Dual Queue Coupled AQM:
Deployable Very Low Queuing Delay for All

Koen De Schepper*¶ Olga Albisser†¶ Olivier Tilmans* and Bob Briscoe‡

Abstract—On the Internet, sub-millisecond queuing delay and capacity-seeking have traditionally been considered mutually exclusive. We introduce a service that offers both: Low Latency Low Loss Scalable throughput (L4S). When tested under a wide range of conditions emulated on a testbed using real residential broadband equipment, queue delay remained both low (median 100–300 μs) and consistent (99th percentile below 2 ms even under highly dynamic workloads), without compromising other metrics (zero congestion loss and close to full utilization). L4S exploits the properties of 'Scalable' congestion controls (e.g., DCTCP, TCP Prague). Flows using such congestion control are however very aggressive, which causes a deployment challenge as L4S has to coexist with so-called 'Classic' flows (e.g., Reno, CUBIC). This paper introduces an architectural solution: 'Dual Queue Coupled Active Queue Management', which enables balance between Scalable and Classic flows. It counterbalances the more aggressive response of Scalable flows with more aggressive marking, without having to inspect flow identifiers. The Dual Queue structure has been implemented as a Linux queuing discipline. It acts like a semi-permeable membrane, isolating the latency of Scalable and 'Classic' traffic, but coupling their capacity into a single bandwidth pool. This paper justifies the design and implementation choices, and visualizes a representative selection of hundreds of thousands of experiment runs to test our claims.

I. Introduction

A. Problem:

With increases in bandwidth, latency is becoming the critical performance factor [31], [53]. Latency is a multi-faceted problem that has to be tackled on many different fronts [14]. This paper tackles queuing delay, which is often one of the top contributions to overall latency, alongside propagation delay.

Although queuing delay is intermittent, as quantified by Hohlfeld *et al* [40], it is prevalent enough to dominate experience for two reasons: i) a few higher delay packets often hold back the whole flow of logic [18]; and ii) human perception is dominated by episodes of poorer quality [52].

Therefore our goal is very low queuing delay, not just on average but for a high percentile of packets. 'Very low' means within single digit milliseconds, for instance 2 ms at the 99th percentile over a typical Internet path, which is an order of magnitude lower than with PIE [51] or FQ-CoDel [39].

A major motivation for drastically cutting tail latency is to leave a larger delay budget for propagation, so that interaction can stretch to transcontinental distances. For example, a 'responsive feel' in remote presence or remote control needs

*Nokia Bell Labs, Belgium, {koen.de_schepper|olivier.tilmans}@nokia.com
†Simula Research, Norway, olga@albisser.org
‡Independent, UK, research@bobbriscoe.net
¶The first two authors contributed equally

delay below about 50 ms [20]. Foreseeable non-network delays will at best consume 13 ms of that [36]. Then, reducing P99 queuing from say 30 ms to 2 ms (Figure 10) would leave 35 ms rather than 7 ms for two-way propagation — increasing reach in fibre from 700 km to 3500 km. Not to mention improving the responsiveness of other delay-sensitive applications like online gaming, interactive video and web.

Our further, more ambitious goal is to enable applications that are currently infeasible because they need both high throughput and low delay. These two are generally considered mutually incompatible without significant sacrifice of utilization. This is because high throughput implies capacity-seeking (or capacity adaptation), which has been the main cause of queuing as sources sawtooth or probe for capacity, even under steady-state conditions. Delay variation is particularly acute when the number of flows sharing a link is small ('low statmux'), which is the common case when the path bottleneck is in each user's access link. This adds the same order of delay as a typical base round trip time, i.e. a peak round trip time (RTT) of about twice the base RTT (see § II-A).

To escape this dilemma we need to go beyond an approach like Diffserv Expedited Forwarding (EF [23]), which offers very low delay but not full utilization — given it limits the capacity available to the EF class. We also want to go beyond the state-of-the-art in Active Queue Management, such as PIE or FQ-CoDel, which offer high utilization but not very low delay. These AQMs certainly remove excessive queuing, but they still have to buffer the sender's sawtooth variations.

To go beyond these network-only solutions we tackle both sender and network behaviour together. This makes evaluation challenging, because we cannot evaluate one change at a time, and comparisons of all the combinations of sender and network approaches would become too much for one paper.

Therefore, this paper confines itself to our foundational work on the network part — the novel 'Dual Queue Coupled AQM' ('DualQ') framework. 'Foundational' means the network part provides the groundwork — specifically high-fidelity congestion signals — on which more dynamic sender behaviours can be built. We only need steady-state traffic over fixed-capacity links to show that the network part works as intended. But we include some dynamic traffic scenarios, albeit only on fixed capacity links, in order to demonstrate the lowest feasible queuing delay under heavy dynamic load. § I-C fully explains this paper's scope.

B. Contributions:

Our main contribution is a new Internet service with an order of magnitude lower queuing delay at any percentile,

typically without sacrificing fixed-capacity utilization. This is relative to state-of-the-art congestion controls like CUBIC over state-of-the-art AQMs like PIE or FQ-CoDel.

We call this new service Low Latency, Low Loss, Scalable throughput (L4S). It is a general-purpose service for capacity-seeking and other traffic — like the Internet's best efforts service, but with very low and consistently low queuing delay.

L4S senders can use any one of the family of 'Scalable' congestion controls (CCs) (see § II-A for the definition of 'Scalable', suffice to say here that DCTCP [3] is a well-known example). With L4S (as with a DCTCP-enabled data centre), the congestion control at the sender is the primary mechanism that keeps delay low and utilization high, but it also needs shallow unsmoothed Explicit Congestion Notification (ECN [54]) in the network (§ II-B). For this paper, we use a derivative of DCTCP called Prague [29].

Our second contribution is a solution to the deployability of these Scalable controls, in coexistence with the traffic already on the public Internet. We use the term 'Classic' for this pre-existing traffic sent by Reno-Friendly[1] congestion controls like CUBIC [35] or Reno itself [5].

Classic CCs need a decent buffer (holding a typical RTT of data) to absorb their sawtooth window variations without underutilization. To avoid the need for this extra delay, Scalable CCs keep their sawtooth delay variation low by responding much less to each ECN mark relative to a Classic CC. However, 'less response' implies 'more aggressive', so Scalable flows would outcompete any Classic flows sharing an ECN-capable bottleneck queue.

To solve this 'coexistence problem', we propose the DualQ Coupled AQM ('DualQ') that can be incrementally added at path bottlenecks. It acts like a semi-permeable membrane: for delay it isolates L4S traffic from Classic in a separate queue; but for throughput it couples the queues to appear as a single bandwidth pool (see § III-B). Typical Internet bottlenecks are low stat-mux, so the capacity needed (e.g. number of flows) in either queue would be highly unpredictable, making it hard to allocate appropriate capacity to either. Instead, the coupling enables sources to share out the pool between themselves; it couples congestion signals across from the Classic to the L4S queue, but to counterbalance the more aggressive L4S sources it emits the L4S signals more aggressively.

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.

We have also tested that either queue can cope with a reasonable proportion of unresponsive traffic (e.g. unresponsive streaming, VoIP, DNS), just as classical best efforts can.

Our fourth contribution is a design that needs no reconfiguration once deployed. Two parameters are entirely absent by design, e.g. queue smoothing time and dynamic capacity sharing, which are controlled by senders.

Our fifth contribution (in § IV) is extensive quantitative evaluation of the above claims; not cherry picking results, but showing all metrics at once: i) dramatically reduced delay

variability without increasing other impairments; ii) limited impact on Classic traffic; iii) a good balance between competing Scalable & Classic flow rates; and iv) overload handling.

C. Scope:

One paper of reasonable length cannot cover the entirety of L4S in sufficient depth. So, the present paper focuses on the main new component—the DualQ Coupled AQM. This is a framework for coupling AQMs generally. So, to be concrete, we describe the DualPI2 algorithm (pronounced dual-pi-squared), and compare its performance with other state-of-the-art AQMs. A complementary paper provides the theoretical background on coupling Classic & Scalable AQMs, albeit in just one queue [26].

Unfortunately, all the following have had to be ruled out of scope in order to keep this paper to a reasonable size:

Detailed treatment of congestion control algorithms:
Other than a brief overview of the rationale in § II-A, and a roadmap of future necessary work in § VI-B, details of sender CC algorithms are given by reference.

Per-flow queue (FQ) AQMs with L4S support: L4S support has been added to the Linux implementation of FQ-CoDel.[2] However, the space needed to explain fine-tuning of the design and to compare with both classical FQ-CoDel and with a DualQ warrants a dedicated paper. Nonetheless, the pros and cons of a DualQ relative to an FQ solution are in scope (§ II-C).

More demanding link types: The DualQ has been implemented for Data Over Cable (DOCSIS) [56] and WiFi links, and simulated for 5G [57]. However, this paper limits discussion and evaluation to dedicated (e.g. DSL, switched Ethernet) rather than shared media link types.

Coexistence in pre-existing non-L4S queues: § V briefly discusses wider deployment considerations, including how a Scalable control should fall back to Reno-Friendly if it encounters a non-L4S bottleneck and other deployment scenarios such as coexistence between Scalable and Classic TCP in heterogeneous or interconnected data centres. A technical report is available on detection of whether an L4S flow is bottlenecked at a Classic ECN AQM [13], and full results with and without detection are available[3].

II. RATIONALE

A. Why a Scalable Congestion Control?

A congestion control (CC) is defined as Scalable if, in steady state, the average time from one congestion signal to the next (the recovery time) does not grow as the flow rate scales up, all other factors being equal [27]. Here, 'congestion signal' means either loss or ECN, and response to delay is not considered (but not disallowed).

Standard Reno CC [5] has become very slow to recover from any disturbance, because its recovery time (or sawtooth

---

[1]Reno-Friendly is a more precise term for TCP-Friendly.

[2]"fq_codel: generalise ce_threshold marking for subset of traffic", https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.git/commit/?id=dfcb63ce1de6b10b

[3]https://l4s.net/ecn-fbk/

duration) grows linearly with flow rate [33]. CUBIC [35] scales better, but it is still not fully scalable. For example, for every 8-fold rate increase, CUBIC's recovery time doubles; e.g. from 100 to 800 Mb/s, recovery time extends from 250 to 500 round trips (for base RTT=20 ms).

We now derive the condition for a CC to be scalable, at least for those with a response to congestion of the form $W \propto 1/p^B$. Here the steady-state window, W, responds to the probability p that a packet carries a congestion signal, and B is a characteristic constant of the algorithm (e.g. $B = \frac{1}{2}$ for Reno or $\frac{3}{4}$ for CUBIC).

In steady state, the number of signals per round, v, is the product of the segments per round W and the probability p that a segment carries a signal, i.e. $v = pW$. Or, substituting for p from the response function:

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

The recovery time is the inverse of v. So, by the earlier definition, v must not reduce as W increases. Therefore, $(1 - 1/B) \geq 0$. So, $B \geq 1$ defines a control as Scalable.

For DCTCP [3] or Prague [29], $B \geq 1$, and with probabilistic marking $B = 1$ (see § III-B), so they are Scalable. The algorithms are designed to induce a high average signalling rate (low recovery time) of two ECN marks per round[4], and that remains invariant whatever the rate.

To allow the window to scale, it is important to prevent the recovery time from growing so that:
1) control does not slacken and it does not become more sensitive to noise from transmission losses or transient queuing.
2) sawtooth amplitude can be small, and stay small.

The schematic in Figure 1 gives the intuition for this latter point. With a large amplitude Classic sawtooth, either queue variability is high (1) or under-utilization is excessive (2). But, with low amplitude sawteeth, both can be minimal (3).

The three columns of results in Figure 2 reinforce these three points empirically; with CUBIC's large Classic sawteeth
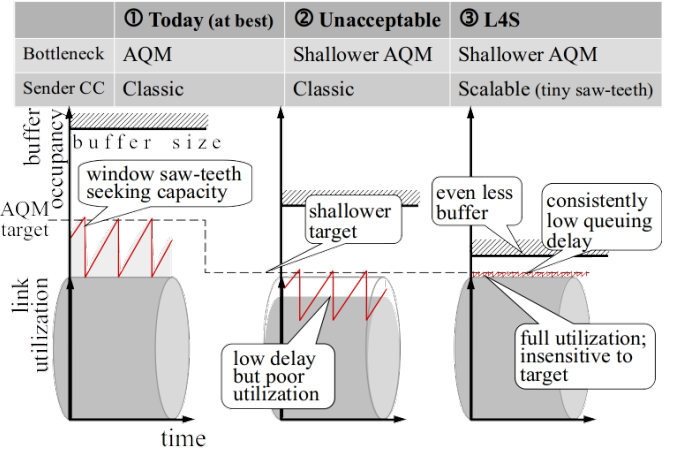
[4]Per 'virtual RTT' when Prague is in RTT-independence mode (see [29]).

Fig. 1: Scalable Congestion Control: Intuition

((1) & (2)), you cannot have both low delay and low under-utilization. But with Prague's tiny sawteeth (3) you can.

The DCTCP analysis [4] calculates that full utilization in steady-state requires the ECN threshold to be set to $\geq 17\%$ of the RTT. But it also says that utilization is fairly insensitive to this setting. We can confirm that a threshold more than an order of magnitude shallower can be used without noticeable under-utilization.[5]

The dynamic behaviours of DCTCP & Prague are not scalable, mainly because they still use an additive increase of one segment per RTT, like Reno. However, the dynamic behaviour can be continuously improved after initial deployment (see § VI-B), whereas the steady-state behaviour is hard to change, once established.

For all experiments in this paper, TCP Prague [16] is used in its default configuration[6]. Prague was developed as the reference CC that implements the 'Prague L4S Require-

[5]Indeed, because the sawteeth are so small, they can either sit just within the buffer, as shown, or just below full utilization, by using a virtual queue, as in HULL [2].

[6]for source code of the exact version used, see https://github.com/L4STeam/linux/blob/66331636f0dd4930/net/ipv4/tcp_prague.c
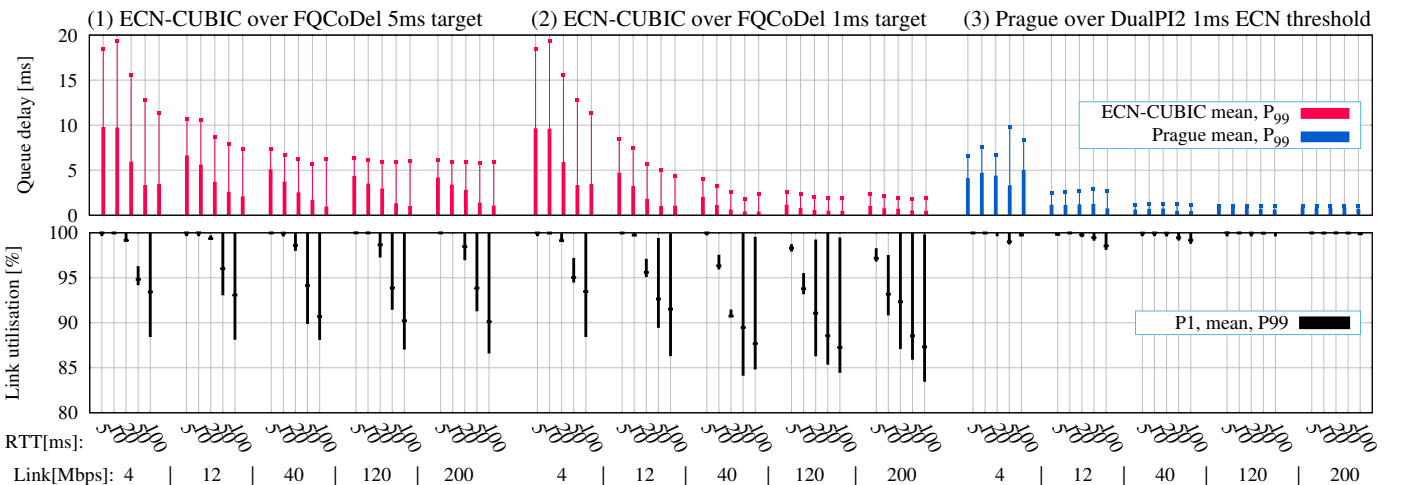

Fig. 2: With a long-running Classic (ECN-CUBIC) flow, reducing the queue target of the FQ-CoDel AQM from 5 ms to 1 ms (left & middle) makes utilization suffer badly for all but the lowest RTTs. Whereas a Scalable (Prague) flow with small sawteeth can keep to a 1 ms ECN threshold without compromising utilization (right).

ments' [27]. It is based on DCTCP, but with the improvements and modifications for use over the Internet specified in [29] that have been implemented over TCP and QUIC. Other scalable controls have been developed, e.g. the ECN-capable part of BBRv2 [19] over TCP or QUIC and a scalable variant of SCReAM [57] for real-time media over RTP.

B. L4S Explicit Congestion Notification (ECN)

L4S uses the same protocol fields as standard ECN [54], but defines new semantics for an ECN mark. This breaks away from its previous equivalence to loss, as recently allowed by the IETF [8], so that L4S can evade the compromises that are inherent in using drop as a congestion signal. This is because drop is also an impairment, so it cannot be signalled too frequently or too immediately. Specifically, L4S exploits ECN to reduce delay in three respects:

1) L4S ECN allows more frequent signalling, which would be untenable as loss, particularly during high load. This facilitates the smaller sawteeth of scalable controllers, which reduce delay as already explained.
2) When a queue starts to grow, a drop-based AQM holds back from introducing loss in case queue growth turns out to be transient. In contrast, an L4S AQM can emit ECN immediately, because it is not also an impairment:
   - With drop, an AQM has to hold back from drop for about 1 RTT. But it does not know each flow's RTT, so it has to hold back for a worst-case (inter-continental) RTT, to avoid causing instability to worst-case RTT flows.
   - With ECN, the AQM can signal immediately, and the sender can smooth the signals—it knows its own RTT, which it can use as the appropriate smoothing time [4] or it can choose to respond without smoothing when appropriate, e.g. at flow start.
3) ECN also offers the obvious latency benefit of near-zero congestion loss, which primarily benefits short flows, as shown by Salim & Ahmed [55]. This removes retransmission and time-out delays and the head-of-line blocking that a loss can cause when a transport with ordered delivery (like TCP) carries a multiplex of streams.

Because L4S requires Scalable traffic to be ECN-capable, it overloads the spare ECN codepoint as the identifier of Scalable packets, for classification into the L4S queue (see § III).

C. Why Is Per-Flow Queuing Not Enough?

Per-flow queuing (as in FQ-CoDel [39]) is intended to isolate a latency-sensitive flow from the delays induced by others. However, FQ alone does not protect a latency-sensitive flow from the saw-toothing queue that a Classic flow still inflicts upon *itself*.[7] This is important for the growing trend

---

[7]It might seem preferable to release data into a dedicated network queue, then: a) it would be ready to go as soon as there was capacity; and b) otherwise the sender would have to hold back the data instead, causing the same delay, just in a different place. However, modern applications, e.g. HTTP/2 [7] or interactive video, need to maintain any self-induced send-queue locally so, at the last moment, they can decide what to send next dependent on the very latest user behaviour and feedback. They cannot alter data already in flight.

of rate-adaptive interactive video-based apps that are both extremely latency-sensitive and capacity-hungry, e.g. interactive or conversational video, remote presence.

Support for scalable congestion controls has been added to FQ-CoDel to address the self-inflicted delay problem.[2] It adds shallow-threshold ECN marking for L4S ECN packets while still directly enforcing throughput equality between Scalable and Classic flows (rather than the DualQ's indirect approach).

However, some network operators still consider that some or all of the following are uncomfortable compromises inherent in per-application-flow queuing (FQ):

Privacy: Layer-3 VPNs hide transport layer headers for privacy, but FQ needs to inspect flow IDs to isolate flows. So FQ does not allow both flow-privacy and low delay.

Neutrality: FQ does not know i) whether a flow using more, or less, than an equal share of a user's own capacity is intentional, or even mission-critical; ii) whether short-term flow rate variations are deliberate, e.g. a more complex video scene; iii) whether a real time congestion control is deliberately adapting slowly to changing numbers of competing flows, in case they are transient.

Simplicity: FQ requires more complex classification, queuing and scheduling structures.

The DualQ was developed for those operators that want to offer low delay service without being obliged to face these compromises. With a DualQ, an operator can still choose to add a flow policer to enforce equal flow-rates, but as an independent policy choice, not as a non-optional side-effect of reducing delay.

III. DualQ Coupled AQM: Solution Design

The solution will be explained in two passes. The first pass introduces the overall structure (§ III-A). Then details of each aspect are given in a second pass.

A. Solution Structure

a) Latency Isolation: L4S and Classic traffic have opposing delay requirements. The first design goal of L4S traffic is very low queuing delay. In contrast Classic congestion controllers (CCs) need a significant queue to avoid underutilization (and other impairments), in the common low statmux bottleneck case. One queue cannot satisfy these opposing goals, so we use two separate buffers.

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 [54]. 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 § V-B).

b) Coexistence: An L4S CC such as Prague achieves low latency, low loss and low rate variations by driving the network to give it frequent ECN marks. A Classic CC (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, meaning rough balance between their steady-state packet rates. This problem has already
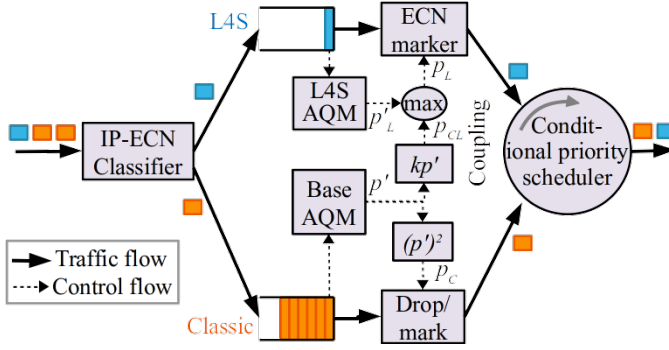
Fig. 3: Dual Queue Coupled AQM: Structure

been solved in the single-queue coupled AQM [26] by inflating the signal intensity applied to L4S traffic to compensate for its reduced response to each signal. In the DualQ case, the congestion signals that the Classic AQM applies to Classic traffic due to its own queue are also coupled across to be applied to L4S traffic, but with increased intensity (see § III-B).

c) Scheduling: Splitting the traffic into two queues raises the question of how often to schedule each queue. We do not want to schedule based on the number of flows in each, because we want to avoid flow identification (see § II-C). Instead, for the most part, the scheduler is arranged not to interfere, so that sources can 'schedule' themselves—just as they would in a FIFO, where the level of congestion signalling rises until each flow draws back just enough to make space for all the other flows. In the DualQ, the coupling of congestion signals from the Classic to the L4S queue makes L4S flows draw back to leave just enough space for the Classic flows.

Figure 4 illustrates this in the example case of one Classic and one L4S flow. It shows only the L queue, to illustrate that the L flow is leaving it empty about half the time.

Bearing in mind that Classic flows tend to build a queue, using a work-conserving strict priority scheduler in favour of the L queue will then be sufficient (superficially, at least). It will serve L packets whenever present (half the time in the example), otherwise it will drain the C queue.

The priority scheduler ensures any queue of C packets yields to L packets. While coupled congestion signals from the C to the L queue make L traffic yield to C. These two opposing mechanisms counterbalance each other, nullifying any bandwidth advantage of priority scheduling. Nonetheless, to avoid short-term starvation of Classic traffic, priority has to be conditional, not strict, as will be explained in the second pass through the design (§ III-C).
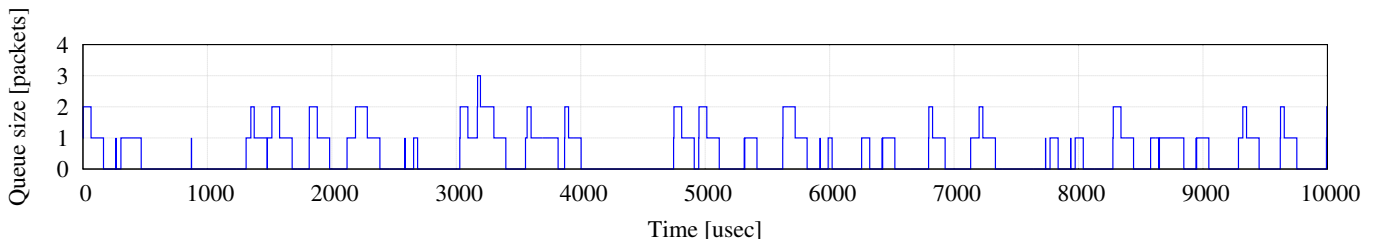
d) The whole picture: The schematic in Figure 3 shows the whole 'Dual Queue Coupled AQM', with the classifier and scheduler as the first and last stages. The detail in between is covered next, but it can be seen that each queue has its own native AQM that works whether or not the other queue is empty, while the coupled signals flow across from the Classic AQM to the L4S marker.

B. Coupled AQM for Window Balance

To derive the strength of the coupling between Classic (C) and L4S (L) congestion signals, we take the equations for the steady-state packet rate, r, of two sender CCs as functions of congestion signalling probability p. Then, equating the two packet rates will give the necessary strength of one congestion signal relative to the other.

The packet rate of most CCs also depends on the end-to-end RTT ($R = R_b + Q$), where R_b is the base RTT and Q is the queue delay. The 'fairness' of CCs has generally always been judged under equivalent conditions, meaning the same RTT, packet size, etc. But with the DualQ, a C and an L flow with the same base RTT will have different e2e RTTs, because the steady-state queue delay (Q) is considerably smaller for L flows. Therefore, before deriving a coupling formula, we need to briefly digress into the RTT-dependence of L4S and Classic congestion controls.

a) RTT-dependence: If the rates of two CCs depend inversely on their RTT then, taking example extreme base RTTs of 4 ms and 100 ms, the worst-case rate ratio will be $(100 + Q)/(4 + Q)$. If we plug in values of Q as technology evolves from tail drop buffers (c.100 ms); through state-of-art AQMs (c.10 ms); to the L4S AQM ($\sim$500 μs), this worst-case ratio explodes: 2, 8 then 22. Thus, 'RTT-unfairness' evolves from a non-problem into a potential starvation problem.

But blame does not lie with lower queuing delay, which has only unmasked the problem. Blame lies with window-based CCs, and that is where the problem should be solved.

There is no reason why congestion control has to be inversely proportional to RTT.[8] Admittedly, the RTT-dependence of existing Classic traffic cannot be 'undeployed'. Nonetheless, the evaluations in § IV-E show that it is possible and sufficient to address RTT-dependence solely in L4S senders, as also required by [27]. This addresses the problem for L4S vs. L4S flows, and does not significantly worsen the problem for L4S vs. Classic compared to Classic vs. Classic.

---

[8]Although it can still depend on RTT outside steady-state.

Fig. 4: L4S queue size over time with 1 long Prague flow and 1 long CUBIC flow in each queue into a 120 Mb/s link with 10 ms base RTT. Due to the coupling, the L4S flow leaves the right amount of scheduling opportunities for the Classic queue.

b) Coupling strength: We take the approach of equalizing the rates of L and C flows at one typical ('reference') base RTT common to both L and C flows (R_b^*). As just discussed, flows with significantly different RTT from the reference should not have significantly different rate: for Classic flows this is due to the cushioning effect of their queue; and for L4S flows it is by design of their RTT-independent CC algorithm.[9]

For C and L flows we use the Reno and Prague equations in Equation 1 and Equation 2, where p_{CL} is the coupled signal as defined in Figure 3. We use Reno because it is the worst case (weakest) and it is widely used, mostly as CUBIC in Reno-compatibility mode (which we call CReno). Coexistence concerns steady-state conditions, so we can use the simplified steady-state Reno equation from [48]. Prague has similar steady-state behaviour to DCTCP, but we do not use the equation from the DCTCP paper [3], which is only appropriate for step marking. Instead, we use the equation that is appropriate to our coupled AQM, where marking is probabilistic, as derived in Appendix A of [26]. Further, we divide by $f(R_L)$, which is a generic function that represents the various ways that have been proposed to reduce Prague's RTT-dependence [29], [15].

$$r_C = \frac{1}{R_C}\sqrt{\frac{3}{2p_C}} \quad (1) \qquad r_L = \frac{2}{f(R_L)p_{CL}}, \quad (2)$$

Then, equating the two rates at R_b^* results in Equation 3:

$$p_C = \frac{3}{8}\left(\left.\frac{f(R_L)}{R_C}\right|_{R_b^*} p_{CL}\right)^2 \qquad := \left(\frac{p_{CL}}{k}\right)^2. \quad (3)$$

We group all the constants into coupling factor k in Equation 3. In Appendix A values of $k = 1.96$ and 2.22 are derived, depending on whether the Classic CC is Reno or CReno. In our implementation we use $k = 2$, both because it is the round value for the worst-case (Reno) and because it is an integer power of 2, which makes implementation efficient.

c) Classic AQM and Coupling: The coupling is implemented by structuring the Classic AQM in two stages (Figure 3). 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, $p_C = (p')^2$. But for L4S traffic it is applied linearly, $p_{CL} = k * p'$. Substituting for p' from the latter into the former proves that the coupling between p_{CL} and p_C will conform to Equation 3, as required.

Diversity of Base AQMs is possible and encouraged. Three have been implemented: a variant of RED called Curvy RED [28, Appx. B], The DualQ specified for Low Latency DOCSIS [56] based on PIE, and Linux DualPI2 [1]. They all control queuing time not queue size, given the rate of each queue varies considerably [47]. This paper assesses our open source DualPI2 implementation in Linux.

The DualPI2 AQM builds on PI² [26], which also couples two AQMs to enable coexistence of different CCs. But, unlike

---

PI², the coupled AQMs are applied in separate queues, with separate delay targets. Nonetheless, the coupling between the AQMs makes flows behave as if they are using a single pool of capacity. This builds on the theoretical and experimental proof in [26] that squaring the output of a PI controller is a more effective, more principled and simpler way of controlling Reno (rate proportional to $1/\sqrt{p'}$) than PI Enhanced (PIE [51]). It has the added advantage that the controller's direct (unsquared) output can be used to control a scalable CC like Prague as well (rate proportional to $1/p'$). The derivation of the coupling formula is more straightforward in the single-queue case of [26], because C and L flows share a common queue delay, so the RTTs cancel out.

C. Dual Queue for Low Latency

a) Native L4S AQM: 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. However, unlike DCTCP, T is cast in units of time [47], [6], so that it is invariant with dequeue rate, which can vary considerably. In case of a low rate link, T has a floor of two packets. On-off marking may [21] or may not [46, §5] be prone to instability. But to test one change at a time we defer investigation of alternatives such as a ramp to future research.

If there is traffic in both queues, an L4S packet can be marked either by the native L4S AQM or by the coupling from the Classic AQM, whichever outputs higher probability (illustrated as the max() function in Figure 3). Marking via the coupling generally ensures that L4S traffic stays below its native threshold (Figure 4), only touching it during bursts or if there is insufficient Classic traffic.

Note that Classic AQMs filter out rapid variations in the queue before it drives dropping or marking, which is necessary to stabilize Classic CCs, but it delays the signals. In contrast, as §II-B explained, the L4S AQM emits ECN marks without delay, and L4S sources only smooth the marks if they need to, e.g. during their congestion avoidance phase.

b) Conditional Priority Scheduler: Earlier, an explanation was promised for why the scheduler's priority has to be conditional. We found that strict priority worked fine in steady-state, but sometimes there would be a brief deadlock when a DNS request or the initial packet(s) of a Classic flow arrived at an empty C queue, while L traffic was keeping the L queue busy. This was because, no matter how long a blocked C packet waited, our implementation didn't increase the coupled marking of L packets, because it only measured a packet's delay when it was dequeued, not while it was waiting. The C packets were only ultimately released when there happened to be a gap in the L traffic.

Conditional priority, such as a weighted round robin (WRR) scheduler with a high weight in favour of the L queue (e.g. $^{15}/_{16}$ or $^9/_{10}$) resolves this deadlock. The actual weight is immaterial, because capacity shares are determined by the response of L4S senders to the coupled signal (not by the scheduler). The weight for Classic traffic only has to be large enough (e.g. $^1/_{16}$ or $^1/_{10}$) to release the deadlock.

---

[9] Our worst-case result was the 'A5:B100' scenario in Figure 8, with a rate ratio of 6.3 for 5 ms Prague vs. 100 ms Reno over DualPI2. This compares with a ratio of 5.5 for 5 ms CUBIC vs. 100 ms Reno over PIE (a harm ratio of 1.14). We believe this is within our "not significantly worse" goal.

We also tried a MEDF scheduler [49], which we call a Time-Shifted FIFO. It selects the packet with the earliest arrival timestamp, after subtracting a constant time-shift to favour L4S packets. It performs nearly as well as WRR despite its simplicity. However, it allows bursts of delay to leak from the C to the L queue, so it is not used further in this paper.

D. Overload Handling

Having introduced a priority scheduler, during overload we must at least ensure that it gives unresponsive traffic no more power to harm other traffic than a single queue would. We prefer to leave flow policing as a policy choice (see § II-C). So, the coupled AQM allows unresponsive traffic below the link rate to just subtract from the overall capacity, whether it classifies itself as L4S or Classic. Then it still allows any responsive flows to share the remaining capacity — as they would in a single queue with the same capacity subtracted.

To handle excessive unresponsive traffic, the Base AQM is actually driven by whichever queue is greater[10], and when its output probability exceeds a threshold it applies drop to ECN-capable packets in either queue with the same probability as Classic (i.e. the coupling becomes bidirectional and equal). By default, this overload threshold is set where the L4S AQM saturates at 100% marking. By equation (3) this occurs once Classic drop reaches $(100\%/k)^2$, which is 25% if $k = 2$.

When any L4S source detects a drop, it is required to react as a classic flow would [27], so balance between flow rates is preserved. The L4S AQM also continues to ECN-mark so that, under unresponsive Classic load, any responsive L4S traffic can maintain the very low queuing delay of the L4S service. These claims are verified in § IV-G.

E. Implementation

Algorithm 1 Dequeue for Dual Queue Coupled AQM

1: **while** LQ.LEN() + CQ.LEN() >0 **do**
2:      **if** SCHEDULER() == LQ **then**
3:          LQ.DEQUEUE(pkt)   ▷ L4S
4:          p'_L = LAQM(LQ.TIME())
5:          p_L = MAX(p'_L, p_{CL})
6:          **if** $p_L >$ RAND() **then**
7:              MARK(pkt)
8:      **else**
9:          CQ.DEQUEUE(pkt)   ▷ Classic
10:         **if** $p_C >$ RAND() **then**
11:             **if** ECN(pkt)==0 **then**   ▷ Not ECT
12:                 DROP(pkt)   ▷ Squared drop
13:                 **continue**   ▷ Redo loop
14:             **else**   ▷ ECT(0)
15:                 MARK(pkt)   ▷ Squared mark
16: RETURN(pkt)   ▷ return the packet, stop here

Algorithm 1 summarizes the per packet dequeue implementation of DualPI2 as pseudocode. The AQMs are applied at dequeue to minimize signalling delay. For clarity, overload logic and edge cases are omitted, but they can be found

---

[10]Even if the L4S queue is the greater, e.g. no C traffic, when not overloaded, it stays well below the Classic target. So it drives p' to zero.

in the open-sourced implementation of the DualPI2 Linux qdisc [1], or as pseudocode in [28, Appx. A.2]. The function LEN() returns the the queue in bytes, while TIME() returns the duration since a packet was time-stamped (its sojourn time).

On enqueue (not shown), packets are time-stamped and classified based on the least significant bit of the IP-ECN field.

On dequeue, line 2 determines which head packet to take. For this paper we use WRR as already explained, but the pseudocode is generalized for any work-conserving scheduler.

If an L4S packet is scheduled, line 4 runs the native L4S AQM (referred to as LAQM()) to output probability p'_L dependent on the service time of the L queue. This is a generalization for whatever native L4S AQM is used, but for the present paper we use a simple [0,1] step function at delay threshold T. Line 6 marks the packet if a random marking decision is drawn according to the probability p_L, which the previous line has taken as the max of the outputs of the native L4S AQM (p'_L) and the coupling (p_{CL}). The latter is maintained by Algorithm 2.

If a Classic packet is scheduled, line 10 decides whether to emit a congestion signal with probability p_C, which is also maintained by Algorithm 2, as explained in § III-Bc). Then line 11 checks whether the Classic packet is not ECN-capable, in which case it uses drop as the signal, otherwise it uses ECN.

The smoothed marking and dropping probabilities are kept up to date by Algorithm 2 which only needs occasional execution [41] every T_{update} (default 16 ms). Line 1 bases the algorithm on the max of the two queues, which handles overload as explained earlier. Then the core PI algorithm updates the internal base signalling probability (p'). The change in queuing time is multiplied by the proportional gain factor β. The integral gain factor α is typically smaller, to restore the delay of any persistent standing queue to the target. These expressions, which can be negative, are added to the previous p' every T_{update}. Then the Coupled and Classic signalling probabilities, p_{CL} and p_C, are derived from p' (see § III-B).

Algorithm 2 PI core: Every T_{update} p is updated

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

IV. EVALUATION

A. Testbed Setup

Experiments were first conducted on a testbed that used realistic DSL equipment, then on a simpler testbed capable of extending over a wider range of BDPs. The DSL testbed was assembled using carrier grade equipment in the same environment as for testing customer solutions, consisting 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 was connected by VDSL to a DSLAM, which was connected to the BNG through an aggregation network, representing a local ISP

or access wholesaler. Traffic was routed to another network representing a global ISP that hosted the application servers and offered breakout to the Internet. The Linux client and server in either pair A or B, were always configured with the same TCP variants and applications.

In a production access network, the BNG is usually deliberately arranged as the downstream bottleneck for each customer (to confine any QoS handling to one point). Traffic from the client-server pairs was routed from the downstream interface of the BNG through a Linux 'AQM server', which acted as the BNG's rate bottleneck where we configured different AQMs to be evaluated. This server also added extra delay, controlled the experiments, captured the traffic and analysed 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 xDSL line was configured at 48 Mb/s downstream and 12 Mb/s upstream. Ethernet links between network elements consisted of at least 1GigE connections, except 100 Mb/s between clients and modem. The base RTT between the clients and servers was 7 ms, which was primarily due to the interleaved Forward Error Correction (FEC) configured for xDSL. Extra base RTT was configured using a netem qdisc on the upstream AQM server interface or on the clients for experiments with different base RTTs.

All experiments presented in this paper were performed on the simpler testbed, which consisted of 5 nodes (2 clients, 2 servers, an AQM server) and 2 dedicated switches, each connecting the clients and servers to the AQM server with 1GigE connections (shown in Figure 5). A consistency check between the two testbeds within the narrower limits of the xDSL testbed showed near-identical results.
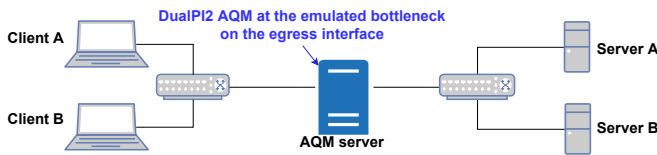

Fig. 5: Simpler testbed topology

All Linux computers were Ubuntu 18.04/20.04 LTS, running an out-of-tree kernel containing TCP Prague and DualPI2.[11] We used Prague for the Scalable congestion control and CUBIC for Classic, both were bundled with the installed kernel version and used with default configuration. For ECN-CUBIC, we enabled TCP ECN negotiation. We compared DualPI2 with PIE and FQ-CoDel, all configured as in Table I. The α and β values for PIE are equivalent to those used for DualPI2, but PIE scales the input parameters internally. All AQM parameters were left at their default values.

B. Experimental Approach

The set of experiments was constructed to evaluate our main performance goals: queuing delay, utilization and rate balance for most experiments, and flow completion efficiency

---

All	Buffer: 40,000 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, TUpdate: 16 ms, L4S T: 1 ms, Classic weight: 10%, α: 0.16, β: 3.2, k: 2, drop_on_overload

TABLE I: Configuration parameters for the different AQMs.

for experiments with short flows.[12] We also show window balance and drop/mark probability as secondary evaluation metrics, mainly to support interpretation of the primary effects. The evaluation metrics are fully specified in Appendix B.

For traffic load we used long-running flows (§§ IV-C to IV-E) and/or dynamic short flows (§ IV-F). We used long flows, not as an example of a realistic Internet traffic mix, rather to aid interpretation of various effects, such as starvation. Heavy load scenarios predominate in our choice of experiments, again not because they are typical, but because they do occur and they are a challenging case.

We mixed different numbers of flows, different congestion controls (CCs) and different RTTs. Also, to verify behaviour with unresponsive flows and overload (§ IV-G), we injected unresponsive UDP load; both below and above link capacity, and both ECN and Not-ECN capable.

As explained in § I-C, we have reported separately on our experiments with combinations of CCs that an AQM is not intended to support.[13] Here we choose examples of typical combinations that are supported: Prague with CUBIC on DualPI2[14] and ECN-CUBIC with CUBIC on current AQMs (PIE and FQ-CoDel). In some cases, we added experiments with Reno instead of CUBIC to check that there was little difference, given CUBIC remains largely in its Reno-friendly mode with the BDPs used in these experiments, which were chosen as typical of 'data centre to the home' scenarios.

Most of our evaluation is comparative, showing the results for each of the AQM and CC combination, apart from the scenario with unresponsive flows and overload, where only results for DualPI2 are presented[15].

C. Basic Steady State Experiments

a) Experimental setup: In the first set of experiments the competing flows in each experiment had equal base RTTs. For the most basic setup, we used two long running flows; one for each CC and ran experiments with all 25 combinations of 5 base RTTs (5–100 ms) and 5 link speeds (4–200 Mb/s) in each of the three columns; one for each AQM, totalling to the 75 scenarios across Figure 6. Each row of the figure reports a different metric: queue delay (linear and log-scale), link utilization, rate & window ratios and mark/drop probability.

b) Results: The primary goal of the DualQ is very low queue delay for L4S traffic, while ensuring coexistence with Classic traffic. So, compared with Classic AQMs, Classic

---

[11]5.10.31-3cc3851880a1-prague-37 see https://github.com/L4STeam/linux/tree/3cc3851880a1

[12]Source data for all presented experiments is available in https://github.com/olgaalb/dualpi2eval

[13]See https://l4s.net/ecn-fbk/results_v2.2/full_heatmap_rrr/ and [13]

[14]We do not report results with ECN-CUBIC over DualPI2, which were hardly any different to CUBIC.

[15]Comparative overload experiments against the other AQMs are available at https://l4steam.github.io/overload-results/
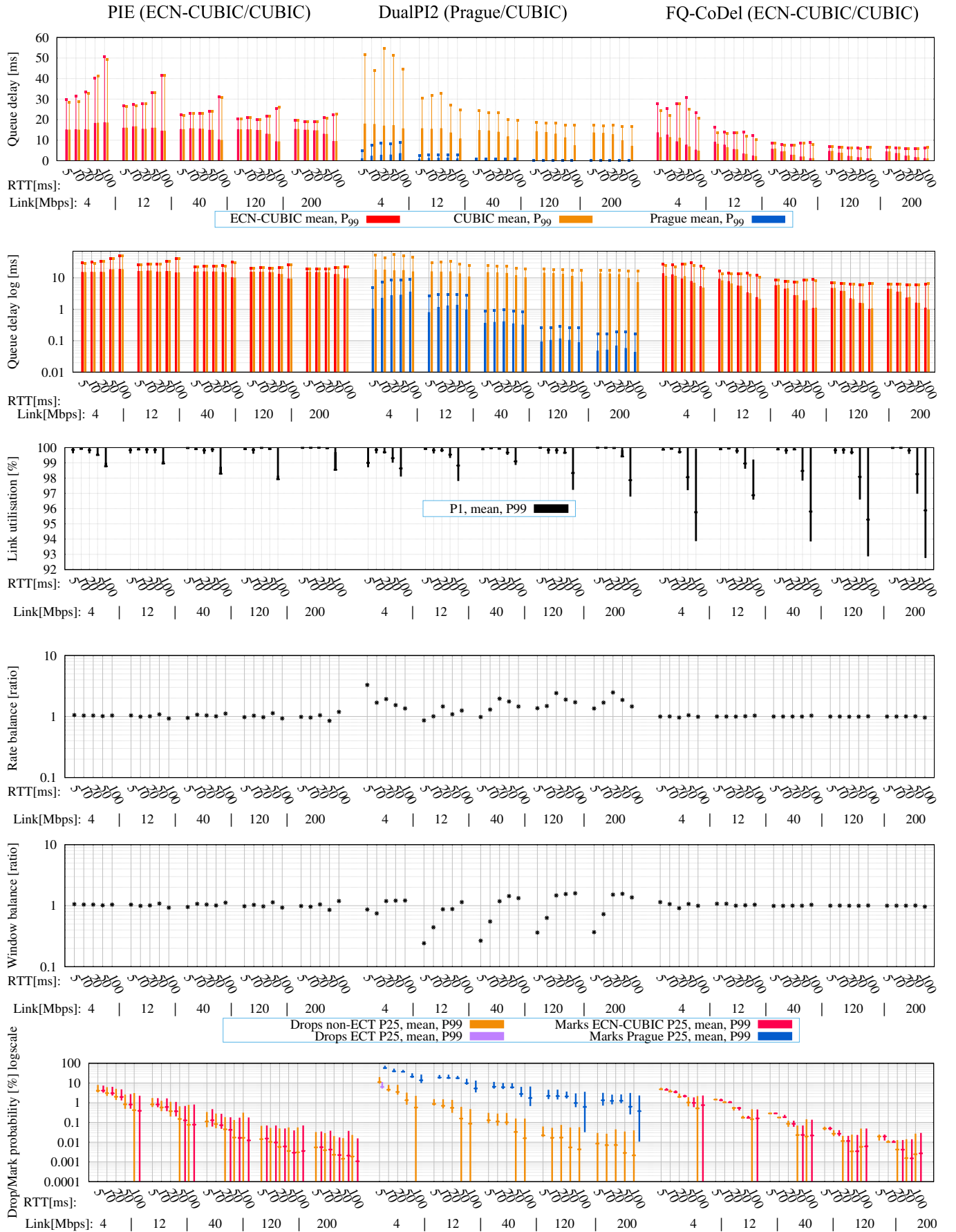
Fig. 6: Basic Experiments: DualPI2 compared to PIE & FQ-CoDel; 1 steady-state flow for each CC with equal base RTT

queueing delay is not expected to be better, but it should be no worse. It is also important to keep a reasonable balance between flow rates despite the different RTTs.

Overall, Figure 6 shows that indeed DualPI2 enables scalable congestion controls (Prague) to achieve an order of magnitude lower queueing delay than the other AQMs with minimal impact on Classic traffic. Importantly, P99 queue delay is also an order of magnitude lower.

The log plot in the second row of Figure 6 shows how the DualPI2 **queue delay** approaches the minimum possible — the mean at each link rate translates to 1 packet of serialization delay and each 99th percentile translates to 2–3 packets — consistent with the example in Figure 4.

This very low delay is achieved with hardly any harm to the Classic traffic. DualPI2 keeps average Classic delay (CUBIC) mostly at the expected target of 15 ms. The 99th percentile is somewhat higher than PIE for the lowest link rates, but it is lower in other cases. Although the Classic queuing delay of DualPI2 is not as low as that of FQ-CoDel, DualPI2 does not underutilize the link as much as FQ-CoDel. As was shown in Figure 2, Classic congestion controls create a dilemma between queue delay and under-utilization. So, the target delay of PIE or PI2 could be reduced from the default configuration to match the delay results of FQ-CoDel, but only by choosing to sacrifice more utilization, as FQ-CoDel does.

The fourth and fifth rows of Figure 6 show **rate** and **window** balance, which are almost perfect for PIE and FQ-CoDel. This is unsurprising given the two flows are essentially identical (except one supports Classic ECN, which is not expected to alter the flow rate). For FQ-CoDel the results are even flatter due to the round robin scheduling.

The rate balance of flows with the same base RTT is not so perfect over DualPI2 because the greater Classic queue causes greater total RTT and Classic congestion controls are RTT-dependent. As explained in §III-B, the proper place to reduce RTT-dependence is in newly deployed L4S congestion controls, and that has indeed been implemented in Prague.

Over the range of scenarios tested, it can be seen that Prague keeps the rate ratio within acceptable bounds; roughly between 0.85 & 2.5, except for the very smallest BDP[16]. Prague's RTT-independence kicks in below 25 ms. This can best be seen in the plots of window balance which stay relatively flat as base RTT is reduced from 100 ms. Then the points drop away below 20 ms. Equivalently, as RTT reduces, the rate balance plots rise but then fall back towards 1 for RTTs below 20 ms.

The last plot of Figure 6 shows how each AQM applies mark and drop probability. It is clearly visible that DualPI2 doubles the signal for Prague and squares it for CUBIC, whereas PIE and FQ-CoDel apply the same signal to both flows.

We conclude that in steady state DualPI2 can achieve near minimum mean and P99 queue delay without compromising utilization and preserving approximate rate fairness.

D. Multiple long-running flows

a) Experimental setup: As in the basic steady-state experiments, this scenario still involves the same two types of flows (ECN & non-ECN, labelled A & B) competing over each AQM. However, here there is not just one flow of each type, but 0–10 of each, as specified along the X-axis of Figure 7. In the second row, we also show results using Reno in place of CUBIC, which illustrates that there is no significant difference.

The Y-axis shows the rate per flow normalized relative to the 'fair' rate, meaning the link rate divided by the total number of flows. Figure 7 shows results over 10 ms base RTT and 40 Mb/s capacity, which are representative of our other tests (not shown) over the 5 link rates and 5 RTTs used in Figure 6.

b) Results: In the PIE and FQ-CoDel cases, all the flows have similar CCs, which results in near equal **per-flow rate**. The FQ-CoDel scheduler keeps the percentile range very tight. Except we sometimes see lower 1%-ile (P1), when a hash collision maps more than one flow into the same queue.

DualPI2 ensures no flow gets much below the 'fair' rate. When many Prague flows compete with few CUBIC flow (A8-B2, A9-B1), each Prague flow gets a little less, and the few CUBIC flows then consume the extra capacity. We traced this to reduced congestion signalling applied to the Classic flows because the weighted scheduler gives Classic traffic priority over approx. 10% of the capacity.[17] This confirms that the simple squared coupling of the DualPI2 AQM adequately counterbalances the more aggressive response of Prague for many different numbers of flow-types, not just 1-to-1.

E. Different RTTs

a) Experimental setup: To evaluate the RTT-dependence of different CCs, we run the same pairs of CCs (labelled A or B) over the same AQMs as in the basic steady-state experiments, but with differing RTTs. For flow B, we also use Reno to check comparability with CUBIC, and Prague to check behaviour with itself. The base RTT of A is either 5 or 100 ms and that of B is taken from the range 5, 10, 20, 50 and 100 ms, as denoted along the X-axes of Figure 8. For example, 5-20 means 5 ms for flow A and 20 ms for B. Runs with no flow B are also included. Results over a 40 Mb/s link are shown, which are representative of tests run over the other 4 link rates (not shown).

b) Results: The zoomed inset in the top row of Figure 8, shows that even with longer RTT flows, Prague's mean **queue delay** on DualPI2 is below 0.5 ms for all tested cases, which is significantly lower than that achieved by Classic flows over PIE or FQ-CoDEL. Note that even Prague's P99 delay is always below 1 ms, when it is driven by both the 1 ms step threshold and coupled congestion marking. When running alone, its P99 is slightly above 1 ms both for small and high RTTs, which is because its congestion marking is solely from the step threshold. The mean and P99 of DualPI2's Classic queue delay is lower or similar to PIE's. Indeed, as Classic RTT rises the mean and P99 diverge in the A5 cases over

---

[16]Here DualPI2 occasionally tips into overload mode, where the last row of plots shows that some drop is applied to the L4S queue. Prague's RTT-independence algorithm scales down the window so currently it hits the minimum window sooner, causing the higher rate ratio shown. It is planned to reduce the minimum window to compensate for this effect.

[17]Also, Prague gets additional marking when L4S queue delay goes above the 1 ms threshhold, which leaves Classic traffic with an even greater link share than the scheduler allocates.
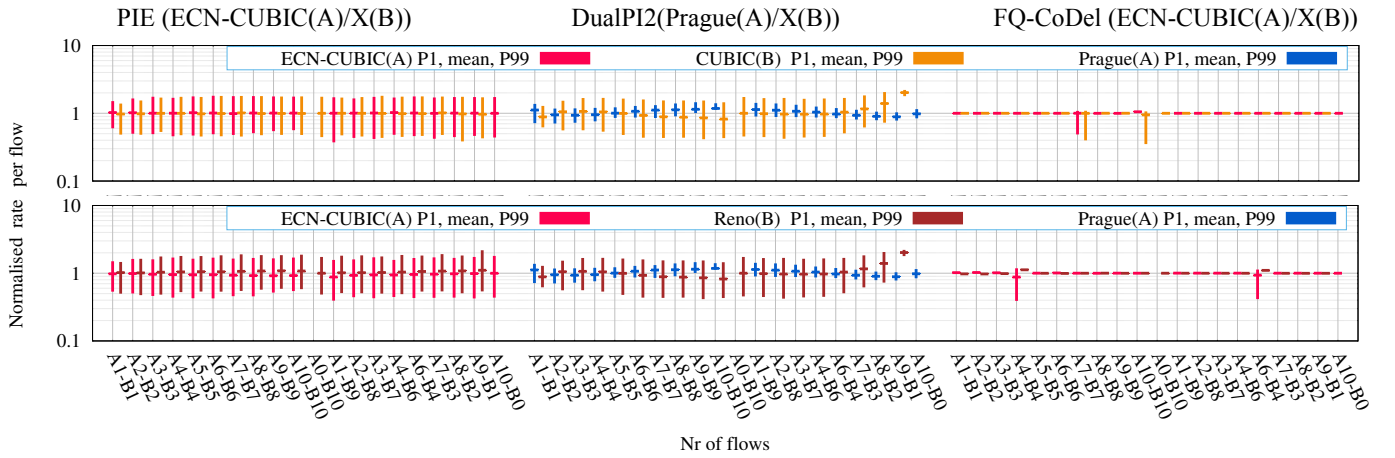
Fig. 7: Set-up as Figure 6 but with different numbers of flows; Link: 40 Mb/s; Base RTT: 10 ms. Queue delay and utilization (not shown) are similar to Figure 6
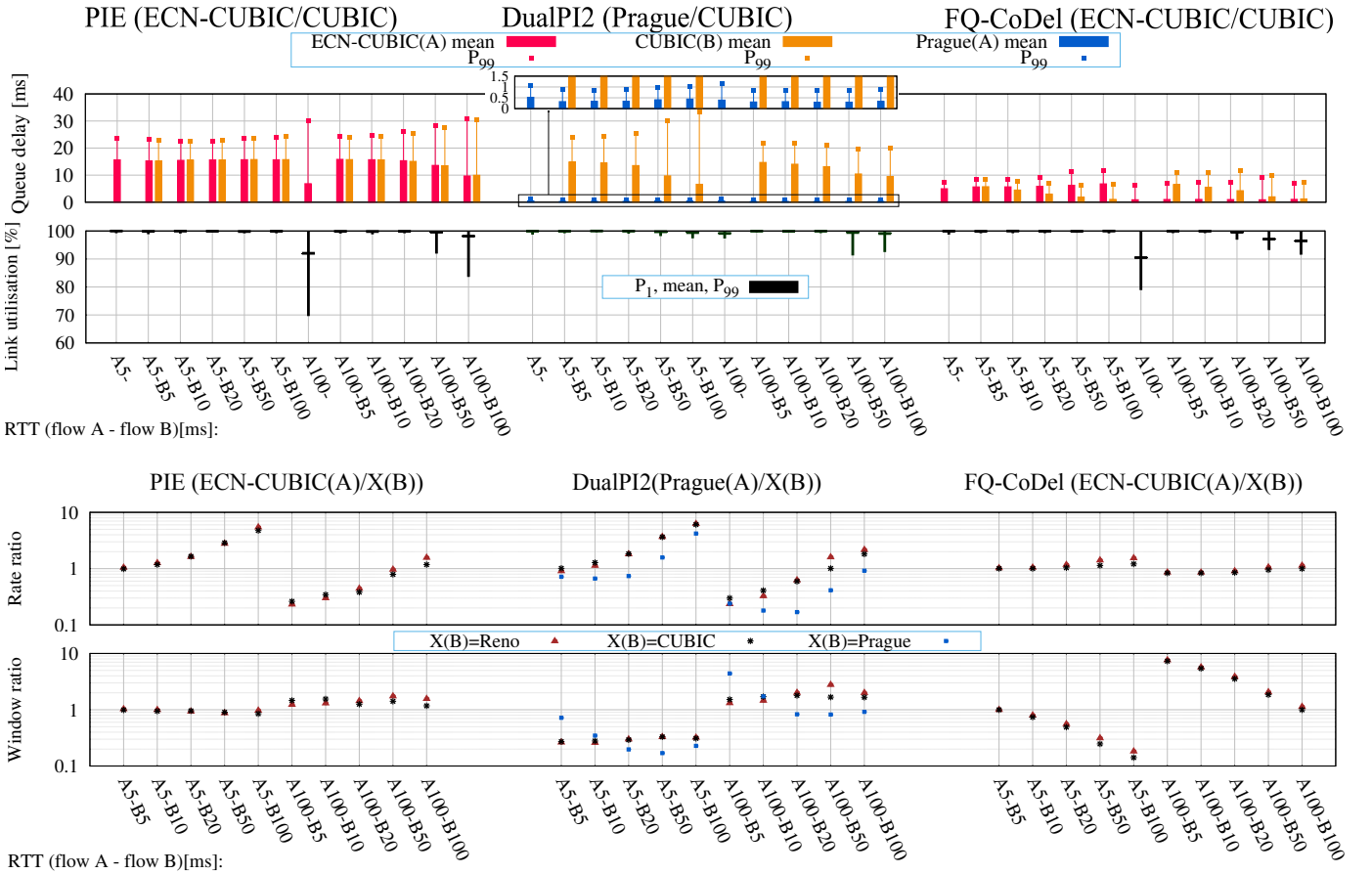

Fig. 8: Set-up as Figure 6 with 1 flow for each CC but mixed RTTs; Link: 40 Mb/s.

DualPI2 and the A100 cases over PIE. Both are due to CReno's larger sawteeth at high RTT, but only when there is no short RTT Classic flow to fill the valleys.

The second row shows that in steady state DualPI2 utilizes the link better than PIE or FQ-CoDel, with mean **utilization** close to 100% in all cases — even with a lone high RTT flow (A100-), where PIE and FQ-CoDel show significant underutilization. The P99 of DualPI2 falls to 90% in the A100-B50 case, mainly due to rate variations of the Classic flow;

and the A100-B100 case is similar. However neither of the Classic AQMs achieve better results in those cases.

The third row shows that the scheduler of FQ-CoDel enforces near-perfect **rate balance**, as expected. Nonetheless, DualPI2 achieves similar rate balance to PIE, confirming that the effect of Prague's RTT-independence algorithm is equivalent to that of a shared queue AQM on Classic flows. It is well-known that competing Classic TCP flows equalize their congestion windows, so that their rates become inversely pro-

portional to their respective RTTs, evidenced by the **window ratio** of PIE staying around 1 (fourth row). For DualPI2, we see that when Prague has a lower RTT it reduces its window to keep the rate ratio within bounds.

Additional rate ratio results are included for two competing Prague flows over different RTTs. Prague behaves as if its RTT is no lower than 25 ms, so the rate ratio would be expected to flatten for RTTs below that. However, as RTT reduces further the rate ratio increases slightly. Further investigation pointed to an interaction between Prague's segmentation offloading and use of sojourn time in the AQM (see [11] for details).

We conclude so far that the very low queuing delay of Prague over DualPI2 is maintained with mixed RTTs, still without compromising utilization or rate balance.

F. Dynamic Load

a) Experimental setup: To evaluate dynamic behaviour, we built on the basic steady-state experiments (§ IV-C), but added emulated web traffic to each of the long-running flows between the two client-server pairs. Two web load profiles (high and low) were run, as defined in Appendix C.
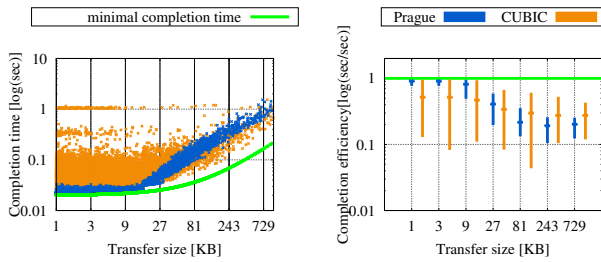

Fig. 9: Efficiency representation (right) of example flow completion time scatter plot (left). High web load and 1 steady-state flow for each CC over DualPI2 AQM.

As before, we show queuing delay and link utilization metrics, but we also show flow completion time (FCT) which is a more relevant metric than rate for short flows. To better quantify the FCT distribution, we used the Completion Efficiency representation on the right of Figure 9. Completion efficiency is defined as a theoretically achievable FCT (the green line at 1) divided by the actual FCT. The theoretically achievable FCT takes the RTT into account for the handshake but then downloads at full link speed. As one example, the left-hand side of Figure 9 shows the high web load DualPI2 AQM test case on a 40 Mb/s link with 10 ms base RTT. It is a log-log scatter plot of the FCT to item size relation, and the green line along the bottom is the theoretically achievable FCT. We then binned the samples in log scale bins (base 3) and plotted the average, 1st and 99th percentiles.

Figure 10 and the top row of Figure 11 show the most challenging high web load scenario over a 120 Mb/s link. They are representative of the full set of experiments performed over all 5 link rates, which are summarized in the rest of Figure 11.

b) Results: The complementary CDFs of **queue delay** in Figure 10 demonstrate that, even under heavy web load, L4S queue delay is below 1 ms at P99 and an order of magnitude lower than PIE or FQ-CoDel at all the higher percentiles.

The **completion efficiency** results in Figure 11 show an immediate latency benefit for short flows (<50KB), while many bigger flows (>50KB) suffer from Prague taking longer to get up to speed, in particular at RTTs above 50 ms (see § VI-B later).

The mean and P1 of short CUBIC flows in both DualPI2 and PIE have poor completion efficiency. The two horizontal stripes at 1 s and 0.3 s on the left of Figure 9 largely explain this. They match the retransmission timeout of a TCP-SYN and a tail data packet respectively, which impact some flows randomly due to the prevailing loss environment. FQ-CoDel avoids this problem by prioritizing lone packets and Prague avoids it by supporting ECN-capable SYNs. CUBIC-ECN is not immune from SYN loss because the Classic ECN spec [54] does not allow TCP SYNs to be ECN-capable.

With dynamic load, Prague over DualPI2 preserves very low **queue delay** by sacrificing **link utilization**. FQ-CoDel's link utilization profile is similarly reduced, but without the low P99 queue delay, particularly for lower link rates (<120 Mb/s), which is due to its short flow prioritization and RR scheduling. Low L4S delay is not at the expense of Classic delay, evidenced by DualPI2's Classic queue delay (mean and P99) often being significantly lower than FQ-CoDel & PIE.

We conclude that in general, high levels of dynamic traffic do not noticeably compromise either L4S or Classic queue delay with DualPI2. Improvement to Prague's mechanisms for getting up to speed are discussed in § VI-B and its references, but evaluating them is outside our AQM-only scope.

G. Unresponsive Flows and Overload

a) Experimental setup: To trigger overload, we added an unresponsive UDP flow to 5 long-running flows of each CC type (Prague & CUBIC) with 10 ms base RTT over DualPI2 on a 100 Mb/s bottleneck link. We tested 5 different nominal UDP sending rates from 50% to 200% of link capacity, as shown on the X-axis of Figure 12, which also shows that we ran 2 sets of tests with the UDP flow's class marked as either ECT(1) (L4S) or not-ECT (Classic).

b) Results: The top row of Figure 12 shows that, if a UDP flow targets the L4S queue rather than Classic, it hardly increases its **rate share**. Recall from § III-D that this satisfies the design goal of not amplifying any existing attack, without resorting to flow rate policing, which is nonetheless still available as an orthogonal policy choice.

All three rows show that, with UDP rate lower than the bottleneck capacity (<100Mb/s), the choice of UDP class has no significant effect on any of the metrics of the responsive flows: **queue delay**, **marking probability** or **rate**.

If a UDP flow targets the L4S queue at ≥100 Mb/s, it can increase L4S **queue delay** to the Classic target (15 ms). Making the two queues behave as one on overload was a deliberate design choice (see § III-D) to ensure that, if L4S traffic overloads the link, it cannot gain any latency benefit. For alternative choices see [28].

We conclude that under overload the DualPI2 uses drop to shed load, effectively becoming a single queue Classic AQM, which excludes excessive unresponsive traffic from the advantages of both ECN and low delay.
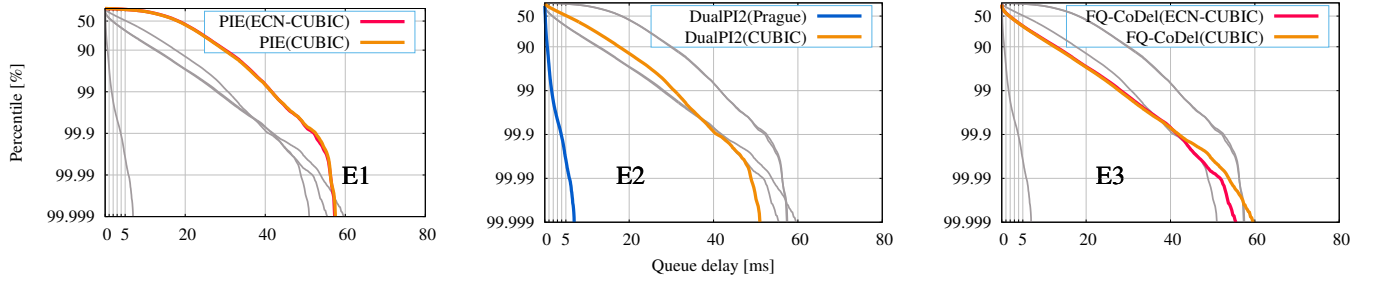
Fig. 10: Log-scale complementary CDF comparing the DualPI2 AQM with PIE & FQ-CoDel, each under the same challenging traffic scenario as Figure 11; Link: 120 Mb/s, RTT: 10 ms. Each experiment run (E1,E2,E3) used 1 of the 3 AQMs and 2 CC types. The results for the AQM under test are shown in colour, with the other 2 AQMs in grey for comparison.
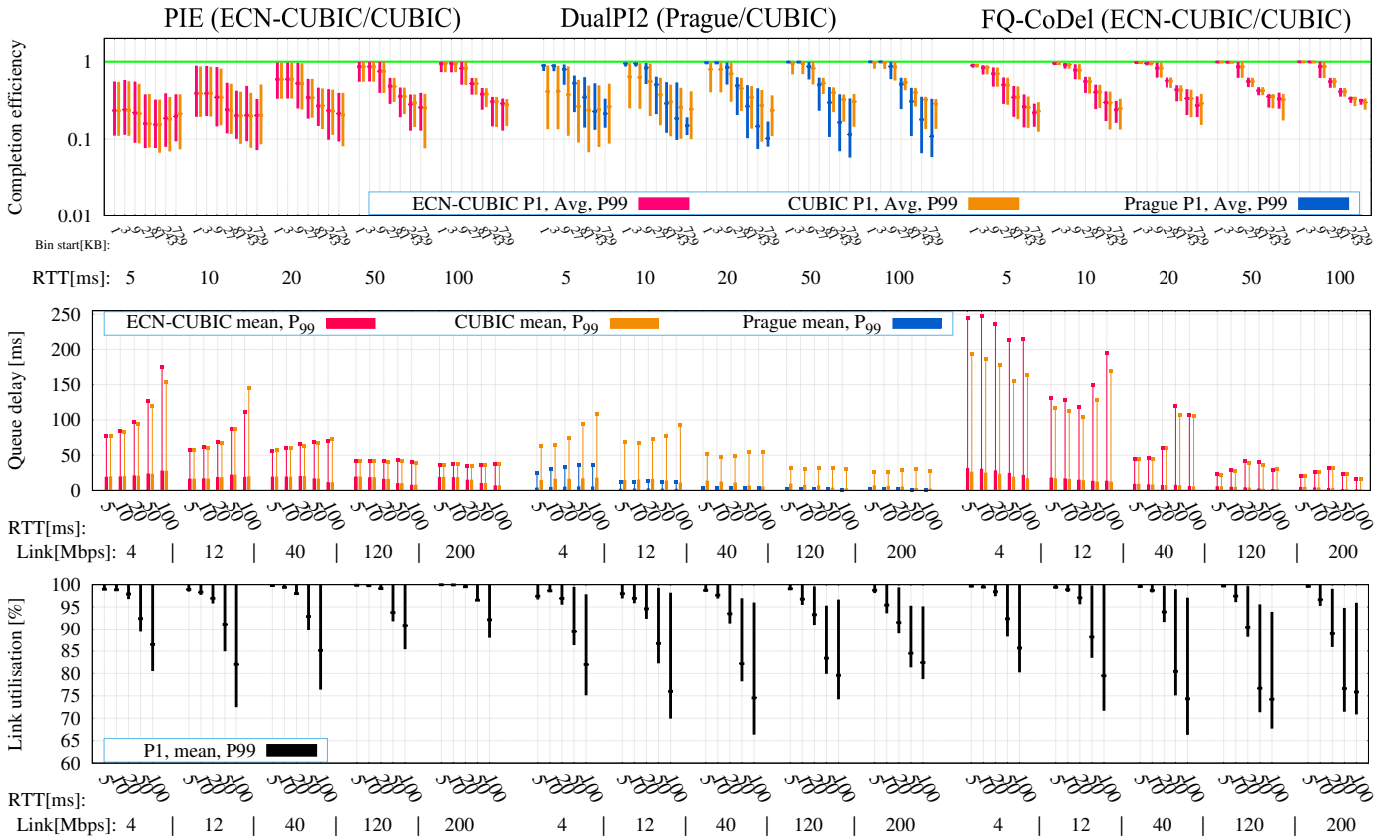

Fig. 11: Heavy dynamic workload. Top plot shows completion efficiency for a traffic mix of 1 long flow and 300 short requests per second for each CC; Link=120 Mb/s, Base RTTs=10 ms. Middle and bottom plots show queue delay and utilisation for the same (proportionately scaled) traffic mix but over the wider set of link/RTT combinations.

V. DEPLOYMENT CONSIDERATIONS

A. Deployment Scenarios

Given access networks are invariably designed to bottleneck in one known location, the DualQ 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 would address the upstream. Nonetheless, the DualQ is simple enough to be deployed at any potential bottleneck—in data centres, peerings, access links or within end systems.

Also, the DualQ makes uncoordinated deployment of DC-TCP practical in data centres, e.g. across multi-tenant data centres or across community of interest networks connecting private data centres. This extends the applicability of DCTCP beyond private networks where a centralized admin can coordinate deployment on a flag-day.

B. Standardization Requirements

The IETF has taken on L4S standardization work [17]. It considered the pros and cons of various candidate identifiers for senders to indicate L4S support and found that none were without problems, but proposed ECT(1) as the least worst [27]. As a consequence, the IETF has updated the ECN standard at the IP layer (v4 and v6) to make the ECT(1) codepoint available for experimentation [8].

The main issue is that there is only one spare codepoint. So, if senders use it to distinguish L and C packets, there is only

Rate share [%]

	ECT(1) UDP	Not-ECT UDP	CUBIC	Prague
	25.6	16.4	3.20	0.49	0.08		22.2	13.3	6.87	0.83	0.06
	22.8	11.5	1.58	0.73	0.05		26.6	15.6	3.00	0.60	0.07
	51.6	71.1	95.2	98.8	99.9		51.2	71.1	90.1	98.6	99.9

mean, P_{25}, P_{99}:

Marks Prague	Drops CUBIC	Drops ECT(1) UDP
Drops Prague	Marks ECT(1) UDP	Drops Not-ECT UDP

Drop/Mark [%] (100 / 10 / 1 / 0.1)

Queue delay [ms]

Prague mean, P_{99}	CUBIC mean, P_{99}

(30 / 25 / 20 / 15 / 10 / 5 / 0)

UDP sending rate [Mbps]: 50 70 100 140 200   50 70 100 140 200
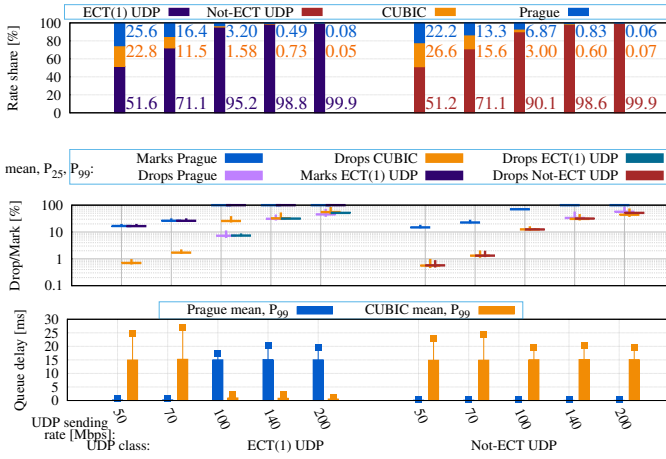
UDP class: ECT(1) UDP   Not-ECT UDP

Fig. 12: Overload experiments on a 100 Mb/s link: 1 UDP flow + 10 long running TCP flows (5 flows for each CC). The plots compare a UDP flow classifying itself as ECT(1) and as not-ECT (L4S and Classic respectively).

one Congestion Experienced (CE) codepoint for the network to mark both L & C packets (the 'CE ambiguity problem'). CE is not ambiguous to a sending host, because it knows whether it was sending L or C packets. However, it is ambiguous within the network, which has the following implications.

In the (unusual) case of multiple ECN bottlenecks along one path, a CE mark from the first AQM is ambiguous to the second—it will not know whether it was originally L or C. Nonetheless, it is benign for an L4S AQM to classify any arriving CE packets into its L queue, because occasional early packets do not trigger spurious retransmissions unless the five unlikely conditions listed in Appendix B of [27] are all true.

Also, the 'Classic' ECN standard [54] requires the network to treat ECT(1) and ECT(0) equally. So an L4S host sending an ECT(1) packet does not explicitly know whether any resulting CE-marking is from an L4S AQM or from a Classic AQM that doesn't understand L4S. If the L4S sender wrongly applies a small L4S congestion response when the mark is actually from a Classic AQM in a shared queue, and it is shared by competing Classic flow(s), the L4S flow will start to take an 'unfair' share of the capacity: the 'coexistence problem'.

This important subject requires a whole paper, and it is beyond the DualQ scope of the present paper anyway. In brief, it is widely believed that most or perhaps all deployments of AQMs that support Classic ECN also use per-flow queuing (FQ). If so, there is only a coexistence problem in the unintended cases where more than one flow shares a per-flow queue, e.g. hash collisions or L3 VPNs. For such cases, and for any cases of single-queue Classic ECN AQMs, an approach has been proposed [13] where the L4S sender detects which type of AQM is marking the packets. The initial detection algorithm resulted in too many false positives, so the authors suggested various avenues for improvement. Another approach was proposed where senders proceed with L4S deployment while monitoring for these potential problems, then targeted out-of-band testing could check for false positives [27].

Having chosen the new ECT(1) identifier, the IETF has also had to define its semantics. The square relationship between an L4S mark and a drop in this paper (Eqn. (3)) has been proposed for experimental standardization [27].

The IETF has not specified a particular DualQ coupled AQM, but it has specified the necessary relationship between AQMs in a DualQ structure in order to couple them [28], so that multiple implementations can be built, tested and compared, possibly using different base AQMs internally. It has been proposed to recommend rather than standardize a value for the coupling factor, k, given differences would not prevent interoperability.

The coupled DualQ AQM structure has already been adopted by the cable industry as part of the mandatory low latency support added to DOCSIS 3.1 [30].

VI. FURTHER WORK

A. Validation

The results in this paper are largely similar to those published in 2015 & 2019 as technical reports [25], [24]. Subsequently, the results regarding capacity sharing and latency isolation were validated independently [9] once it was clarified that L4S sources need to employ pacing.

Heist *et al* [37] identifies a number of scenarios where L4S performance appears to suffer and raises a question about the impact that bursty traffic can have on that performance. This question is relevant to the AQM in this paper and it is believed that performance in the presence of bursty traffic can be improved by basing marking on the time it would take to drain the backlog at dequeue [11], rather than at enqueue, which is what sojourn time measures. Work is ongoing to identify which of the other issues are specific to L4S. Of those that are, most are related to congestion control (see § VI-B below) and some have already been fixed after having been reproduced using ns-3 models of L4S components [38].

B. Congestion Control Roadmap

TCP Prague [29] as used in this paper is a set of fixes to issues in DCTCP so that it can act as a reference scalable congestion control for proving and testing the L4S architecture. It is not necessarily intended to be maintained as a production congestion control that is optimized for the range of environments found on the Internet, although the open source code is available for others to incorporate in parts or as a whole. Nonetheless, it is sufficient to exercise the parameter space of our experiments in order to evaluate the network mechanism of L4S, without which end-system performance improvements would be moot. To further clarify the status of the Prague Congestion Control, pending improvements are listed below in priority order. They are a summary of the L4S transport layer behaviours identified by the IETF [27, Appx. A], which are in turn adapted from the "Prague L4S requirements", named after the meeting in Prague of a large group of DCTCP developers that informally agreed them [10].

1) Less drastic exit from slow-start, similar goal to Flow-Aware (FA-DCTCP) [43] or Paced Chirping [50];

2) Faster-than-additive increase, e.g. Adaptive Acceleration (A^2DTCP) [60] or Paced Chirping [50];

3) Yield fast to excessive delay increase, e.g. due to capacity reduction;

4) Handle a window of less than 2, rather than grow the queue if base RTT is low [16, § 3.1.6];

5) Fall back to a Classic congestion response if a classic ECN bottleneck is detected (Prague contains an optional initial algorithm that was disabled for the present experiments [13]);

6) Improve RTT-independence of rate (see § III-B & [29]).

VII. Related Work

In 2002, Gibbens and Kelly [34] 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 [46, §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 [32].

Wu et al. [58] investigates a way to incrementally deploy DCTCP within data centres, marking ECN when the temporal queue exceeds a shallow threshold but using standard ECN [54] on end-systems. Also in a data centre context, Irteza *et al.* [42] investigates potential pre-existing solutions to the congestion control coexistence problem and finds them unsatisfactory. Kuhlewind *et al.* [45] showed that DCTCP and Reno could co-exist in the same queue configured with a form of WRED [22] classifying on ECN rather than Diffserv, but only over a limited set of conditions. Judd [44] 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) [59] is an AQM for DC switches building on CoDel. Unlike the DualQ, FaLL inspects the transport layer of sample packets to focus more marking onto faster flows while keeping the queue short.

VIII. 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 how the network 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 dispense with 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 inflate (worsening impairments) as BDP scales. This transition to a scalable regime is as important as the low delay that L4S offers today.

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 (P99 of 2 ms); 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 not to 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 very 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 inherently auto-tuning AQMs to link rate, but also shifting RTT-dependent smoothing to end-systems, which know their own RTT.

References

[1] Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and Steen, H. DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM. In *Proc. Netdev 0x13* (Mar. 2019).

[2] Alizadeh, M., et al. Less Is More: Trading a Little Bandwidth for Ultra-Low Latency in the Data Center. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)* (Apr. 2012).

[3] 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.

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

[5] Allman, M., Paxson, V., and Blanton, E. TCP Congestion Control. RFC 5681, RFC Editor, Sept. 2009.

[6] 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.

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

[8] Black, D. Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation. RFC 8311, RFC Editor, Jan. 2018.

[9] Boru Oljira, D., Grinnemo, K.-J., Brunstrom, A., and Taheri, J. Validating the Sharing Behavior and Latency Characteristics of the L4S Architecture. *ACM SIGCOMM Computer Communication Review 50*, 2 (May 2020), 37–44.

[10] 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.

[11] BRISCOE, B. Rapid Signalling of Queue Dynamics. Technical Report TR-BB-2017-001; arXiv:1904.07044 [cs.NI], bobbriscoe.net, Apr. 2019.

[12] BRISCOE, B. PI^2 Parameters. Technical Report TR-BB-2021-001; arXiv:2107.01003 [cs.NI], bobbriscoe.net, Oct. 2021.

[13] BRISCOE, B., AND AHMED, A. S. TCP Prague Fall-back on Detection of a Classic ECN AQM. Technical Report TR-BB-2019-002; arXiv:1911.00710 [cs.NI], bobbriscoe.net, Feb. 2021.

[14] 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.

[15] BRISCOE, B., AND DE SCHEPPER, K. Resolving Tensions between Congestion Control Scaling Requirements. Technical Report TR-CS-2016-001; arXiv:1904.07605, Simula, July 2017.

[16] BRISCOE, B., DE SCHEPPER, K., ALBISSER, O., MISUND, J., TILMANS, O., KÜHLEWIND, M., AND AHMED, A. S. Implementing the 'TCP Prague' Requirements for L4S. In *Proc. Netdev 0x13* (Mar. 2019).

[17] BRISCOE, B., DE SCHEPPER, K., BAGNULO, M., AND WHITE, G. Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture. Internet Draft draft-ietf-tsvwg-l4s-arch-19, IETF, July 2022. (Work in Progress).

[18] BRISCOE, B., WHITE, G., GOEL, V., AND SCHEPPER., K. D. A Single Common Metric to Characterize Varying Packet Delay. In *Proc. IAB Workshop on Measuring Network Quality for End-Users* (Sept. 2021).

[19] CARDWELL, N., CHENG, Y., YEGANEH, S. H., SWETT, I., AND JACOBSON, V. BBR Congestion Control. Internet Draft draft-cardwell-iccrg-bbr-congestion-control-02, IRTF, Mar. 2022. (Work in Progress).

[20] CARMACK, J. Latency Mitigation Strategies. Blog post: https://www.twentymilliseconds.com/post/latency-mitigation-strategies/, 2013.

[21] CHEN, W., CHENG, P., REN, F., SHU, R., AND LIN, C. Ease the Queue Oscillation: Analysis and Enhancement of DCTCP. In *IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)* (July 2013), pp. 450–459.

[22] CLARK, D. D., AND FANG, W. Explicit allocation of best-effort packet delivery service. *IEEE/ACM ToN 6*, 4 (Aug. 1998), 362–373.

[23] DAVIE, B., ET AL. An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246, IETF, Mar. 2002.

[24] DE SCHEPPER, K., ALBISSER, O., TSANG, I.-J., TILMANS, O., AND BRISCOE, B. 'Data Centre to the Home': Deployable Ultra-Low Queuing Delay for All. Draft, July 2019. https://bobbriscoe.net/pubs.html#DCttH_TR.

[25] DE SCHEPPER, K., BONDARENKO, O., TSANG, I.-J., AND BRISCOE, B. 'Data Center to the Home': Ultra-Low Latency for All. Technical report, RITE Project, June 2015. https://riteproject.eu/publications/.

[26] 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.

[27] DE SCHEPPER, K., AND BRISCOE (ED.), B. Explicit Congestion Notification (ECN) Protocol for Very Low Queuing Delay (L4S). Internet Draft draft-ietf-tsvwg-ecn-l4s-id-28, IETF, Aug. 2022. (Work in Progress).

[28] DE SCHEPPER, K., BRISCOE (ED.), B., AND WHITE, G. DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput (L4S). Internet Draft draft-ietf-tsvwg-aqm-dualq-coupled-24, IETF, July 2022. (Work in Progress).

[29] DE SCHEPPER, K., TILMANS, O., AND BRISCOE, B. Prague Congestion Control. Internet Draft draft-briscoe-iccrg-prague-congestion-control-01, IRTF, July 2022. Work in Progress.

[30] Data-Over-Cable Service Interface Specifications DOCSIS® 3.1; MAC and Upper Layer Protocols Interface Specification. Specification CM-SP-MULPIv3.1-I17-190121, CableLabs, Jan. 2019.

[31] DUKKIPATI, N., AND MCKEOWN, N. Why Flow-Completion Time is the Right Metric for Congestion Control. *ACM SIGCOMM Computer Communication Review 36*, 1 (Jan. 2006), 59–62.

[32] FLOYD, S. TCP and Explicit Congestion Notification. *ACM SIGCOMM Computer Communication Review 24*, 5 (Oct. 1994), 10–23.

[33] FLOYD, S. HighSpeed TCP for Large Congestion Windows. RFC 3649, RFC Editor, Dec. 2003.

[34] GIBBENS, R. J., AND KELLY, F. P. On Packet Marking at Priority Queues. *IEEE Trans. Automatic Control 47*, 6 (June 2002), 1016–1020.

[35] HA, S., RHEE, I., AND XU, L. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Operating Systems Rvw 42*, 5 (July 2008), 64–74.

[36] HAN, L., APPLEBY, S., AND SMITH, K. Problem Statement: Transport Support for Augmented and Virtual Reality Applications. Internet Draft draft-han-iccrg-arvr-transport-problem-00, IRTF, Mar. 2017.

[37] HEIST, P., AND MORTON, J. L4S Tests. Online https://github.com/heistp/l4s-tests/#readme, 2020-21.

[38] HENDERSON, T., TILMANS, O., AND WHITE, G. Testbed and simulation results for tsvwg scenarios. Online https://l4s.cablelabs.com/, 2019.

[39] HOEILAND-JOERGENSEN, T., MCKENNEY, P., TÄHT, D., GETTYS, J., AND DUMAZET, E. The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm. RFC 8290, RFC Editor, Jan. 2018.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] JUDD, G. Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter. In *Symp. Networked Systems Design and Implementation (NSDI'15)* (Oakland, CA, May 2015), USENIX Assoc., pp. 145–157.

[45] 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.

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

[47] 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.

[48] 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).

[49] 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.

[50] MISUND, J., AND BRISCOE, B. Paced Chirping - Rethinking TCP start-up. In *Proc. Netdev 0x13* (Mar. 2019).

[51] 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.

[52] PARK, J., SESHADRINATHAN, K., LEE, S., AND BOVIK, A. C. Video Quality Pooling Adaptive to Perceptual Distortion Severity. *IEEE Transactions on Image Processing 22* (2013), 610–620.

[53] RAJIULLAH, M. Towards a Low Latency Internet: Understanding and Solutions. Master's thesis, Karlstad Uni, Aug. 2015.

[54] RAMAKRISHNAN, K. K., FLOYD, S., AND BLACK, D. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, RFC Editor, Sept. 2001.

[55] SALIM, J. H., AND AHMED, U. Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks. RFC 2884, RFC Editor, July 2000.

[56] WHITE, G., SUNDERESAN, K., AND BRISCOE, B. Low Latency DOCSIS: Technology Overview. White paper, CableLabs, Feb. 2019.

[57] WILLARS, P., ET AL. Enabling time-critical applications over 5G with rate adaptation. White Paper BNEW-21:025455 Uen, Ericsson – Deutsche Telekom, May 2021.

[58] 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.

[59] 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.

[60] 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.

APPENDIX

A. DualQ Coupling Factor

Here we derive the constant coupling factor k defined as follows in Equation 3 of § III-B using the terminology defined there:

$$k = \sqrt{\frac{8}{3}} \left. \frac{R_C}{f(R_L)} \right|_{R_b^*} \tag{4}$$

R_C is the Classic RTT averaged over the Classic sawteeth. So it can be related geometrically to the RTT at the AQM's operating point:

$$R_C \approx \frac{(1 + \beta_C)}{2}(R_b + \text{target}), \tag{5}$$

where β_C is the multiplicative decrease factor of the Classic flow (0.5 for Reno or 0.7 for CReno). $(R_b + \text{target})$ is roughly where the peaks of the Classic RTT sawteeth settle, not their average [12, § 3.3].[18] So it is factored down to the average by $(1 + \beta_C)/2$.

To calculate k it is only necessary to know the value of $f(R_L)$ at the reference RTT, R_b^*, chosen by the operator. It is likely that this will be an RTT where the implementer has ensured that $f(R_L) \approx R_L$, and one can assume that $R_L \approx R_b$ because the L queue is negligible. For instance, for the current Linux implementation of Prague $f(R_L) = \max(R_L, 25\,\text{ms})$. So, if the operator chooses $R_b^* = 25\,\text{ms}$, which is a typical base RTT for Internet traffic [12], it is indeed true there that $f(R_L) \approx R_b$. In such cases:

$$k \approx \sqrt{\frac{8}{3}} \frac{(1 + \beta_C)}{2} \left(1 + \frac{\text{target}}{R_b^*}\right), \tag{6}$$

Then, taking the recommended value of $\text{target} = 15\,\text{ms}$, $k = 1.96$ for Reno[19] or 2.22 for CReno.

B. Evaluation metrics

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. The measurements were started after long-runnning TCP flows reached their steady states, skipping the slow start. To skip the slow start, we added a waiting time of T_w seconds before starting the measurements, which was calculated as $T_w = 5 + \text{rate} * \text{rtt}/100$. The slow start was not skipped for short dynamic flows.

Queuing delay was measured inside the qdiscs per packet, at dequeue time. All samples were further processed to derive mean and 99th percentile.

---

[18]This is because, at the scale of today's Internet, the duration of each sawtooth of a Classic congestion control (the recovery time) is usually significantly longer than the time it takes for a Classic AQM like PI^2 to converge to a stable congestion level.

[19]Equation 5 is only valid for $R_b \leq \text{target} * \beta_C/(1 - \beta_C)$. For base RTTs above this, the window sawtooth under-utilizes the buffer. Therefore the values plugged in are outside the range of validity for Reno, but inside for CReno. However, the approximation is still close enough in practice, particularly because the tips of the sawteeth actually settle slightly above the AQM target, which slightly expands the range of validity.

Drop probability was also measured inside the qdiscs by keeping the counter of dropped packets, while mark probability was obtained by inspecting the packet headers and checking which packets were marked. Both mark and drop probability are represented as a percentage of all packets that passed the network interface where the qdisc was installed. All collected 1-second samples per experiment were processed to obtain mean, 25th and 99th percentiles.

To evaluate rate and window behaviour, we measured throughput for each long-running flow by capturing all traffic at the AQM node. Since some experiments had an unequal number of flows for each CC, we sampled average rate per flow and normalized it by dividing by the fair rate per flow. Fair rate was calculated by dividing the total link capacity by total number of competing flows. The same approach was applied to window measurements. For convenience, Window size was approximated on the AQM node as $W = r * (qd_{avg} + R)$, where r is measured rate in bytes, qd_{avg} is average queue delay and R is base RTT (both delay and RTT were converted to fractions of a second). All rate and window measurements were sampled per second, and all samples were further processed to derive mean, 1st and 99th percentiles. Rate and window balance ratio was calculated by dividing average rate/window per flow of the ECN-capable CC by average rate/window per flow of the Classic CC for the entire experiment.

Utilization was measured by comparing all traffic captured at the interface to the total link capacity in 1-second intervals; as with other measurements, we derived mean, 1st and 99th percentiles by processing all obtained samples.

For short dynamic flows, the client logged the completion time and downloaded size. Timing was started just before opening the TCP socket, and stopped after the connection close by the server was detected.

C. Web Traffic Model

To model the web traffic in the dynamic load experiments (§ IV-F), an exponential arrival process was used with an average of 1 (low load) or 10 (high load) requested items per second for the 4 Mb/s link capacity, scaled for the higher link speeds up to 50 (low) or 500 (high) requests for the 200 Mb/s 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.