN() + CQ.LEN() >0 do
2:     if LQ.TIME() + D ≥ CQ.TIME() then
3:         LQ.DEQUEUE(pkt)                     ▷ L4S
4:         if (LQ.TIME() > T) ∨ (p_L > RAND()) then
5:             MARK(pkt)
6:     else
7:         CQ.DEQUEUE(pkt)                     ▷ Classic
8:         if p_C > RAND() then
9:             if ECN(pkt)==0 then             ▷ Not ECT
10:                DROP(pkt)                   ▷ Squared drop
11:                continue                    ▷ Redo loop
12:            else                            ▷ ECT(0)
13:                MARK(pkt)                   ▷ Squared mark
14:    RETURN(pkt)                             ▷ return the packet, stop here
```

Algorithms 1 & 2 summarize the per packet enqueue

and dequeue implementations of DualPI2 as pseudocode For clarity, overload and saturation logic are omitted. The full code is available as the Dualq option to the PI2 Linux qdisc implementation.[1] On enqueue, packets are time-stamped and classified. On dequeue, line 2 implements the time-shifted FIFO scheduler. It takes the packet that waited the longest, after adding time-shift D to the L4S queuing time. If an L4S packet is scheduled, line 4 marks the packet either if the L4S threshold is exceeded, or if a random marking decision is drawn according to the probability p_L. If a Classic packet is scheduled, line 8 decides whether to emit a congestion signal with probability p_C. Then line 9 checks whether the Classic packet is not ECN-capable, in which case it uses drop as the signal, otherwise it uses ECN.

The internal base signalling probability (p') is kept up to date by the core PI Algorithm (3) which only needs occasional execution [25]. The proportional gain factor β is multiplied by the change in queuing time. The integral gain factor α is typically smaller, to restore any persistent standing queue to the target delay. These expressions, which can be negative, are added to the previous p every T_{update} (default 16 ms). Then the L4S and Classic signalling probabilities, p_L and p_C are derived from p.

Algorithm 3 PI core: Every T_{update} p is updated

1: $curq = \text{CQ.TIME}()$
2: $p' = p' + \alpha * (curq - TARGET) + \beta * (curq - prevq)$
3: $p_L = k * p'$
4: $p_C = (p')^2$
5: $prevq = curq$

4. EVALUATION

4.1 Testbed Setup

We used a testbed to evaluate the proposed DualQ AQM mechanism in a realistic setting, and to run repeatable experiments in a controlled environment. The testbed was assembled using carrier grade equipment in the same enviroment as for testing customer solutions. Figure 3 depicts the testbed, which consists of a classical residential service delivery network composed of Residential Gateway, xDSL DSLAM (DSL Access Multiplexer), BNG (Broadband Network Gateway), Service Routers (SR) and application servers. The Residential Gateway is connected by VDSL to a DSLAM, which is connected to the BNG through an aggregation network, representing a local ISP or access wholesaler. Traffic is routed to another network representing a global ISP that hosts the application servers and offers breakout to the Internet. The client computers in the home network and the application servers at the global ISP are Linux machines, which can be configured to use any TCP variant, start applications and test traffic. The two client-server pairs (A and B) are respectively configured with the same TCP variants and applications.

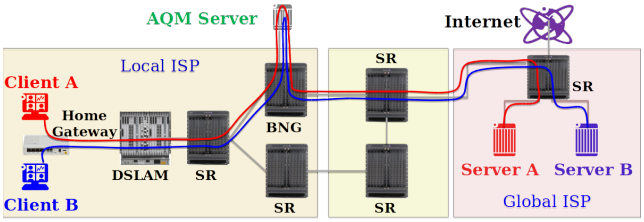

Figure 3: Testbed configuration

In a production access network, per-customer queues form the leaves of a hierarchical scheduling tree and they are deliberately arranged as the downstream bottleneck for each customer. Traffic from the client-server pairs is routed from the BNG through a Linux box ('AQM server'), which acts as the rate bottleneck where we configure the different AQMs being evaluated for the BNG. This server also emulates extra delay, controls the experiments, captures the traffic and analyses it. In practice it would also be important to deploy an AQM in the home gateway, but in our experiments the ACK traffic was below the upstream capacity.

The two client computers were connected to a modem using 100 Mbps Fast Ethernet; the xDSL line was configured at 48 Mbps downstream and 12 Mbps upstream; the links between network elements consisted of at least 1GigE connections. The base RTT (T_0) between the clients and servers was 7 ms, which was primarily due to the interleaved Forward Error Correction (FEC) configured for xDSL. We configured the different bottlenecks on the AQM server at the BNG on the downstream interface where the AQM was configured. Extra delay was configured on the upstream interface using a netem qdisc, to compose the total base RTTs tested.

To support higher bottleneck rates and lower RTTs all experiments were performed with the clients connected directly to the BNG with 1GigE connections. Those experiments fitting within xDSL limits were validated on the full testbed and compared, showing near identical results. All Linux computers were Ubuntu 14.04 LTS with kernel 3.18.9, which contained the implementations of the TCP variants and AQMs.

We used DCTCP for the Scalable congestion control and both Reno and Cubic for Classic, all with their default configurations[2]. In this paper we do not show Reno because the Cubic results were generally similar but not always as good. For ECN-Cubic, we enable TCP ECN negotiation. We compared DualPI2 with PIE and FQ-CoDel, all configured as in Table 1.

4.2 Experimental Approach

For traffic load we used long-running flows (§§ 4.3 & 4.4) and/or dynamic short flows (§ 4.5).

[1] Open source at https://github.com/olgabo/dualpi2

[2] Except DCTCP is patched to fix a bug that prevented it falling back to Reno on detecting a drop.

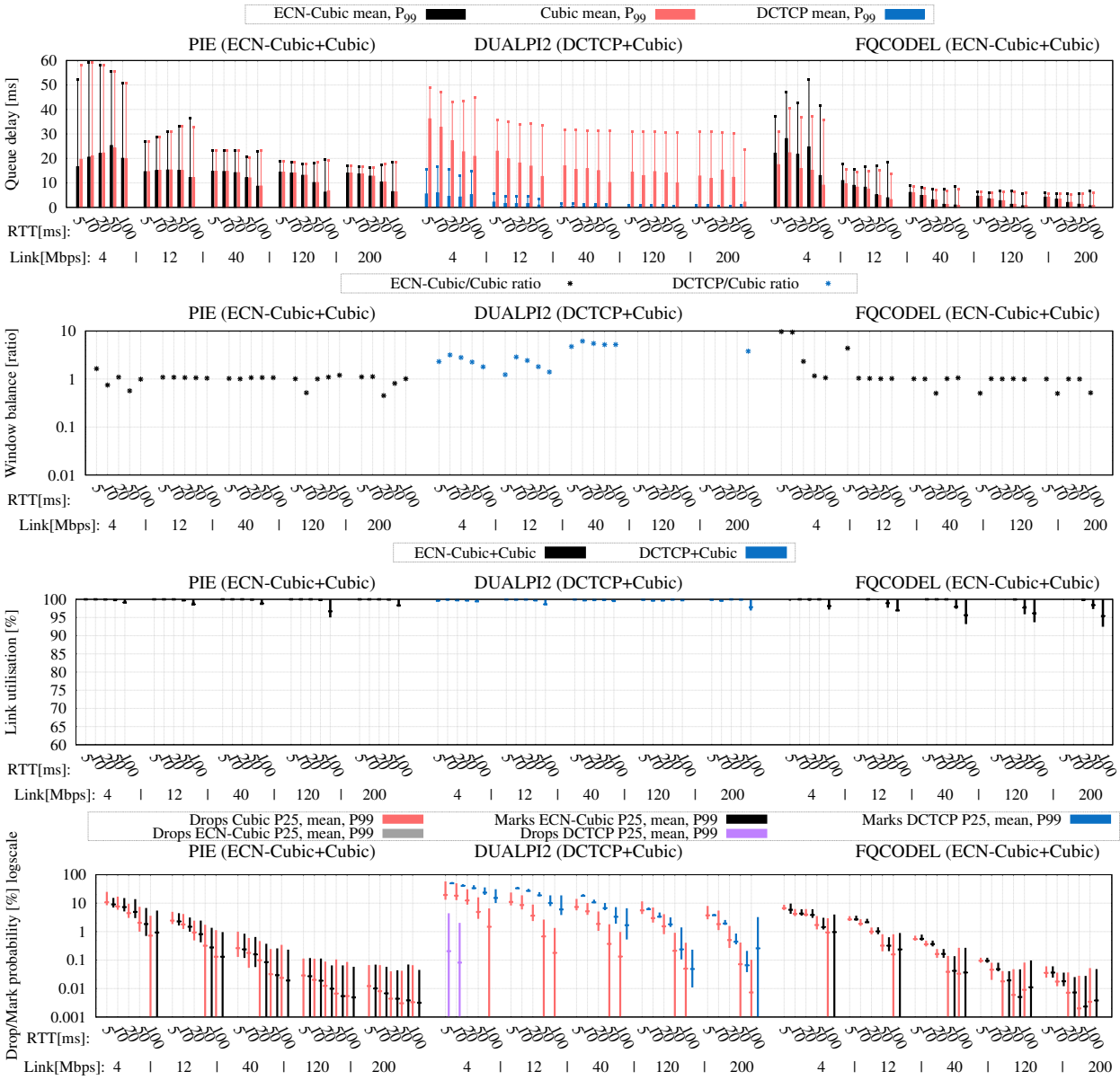

Figure 4: Equal RTT with 1 flow for each CC

All	Buffer: 40000 pkt, ECN enabled
PIE	Target delay: 15 ms, Burst: 100 ms, TUpdate: 16 ms, α: 1/16, β: 10/16, ECN_drop: 25%
FQ-CoDel	Target delay: 5 ms, Burst: 100 ms
DualPI2	Target delay: 15 ms, L4S T: 1 ms, D: 30 ms, α: 5/16, β: 50/16, k: 2, ECN_drop: 100% L4S

Table 1: Default parameters for the different AQMs.

We used long flows, not as an example of a realistic Internet traffic mix, rather to aid interpretation of various effects, such as starvation.

From all our experiments, we selected a representative subset to evaluate our two main performance goals: queuing delay and window balance. We also show rate balance, link utilization and drop/mark probability, as well as flow completion times in short flow experiments. Heavy load scenarios predominate in our selection, again not because they are typical, but because they do occur and they are the worst case.

We mixed different number of flows, evaluated flows with different congestion controls (CCs) and RTTs, and to verify behaviour on overload (§ 4.6), we injected unresponsive UDP load, both ECN and Not-ECN capable.

We configured PIE and FQ-CoDel with ECN as well as without, as a control so as not to attribute any performance gains to L4S ECN that are already available from Classic ECN. In this paper we present those combinations of CC and AQM that each AQM is intended to support: DCTCP with Cubic on DualPI2; and ECN-Cubic with Cubic on PIE and FQ-CoDel.

4.3 Experiments with long-running flows

Each experiment (lasting 250 s) was performed with a specified TCP variant configured on each client-server pair A and B and a specified AQM, bottleneck link speed and RTT on the AQM server. We performed a large number of experiments with different combinations of long-running flows, where each client started 0 to 10 file downloads on its matching server, resulting in 120 flow combinations competing at a common bottleneck for 250 seconds. These 120 experiments were executed for the 25 combinations of 5 RTTs (5, 10, 20, 50 and 100 ms) and 5 link speeds (4, 12, 40, 120 and 200 Mbps).

For the 1-1 (one flow on pair A and one on B) combination Figure 4 shows queue delay, window ratio, link utilization and mark/drop probability for each AQM and congestion control. The results are plotted for different combinations of link speeds and RTTs on the x-axis.

Looking at queuing delay we can clearly see that L4S delay and delay variance are significantly lower than the other AQMs. All AQMs roughly hold to their target(s), except with higher delays for lower rates and some expected under-utilization for higher base RTTs. The lower link rates drive the non-ECN AQMs up to drop probabilities around 10%.

For the medium and high throughputs, L4S achieves sub-millisecond average delays with 99$^{\text{th}}$ percentile around 2–3 ms. The higher queuing delays for the smaller throughputs are due to the single packet serialization time of 3 ms (1 ms) on a 4 Mbps (12 Mbps) link. This is why we set a floor of 2 packets for the L4S marking threshold otherwise it would always mark 100%. The Cubic flows on the DualPI2 AQM achieve a similar average queuing delay as with the PIE AQM. Due to the time-shifted overload mechanism the 99$^{\text{th}}$ percentile of the Cubic flows pushes up the average and 99$^{\text{th}}$ percentile of the L4S queue delay.

The drop/mark plot clearly demonstrates the difference between drop and mark for the DualPI2 AQM. The squared drop probability results in near-equal windows for the different CCs, as demonstrated in the window balance plot. Due to the small queue delay of the L4S traffic, the total amount of packets in flight is smaller than with the other AQMs. To compensate, a higher drop and mark probability is needed. For the 4 Mbps and 5 ms base RTT, the probabilities sporadically start to exceed the coupled 25% drop and 100% mark thresholds, with some L4S drop as a result. For the higher BDPs, the links are less utilized due to the large window reduction of Cubic, resulting in more on/off-type marking for DCTCP. Even when DCTCP is not able to fill this gap due to its additive increase, it still reduces less than Cubic, with a higher DCTCP window as a result. For the very high BDPs Cubic starts to switch out of its Reno mode, resulting in the higher window of pure Cubic mode.

Figure 5 shows the window ratio for different combinations of numbers of long-running flow. The figure shows the results for a 40 Mbps link and 10 ms RTT, which was representative of the other link rates and RTTs. The number of flows for each pair (A and B) is shown on the x-axis: the first value is the total number of ECN-capable flows (ECN-Cubic or DCTCP), while the second is the number of Cubic flows.

The results show that in general window sizes are well-balanced with all combinations. This confirms that the simple squared coupling of the DUALPI2 AQM counterbalances the more aggressive response of DCTCP remarkably precisely over the whole range of combinations of flows.

Only when there are very few Classic flows compared to L4S flows does the DCTCP window become smaller. This is due to the low and bursty queue occupancy of Classic flows,which causes DCTCP flows to frequently hit the L4S threshold. This results in additional marking and a smaller window for DCTCP. A higher L4S-threshold removes this effect. As the higher throughput for one Classic flow is spread over multiple L4S flows, the throughput of the L4S flows is not heavily impacted, suggesting that if a compromise needs to be struck between low L4S latency and window balance, a low L4S threshold will always be preferable.

Throughput variance experiments with more than 2 flows (not shown due to space limitations) illustrate that, when a Classic flow competes with an L4S, it conveys its variations to the L4S flow (which fills up the gaps). However, when solely DCTCP flows compete their rates are much more stable.

4.4 Experiments with different RTTs

To evaluate the RTT-dependence of the windows and rates of different CCs, we conducted additional experiments with one flow per client server pair, each having different base RTTs. These experiments were repeated for the 5 link speeds.

Figure 6 shows queue delay and window and rate ratio for flows with unequal RTTs, running concurrently. We use one flow for each congestion control, labelled as flow A for ECN congestion controls (ECN-Cubic or DCTCP) and flow B for Cubic. Different combinations of RTTs for each of the flows are shown on the x-axis. For example, 5-20 means 5 ms base RTT for flow A and 20 ms for flow B.

Looking first at queuing delays in the DualPI2 AQM, it can be seen that the extremely low latency for L4S traffic is preserved in all cases, including in the presence of longer RTT traffic. Large-RTT Classic flows combined with small-RTT L4S flows result in a longer average Classic queue (see A-B = 5-100). This is again due to the bursty character of ACK-clocked Classic TCP flows, which need to wait until the L4S traffic has backed off sufficiently to create scheduling opportunities for the Classic flows. This effect is tempered by the time-shifted scheduler, which limits the waiting

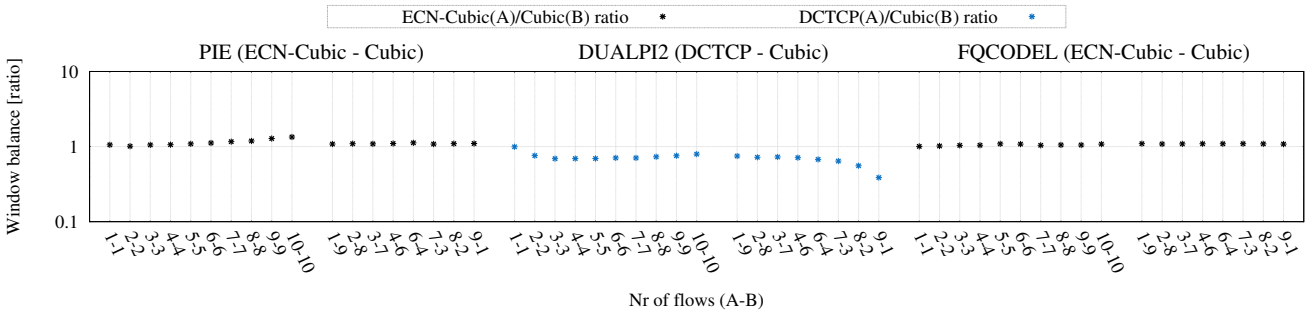

Figure 5: Different number of flows on a 40 Mbps link with 10 ms RTT.

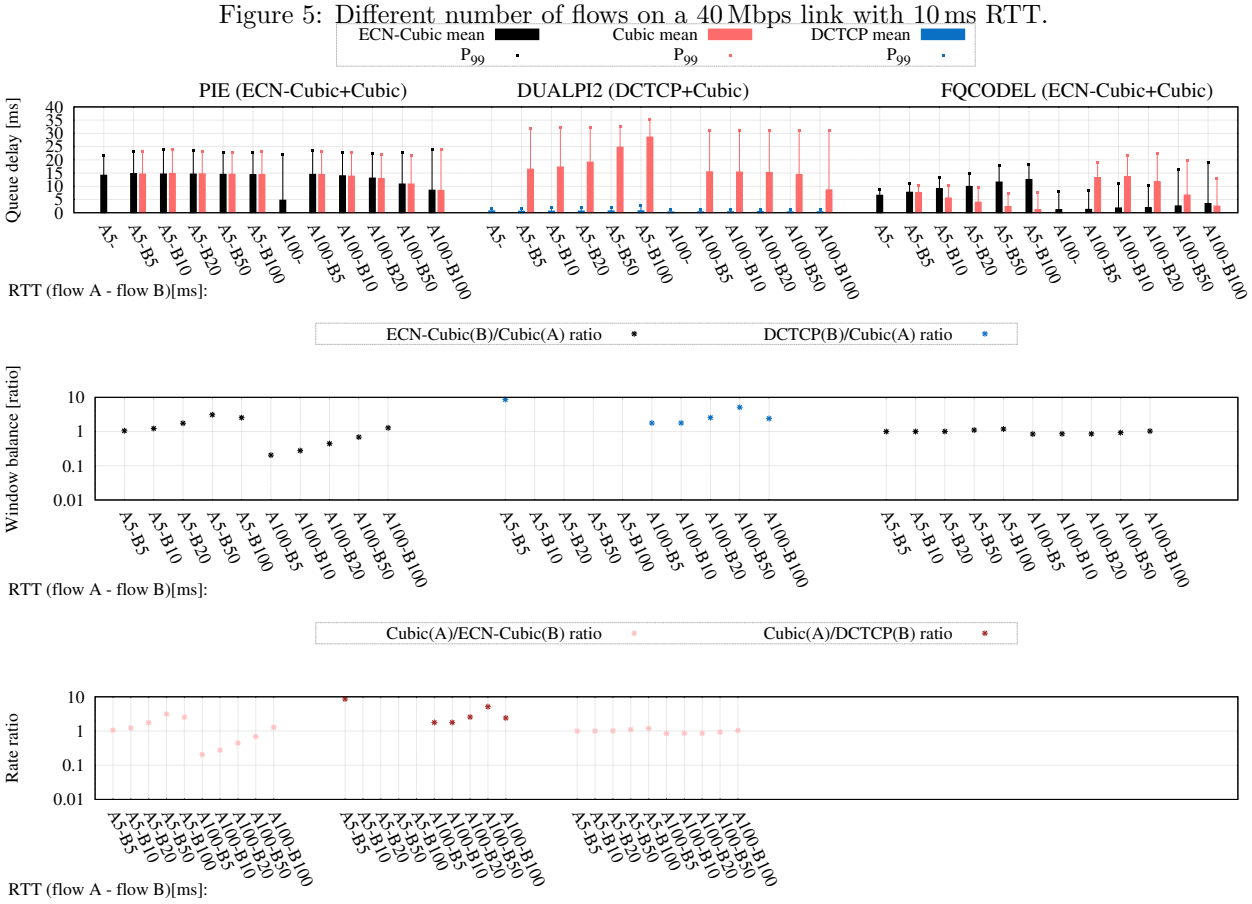

Figure 6: Mixed RTT with 1 flow for each CC on a 40 Mbps link.

time for the burst to 30 ms at the expense of higher 99$^{\text{th}}$ percentile delay for the L4S traffic.

In this same 5-100 case, window balance also suffers. The bursty Classic traffic with its associated higher L4S threshold marking drags down the L4S window size.

Comparing the bottom two plots, particularly in the 5-100 case, with PIE or DUALPI2 it can be seen that window balance leads to considerable rate imbalance. This is not surprising, because it is well known that competing TCP flows equalize their congestion windows so their bit rates will be inversely proportional to their RTTs. However, as AQM reduce queueing delay they intensify this effect, because the ratio between total RTTs tends towards the ratio between base RTTs. The implications of this trend are discussed in § 5.2.

For instance, in the 5-100 case when the ratio between base RTTs is 20×, the ratio between flow rates is about 6×. This is because PIE holds queuing delay at about 15 ms, and $(100 + 15)/(5 + 15) \approx 6$.

L4S all-but eliminates queuing delay so total RTT is hardly any greater than base RTT. Therefore even for the 5-50 case, rate imbalance is already approaching 10×. In the 5-100 case, it can be seen that the rate-imbalance trend reverses. However, this is due to the increased variance of the L4S queue in response to increased Cubic burstiness as discussed above. In other experiments (not shown) with the burstiness of Cubic removed by using 2 DCTCP flows alone, rate imbal-

ance does indeed tend towards the inverse of the ratio between the base RTTs of the flows.

Conversely, with FQ-CoDel the Flow Queuing scheduler enforces rate balance, which necessarily requires considerable window imbalance.

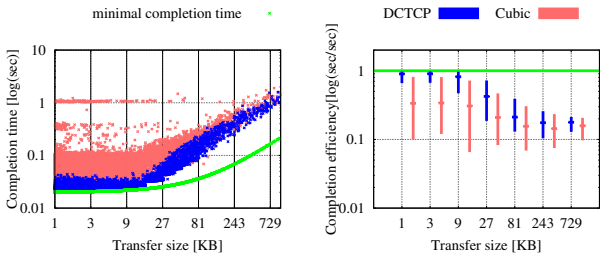

Figure 7: Completion time against efficiency representation for 1 long flow and high dynamic load each on a 40 Mbps link with 10 ms base RTT.

4.5 Experiments with dynamic short flows

On top of the long flow experiments, we added emulated web traffic load patterns between each client-server pair, to evaluate the dynamic behaviour of the AQMs with their congestion controllers. For this we used an exponential arrival process with an average of 1 (low load) or 10 (high load) requested items per second for the 4 Mbps link capacity, scaled for the higher link speeds up to 50 (low) or 500 (high) requests for the 200 Mbps links. Every request opened a new TCP connection, closed by the server after sending data with a size according to a Pareto distribution with $\alpha = 0.9$ and a minimum size of 1 KB and maximum 1 MB. The client logged the completion time and downloaded size. Timing was started just before opening the TCP socket, and stopped after the close by the server was detected.

The left-hand side of Figure 7 shows a log-log scatter plot of the completion time to item size relation for the high load DualPI2 AQM test case on a 40 Mbps link with 10 ms base RTT. The green line is the theoretically achievable completion time, taking the RTT into account but downloading at full link speed from the start. As can be seen, the L4S short flows (within the initial window size of 10 packets) closely achieve this. They leave the TCP stack in a burst and face very low delay in the network. This same representation also helps in understanding where Classic download time is typically lost. Around 1 second a lot of downloads had to wait for the retransmission time-out after lost SYN packets. Around 200 ms the minimum retransmission time-out for tail loss is clearly visible. Long flows share the throughput better, which is why they are further from the theoretical completion time for a lone flow.

To better quantify the average and percentiles of the completion times, we used the Completion Efficiency representation on the right of Figure 7. To calculate its completion efficiency for each item we divided actual by theoretical completion time. We then binned the samples in log scale bins (base 3) and calculated the average, 1st and 99th percentiles. The green theoretical completion time is now at 1 (maximum efficiency).

Figure 8 shows completion efficiency for a high load of short flows plus a single long-running flow for each congestion control on a 120 Mbps link with different RTTs.

With DualQ or FQ the completion times of short flows are near-ideal. DualQ achieves this by keeping the queue very shallow for all L4S flows. In contrast FQ explicitly identifies and priority-schedules short flows.

In higher BDP cases, and in the high load case shown, the completion times of larger downloads are longer with DualPI2 than with the other AQMs. This is partly due to the additional marking of bursty traffic due to the shallow L4S threshold, which gives Cubic flows an advantage (as already discussed). However, the primary cause is a known problem with DCTCP convergence time. When a DCTCP flow is trying to push in against a high load of other DCTCP flows, it drops out of slow start very early, because of the higher prevailing marking level. Then it falls back to pushing in very slowly using only additive increase. Similarly, when another flow departs, the additive increase of DCTCP takes many round trips to fill the newly available capacity.

Others have noticed this problem and modified the additive increase of DCTCP [44]. Nonetheless, DCTCP slow start also has to be modified—the aggression of slow start in one flow has to increase to match the increased aggression of congestion avoidance in others. Solving this problem is included in the TCP Prague requirements (see § 5.2), but it is outside the AQM-only scope of the present paper.

Figure 9 adds further weight to the argument that DCTCP, not the DualQ AQM, is the cause of the longer completion times. Average queue delay, queue variance and link utilization are all better with L4S/DualQ than with FQ-CoDel. So it seems that DCTCP is just not exploiting these advantages.

If we now compare the results in Figure 9 with those for just long-running flows in Figure 4, we see the effect of adding dynamic flows. They dramatically increase queue delay variance (note the change in scale), particularly with FQCODEL and PIE. Nonetheless, L4S queuing delay is still extremely low, with only a slight increase in variance.

Comparing the link utilization plots, the added dynamic flows universally reduce utilization as arriving flows take a while to use up the capacity that departing flows vacate, particularly at higher RTTs. With DUALPI2, under-utilization is only a little worse than with PIE, despite DCTCP's convergence problem (discussed above). This is because the Classic Cubic traffic takes up some of the slack.

4.6 Overload experiments

To validate the correct overload behaviour, we added an unresponsive UDP flow to 5 long-running flows of each congestion control type (ECN and non-ECN) over

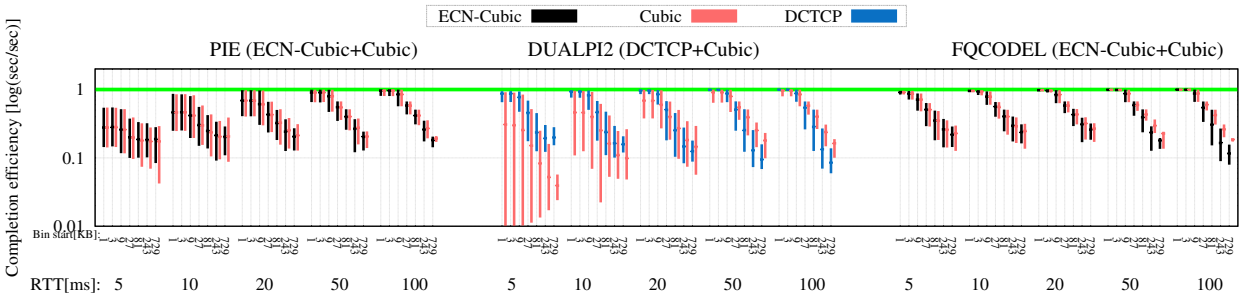

Figure 8: Heavy dynamic workload: 1 long flow and 300 short requests per second for each CC on a 120 Mbps link with equal 10 ms base RTT. The bin boundaries are 1 KB, 3 KB, 9 KB, 27 KB, 81 KB, 243 KB, 729 KB and 1 MB.

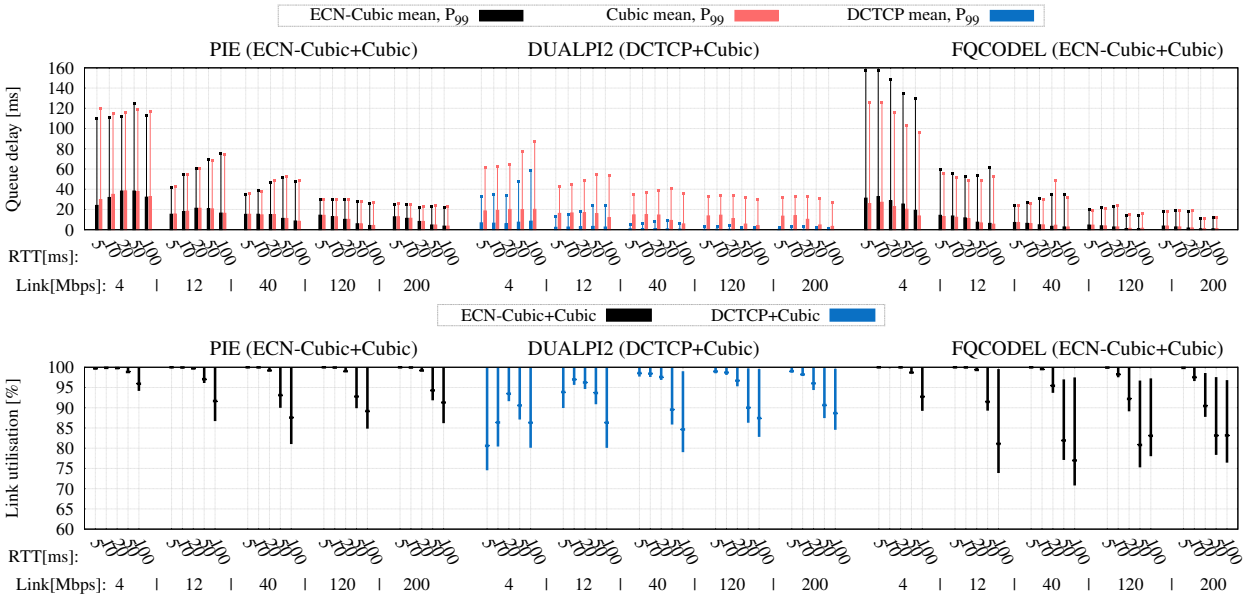

Figure 9: Heavy dynamic workload: 1 long flow and 300 short requests per second for each CC.

a 100 Mbps bottleneck link with 10 ms base RTT. For each AQM we ran 2 sets of tests with the UDP traffic marked as either ECN/L4S or non-ECN. Each set tested 5 different UDP rates (50, 70, 100, 140 and 200 Mbps).

Figure 10 shows the results for the DualQ AQM. The top plot shows the link output rate for each traffic type. The more the UDP flow squeezes the responsive flows, the more they drive up the congestion level (ECN or drop). Only responsive flows heed ECN marks. So, in the ECN UDP flow case, before congestion reaches the level where the AQM starts dropping ECN packets, the UDP flow is unaffected by congestion.

Once the AQM starts dropping ECN packets (and in the non-ECN UDP flow case), the drop probability necessary to make the responsive flows fit into the remaining capacity also subtracts from the UDP flow, freeing up some extra capacity for the responsive traffic.

The capacity left by the UDP flow for responsive traffic is roughly the same whether the UDP flow uses L4S-ECN or not, but the largest difference is where the arrival rate of the UDP flow is around 100% of the capacity. Once unresponsive traffic significantly exceeds 100%, it leaves very little capacity for the responsive traffic.

All this behaviour was exactly the same as with a single queue AQM (i.e. PIE), which was our intention. We wanted to ensure that introducing two queues would not introduce any new pathologies. Then any applications relying on unresponsive behaviour should work the same, and any optional mechanisms to police unresponsive flows should also work the same.

In contrast, flow queuing starts dropping unresponsive traffic when it exceeds an equal share of throughput. For instance, a 50 Mbps flow experiences about 80% drop, to force it to share the capacity equally with 10 other flows.

The middle plot shows that the windows of the DCTCP and Cubic flows balance as long as the unresponsive traffic is no greater than the link capacity. For higher levels of unresponsive traffic, the throughput of the responsive traffic is more dominated by long retransmission time-outs, which results in more equal

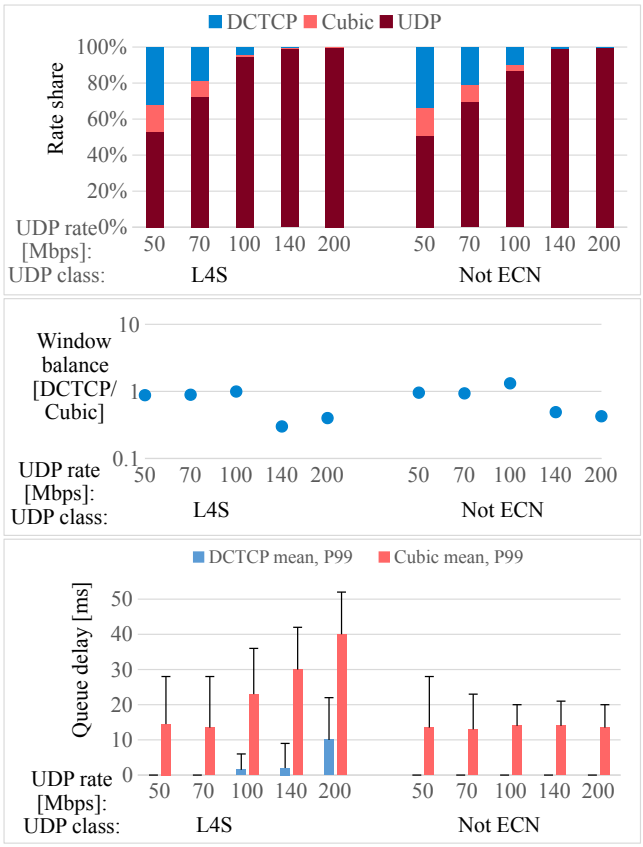

Figure 10: Overload experiments on a 100 Mbps link

rates, causing window imbalance because of the different RTTs.

Finally the bottom plot shows the queuing delay for the DualQ during the same experiments. The most notable feature is that, whether the unresponsive traffic is L4S or Not ECN, average L4S queuing delay remains below about 2 ms, except in the L4S UDP case, and then only once it exceeds 140% of capacity.

In the case when the UDP traffic is not ECN, the PI2 AQM holds Classic queue delay to its target by applying sufficient drop. The coupled AQM translates this to a high level of L4S marking or, if congestion is high, it applies the same level of drop to both queues. Given L4S throughput is relatively low in this case, it is easy for L4S queuing delay to remain very low.

In the case when the UDP traffic is L4S, the majority of the load arrives at the L4S queue. The native L4S threshold only applies marking, which the UDP traffic ignores. So the overload mechanism described in § 3.4 starts to dominate. This takes over whenever the Classic queue is empty, which happens increasingly often as more UDP L4S traffic arrives. At such times, the base AQM (PI controller) uses the L4S queue delay to drive its output, still aiming for the Classic 15 ms target. The more unresponsive traffic that arrives at the L4S queue, the more the L4S queue shifts from the 1 ms L4S threshold to the 15 ms Classic target. This effect can be seen between 100% and 200% in the L4S UDP case.

5. DEPLOYMENT CONSIDERATIONS

5.1 Standardization Requirements

The IETF has taken on L4S standardization work, in principle. It has adopted a proposal [6] to make the ECT(1) codepoint available for experimental classification of L4S packets at the IP layer (v4 and v6), as described in § 3. [18] considers the pros and cons of various candidate identifiers and finds that none are without problems, but proposes ECT(1) as the least worst.

The main issue is that there is only one spare codepoint, so a queue can distinguish L and C packets, but congestion marking has to use the same Congestion Experienced (CE) codepoint for both L & C packets. This is not a problem for hosts but, in the (unusual) case of multiple simultaneous bottlenecks, any packet already marked CE upstream will have to be classified into the L queue, irrespective of whether it was originally C or L. This is considered acceptable for TCP given that, if a few packets arrive early out of order, subsequent packets still advance the ACK counter.

Operators will be able to classify L4S on additional identifiers (e.g. by ECN plus address range or VLAN ID), which they might use for initial exclusivity, without compromising long-term interoperability.

The IETF also plans to define the semantics of the new identifier. The 'Classic' ECN standard [39] defines a CE mark as equivalent to a drop, so queuing delay with Classic ECN cannot be better than with drop (this may be why operators have not deployed Classic ECN [41, § 5]). The square relationship between an L4S mark and a drop in this paper (Eqn. (2)) has been proposed for experimental standardization [18]. Nonetheless, it has been proposed to recommend rather than standardize a value for the coupling factor, k, given differences would not prevent interoperability.

The IETF is also adopting a specification of the dualQ coupled AQM mechanism [17] so that multiple implementations can be built, tested and compared, possibly using different base AQMs internally.

5.2 Congestion Control Roadmap

This paper uses DCTCP unmodified[3] in all experiments i) to focus the parameter space of our experiments on the network mechanism, without which end-system performance improvements would be moot; and ii) to emphasize that the end-system side of the multi-party deployment is already available (in the Linux mainline and Windows), at least for testing purposes. Nonetheless, numerous improvements to DCTCP can be envisaged for this new public Internet scenario. They are listed below in priority order starting with those necessary for safety, and ending with performance improvements. They are adapted from the congestion

[3]See footnote 2.

control requirements identified in the IETF L4S architecture draft [12], which are in turn adapted from the "TCP Prague requirements", named after the meeting in Prague of a large group of DCTCP developers that informally agreed them [8]:

1. Fall back to Reno/Cubic on loss (Windows does, but Linux does not due to a bug—fix submitted);
2. Negotiate altered feedback semantics [30, 11];
3. Use of a standardized packet identifier [18];
4. Handle a window of less than 2, rather than grow the queue if base RTT is low [10];
5. Smooth ECN feedback over a flow's own RTT, not the RTT hard-coded for data-centres [2, § 5];
6. Fall back to Reno/Cubic if increased delay of classic ECN bottleneck detected;
7. Faster-than-additive increase, e.g. Adaptive Acceleration (A^{2}DTCP) [44];
8. Less drastic exit from slow-start, similar goal to Flow-Aware (FA-DCTCP) [27];
9. Reduce RTT-dependence of rate [2, § 5] (see below).

With tail-drop queues, so-called 'RTT-unfairness' had never been a great cause for concern because the RTTs of all long-running flows included a common queuing delay component that was no less than worst-case base RTT (due to the historical rule of thumb for sizing access link buffers[4] at 1 worst-case RTT). So, even where the ratio between base delays was extreme, the ratio between total RTTs rarely exceeded 2 (e.g. if worst-case base RTT is 100 ms, worst-case total RTT imbalance tends to $(100 + 100)/(0 + 100)$.

However, Classic AQMs reduce queuing delay to a typical, rather than worst-case, RTT. For instance, with PIE, the queuing delay common to each flow is 15 ms. Therefore, worst-case rate imbalance will be $(100 + 15)/(0 + 15) \approx 8$ (see the explanation in § 4.4 of the rate imbalance in Figure 6).

Because of the cushioning effect of queuing delay, even when base RTTs are extremely imbalanced rates are not. But, because L4S all-but eliminates queuing delay, it exposes the full effect of the 'RTT-unfairness' issue.

We do not believe the network needs to be involved in addressing this problem. RTT-dependence is a feature of end-to-end congestion controls, so that is where it should be addressed. Classic CCs will not need to change, because classic queues will still need to be large to avoid under-utilization. However, L4S congestion controls will need to be less RTT-dependent, to avoid starving any L4S and Classic flows with larger RTTs (hence reduced RTT-dependence has been added to the TCP-Prague requirements above).

As a fortunate side-effect, it will be easier to define the coupling factor k (see § 3.2) to balance throughput between RTT-independent L4S traffic and large-queued Classic traffic.

[4]Note that access buffers cannot exploit such high flow aggregation as in the core [20]

5.3 Deployment Scenarios

The applicability of the DualQ is of course not limited to fixed public access networks. The DualQ Coupled AQM should also enable DCTCP to be deployed across multi-tenant data centres or across community of interest networks connecting private data centres—anywhere where the lack of a centralized system-admin makes coordinated deployment of DCTCP impractical. The most likely DC bottlenecks could be prioritized for deployment, e.g. at the ingress and egress of hypervisors or top-of-rack switches depending on topology, and at WAN access points.

In mobile networks the bottleneck is usually the radio access where buffering is more complex, but in principle an AQM similar to the Coupled DualQ ought to work.

6. RELATED WORK

In 2002, Gibbens and Kelly [21] developed a scheme to mark ECN in a priority queue based on the combined length of both queues. However, they were not trying to serve different congestion controllers as in the present work. In 2005 Kuzmanovic [32, §5] presaged the main elements of DCTCP showing that ECN should enable a naïve unsmoothed threshold marking scheme to outperform sophisticated AQMs like the proportional integral (PI) controller. It assumed smoothing at the sender, as earlier proposed by Floyd [19].

Wu et al. [42] investigates a way to incrementally deploy DCTCP within data centres, marking ECN when the temporal queue exceeds a shallow threshold but using standard ECN [39] on end-systems. Kuhlewind et al. [31] showed that DCTCP and Reno could co-exist in the same queue configured with a form of WRED [14] classifying on ECN not Diffserv. Judd [28] uses Diffserv scheduling to partition data centre switches between DCTCP and classic traffic in a financial data centre scenario, but as already explained this relies on management configuration based on prediction of the traffic matrix and its dynamics, which becomes hard on low stat-mux links. Fair Low Latency (FaLL) [43] is an AQM for DC switches building on CoDel [37]. Unlike the DualQ, FaLL inspects the transport layer of sample packets to focus more marking onto faster flows while keeping the queue short.

7. CONCLUSION

Classic TCP induces two impairments: queuing delay and loss. A good AQM can reduce queuing delay but then TCP induces higher loss. In a low stat-mux link, there is a limit to how much an AQM can reduce queuing delay without TCP's sawteeth introducing a third impairment: under-utilization. Thus TCP is like a balloon: when the network squeezes one impairment, another bulges out.

This paper moves on from debating where the net-

work should best squeeze the TCP balloon. It recognizes that the problem is now wholly outside the network: Classic TCP (the balloon itself) is the problem. But this does not mean the solution is also wholly outside the network. This paper has shown that the network plays a crucial role in enabling hosts to transition away from the Classic TCP balloon. The 'DualQ Coupled AQM' detailed in this paper is not notable as somehow a 'better' AQM than others. Rather, it is notable as a coupling between two AQMs in two queues—as a transition mechanism to enable hosts to kick out their old TCP balloon.

Hosts will then be able to transition to a member of the family of scalable congestion controls. This can still be likened to a balloon. But it is a tiny balloon (near-zero impairments) and, importantly, it will stay the same tiny size (invariant impairments as BDP scales). Whereas the Classic TCP balloon is continuing to grow (worsening impairments) as BDP scales. This is why we call the new Internet service 'Low Latency Low Loss Scalable throughput' (L4S).

The paper provides not just the mechanism but also the incentive for transition—the tiny size of all the impairments. For link rates from 4–200 Mb/s and RTTs from 5–100 ms, our extensive testbed experiments with a wide range of heavy load scenarios have shown near-zero congestion loss; sub-millisecond average queuing delay (roughly 500 μs) with tight variance; and near-full utilization.

We have been careful as far as possible to do no harm to those still using the Classic service. Also, given the network splits traffic into two queues, when it merges them back together, we have taken great care that it does not enforce flow 'fairness'. Nonetheless, if hosts are aiming for flow 'fairness' they will get it, while remaining oblivious to the difference between Scalable and Classic congestion controls.

We have been careful to handle overload in the same principled way as normal operation, preserving the same ultra-low delay for L4S packets, and dropping excess load as if the two queues were one.

And finally, we have been careful to heed the zero-config requirement of recent AQM research, not only ensuring the AQMs inherently auto-tune to link rate, but also shifting RTT-dependent smoothing to end-systems, which know their own RTT.

8. REFERENCES

[1] Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data Center TCP (DCTCP). *Proc. ACM SIGCOMM'10, Computer Communication Review 40*, 4 (Oct. 2010), 63–74.

[2] Alizadeh, M., Javanmard, A., and Prabhakar, B. Analysis of DCTCP: Stability, Convergence, and Fairness. *Proc. ACM SIGMETRICS'11* (2011).

[3] Bai, W., Chen, K., Chen, L., Kim, C., and Wu, H. Enabling ECN over Generic Packet Scheduling. In *Proc. Int'l Conf Emerging Networking EXperiments and Technologies* (New York, NY, USA, 2016), CoNEXT '16, ACM, pp. 191–204.

[4] Bansal, D., and Balakrishnan, H. Binomial Congestion Control Algorithms. In *Proc. IEEE Conference on Computer Communications (Infocom'01)* (Apr. 2001), IEEE, pp. 631–640.

[5] Belshe, M., Peon, R., and Thomson (Ed.), M. Hypertext Transfer Protocol version 2 (HTTP/2). Request for Comments 7540, RFC Editor, May 2015.

[6] Black, D. Explicit Congestion Notification (ECN) Experimentation. Internet Draft draft-ietf-tsvwg-ecn-experimentation-00, Internet Engineering Task Force, Dec. 2016. (Work in Progress).

[7] Bondarenko, O., De Schepper, K., Tsang, I.-J., Briscoe, B., Petlund, A., and Griwodz, C. Ultra-Low Delay for All: Live Experience, Live Analysis. In *Proc. ACM Multimedia Systems; Demo Session* (New York, NY, USA, May 2016), ACM, pp. 33:1–33:4.

[8] Briscoe, B. [tcpPrague] Notes: DCTCP evolution 'bar BoF': Tue 21 Jul 2015, 17:40, Prague. Archived mailing list posting URL: https://mailarchive.ietf.org/arch/msg/tcpprague/mwWncQg3egPd15FItYWiEvRDrvA, July 2015.

[9] Briscoe, B., Brunstrom, A., Petlund, A., Hayes, D., Ros, D., Tsang, I.-J., Gjessing, S., Fairhurst, G., Griwodz, C., and Welzl, M. Reducing Internet Latency: A Survey of Techniques and their Merits. *IEEE Communications Surveys & Tutorials 18*, 3 (Q3 2016), 2149–2196.

[10] Briscoe, B., and de Schepper, K. Scaling TCP's Congestion Window for Small Round Trip Times. Technical report TR-TUB8-2015-002, BT, May 2015. http://riteproject.eu/publications/.

[11] Briscoe, B., Kühlewind, M., and Scheffenegger, R. More Accurate ECN Feedback in TCP. Internet Draft draft-ietf-tcpm-accurate-ecn-02, Internet Engineering Task Force, Oct. 2016. (Work in Progress).

[12] Briscoe (Ed.), B., De Schepper, K., and Bagnulo, M. Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture. Internet Draft draft-briscoe-tsvwg-l4s-arch-00, Internet Engineering Task Force, Oct. 2016. (Work in Progress).

[13] Chen, W., Cheng, P., Ren, F., Shu, R., and Lin, C. Ease the Queue Oscillation: Analysis and Enhancement of DCTCP. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd*

International Conference on (July 2013), pp. 450–459.

[14] Clark, D. D., and Fang, W. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking 6*, 4 (Aug. 1998), 362–373.

[15] Davie, B., et al. An Expedited Forwarding PHB (Per-Hop Behavior). Request for Comments 3246, Internet Engineering Task Force, Mar. 2002.

[16] De Schepper, K., Bondarenko, O., Tsang, I.-J., and Briscoe, B. PI^2 : A Linearized AQM for both Classic and Scalable TCP. In *Proc. ACM CoNEXT 2016* (New York, NY, USA, Dec. 2016), ACM.

[17] De Schepper, K., Briscoe (Ed.), B., Bondarenko, O., and Tsang, I.-J. DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput. Internet Draft draft-briscoe-tsvwg-aqm-dualq-coupled-00, Internet Engineering Task Force, Oct. 2016. (Work in Progress).

[18] De Schepper, K., Briscoe (Ed.), B., and Tsang, I.-J. Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay. Internet Draft draft-briscoe-tsvwg-ecn-l4s-id-02, Internet Engineering Task Force, Oct. 2016. (Work in Progress).

[19] Floyd, S. TCP and Explicit Congestion Notification. *ACM SIGCOMM Computer Communication Review 24*, 5 (Oct. 1994), 10–23. (This issue of CCR incorrectly has '1995' on the cover).

[20] Ganjali, Y., and McKeown, N. Update on Buffer Sizing in Internet Routers. *ACM SIGCOMM Computer Communication Review 36* (Oct. 2006).

[21] Gibbens, R. J., and Kelly, F. P. On Packet Marking at Priority Queues. *IEEE Transactions on Automatic Control 47*, 6 (June 2002), 1016–1020.

[22] Ha, S., Rhee, I., and Xu, L. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Operating Systems Review 42*, 5 (July 2008), 64–74.

[23] Hoeiland-Joergensen, T., McKenney, P., Täht, D., Gettys, J., and Dumazet, E. The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm. Internet Draft draft-ietf-aqm-fq-codel-06, Internet Engineering Task Force, Mar. 2016. (work in progress).

[24] Hohlfeld, O., Pujol, E., Ciucu, F., Feldmann, A., and Barford, P. A QoE Perspective on Sizing Network Buffers. In *Proc. Internet Measurement Conf (IMC'14)* (Nov. 2014), ACM, pp. 333–346.

[25] Hollot, C. V., Misra, V., Towsley, D., and Gong, W. Analysis and design of controllers for AQM routers supporting TCP flows. *IEEE Transactions on Automatic Control 47*, 6 (Jun 2002), 945–959.

[26] Irteza, S., Ahmed, A., Farrukh, S., Memon, B., and Qazi, I. On the Coexistence of Transport Protocols in Data Centers. In *Proc. IEEE Int'l Conf. on Communications (ICC 2014)* (June 2014), pp. 3203–3208.

[27] Joy, S., and Nayak, A. Improving Flow Completion Time for Short Flows in Datacenter Networks. In *Int'l Symposium on Integrated Network Management (IM 2015)* (May 2015), IFIP/IEEE, pp. 700–705.

[28] Judd, G. Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, May 2015), USENIX Association, pp. 145–157.

[29] Kelly, T. Scalable tcp: Improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review 32*, 2 (Apr. 2003).

[30] Kühlewind, M., Scheffenegger, R., and Briscoe, B. Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback. Request for Comments 7560, RFC Editor, Aug. 2015.

[31] Kühlewind, M., Wagner, D. P., Espinosa, J. M. R., and Briscoe, B. Using Data Center TCP (DCTCP) in the Internet. In *Proc. Third IEEE Globecom Workshop on Telecommunications Standards: From Research to Standards* (Dec. 2014), pp. 583–588.

[32] Kuzmanovic, A. The Power of Explicit Congestion Notification. *Proc. ACM SIGCOMM'05, Computer Communication Review 35*, 4 (2005).

[33] Kwon, M., and Fahmy, S. A Comparison of Load-based and Queue-based Active Queue Management Algorithms. In *Proc. Int'l Soc. for Optical Engineering (SPIE)* (2002), vol. 4866, pp. 35–46.

[34] Mathis, M. Relentless Congestion Control. In *Proc. Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports (PFLDNeT'09)* (May 2009).

[35] Mathis, M., Semke, J., Mahdavi, J., and Ott, T. The macroscopic behavior of the TCP Congestion Avoidance algorithm. *Computer Communication Review 27*, 3 (July 1997).

[36] Menth, M., Schmid, M., Heiss, H., and Reim, T. MEDF - a simple scheduling algorithm for two real-time transport service classes with application in the UTRAN. In *Proc. IEEE*

Conference on Computer Communications (INFOCOM'03) (Mar. 2003), vol. 2, pp. 1116–1122.

[37] Nichols, K., and Jacobson, V. Controlling Queue Delay. *ACM Queue 10*, 5 (May 2012).

[38] Pan, R., Piglione, P. N. C., Prabhu, M., Subramanian, V., Baker, F., and Ver Steeg, B. PIE: A Lightweight Control Scheme To Address the Bufferbloat Problem. In *High Performance Switching and Routing (HPSR'13)* (2013), IEEE.

[39] Ramakrishnan, K. K., Floyd, S., and Black, D. The Addition of Explicit Congestion Notification (ECN) to IP. Request for Comments 3168, RFC Editor, Sept. 2001.

[40] Salim, J. H., and Ahmed, U. Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks. Request for Comments 2884, RFC Editor, July 2000.

[41] Welzl, M., and Fairhurst, G. The Benefits of using Explicit Congestion Notification (ECN). Internet Draft draft-ietf-aqm-ecn-benefits-08, Internet Engineering Task Force, Nov. 2015. (Work in Progress).

[42] Wu, H., Ju, J., Lu, G., Guo, C., Xiong, Y., and Zhang, Y. Tuning ECN for Data Center Networks. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2012), CoNEXT '12, ACM, pp. 25–36.

[43] Xue, L., Chiu, C.-H., Kumar, S., Kondikoppa, P., and Park, S.-J. FaLL: A fair and low latency queuing scheme for data center networks. In *Intl. Conf. on Computing, Networking and Communications (ICNC 2015)* (Feb. 2015), pp. 771–777.

[44] Zhang, T., Wang, J., Huang, J., Huang, Y., Chen, J., and Pan, Y. Adaptive-Acceleration Data Center TCP. *IEEE Transactions on Computers 64*, 6 (June 2015), 1522–1533.