Preprint

Under Submission

Until published, please cite as:

De Schepper, K.; Bondarenko, O.; Tsang, I.-J. and Briscoe, B. "'Data Centre to the Home': Ultra-Low Latency for All" (Under submission, 2015)

Document history

Version	Date	Author	Details of change
Issue 01	12 Jun 2015	Koen De Schepper	Issue for submission
Issue 01A	06 Jul 2015	Bob Briscoe	Corrections and Enhancements
Issue 01B	07 Jul 2015	Bob Briscoe	Cover page added
Issue 01C	28 Jul 2015	Bob Briscoe	Altered L (Figure 3) to match updated ref [2].

‘Data Centre to the Home’: Ultra-Low Latency for All

Koen De Schepper[†] Olga Bondarenko[*] [‡] Ing-Jyh Tsang[†] Bob Briscoe[§]
[†]Alcatel-Lucent, Belgium [‡]Simula Research Laboratory, Norway [§]BT, UK
[†]{koen.de_schepper|ing-jyh.tsang}@alcatel-lucent.com
[‡]olgabo@simula.no [§]research@bobbriscoe.net

ABSTRACT

Data Centre TCP (DCTCP) was designed to provide predictably low queuing latency, near-zero loss, and throughput scalability using explicit congestion notification (ECN) and an extremely simple marking behaviour on switches. However, DCTCP does not co-exist with existing TCP traffic—throughput starves. So, until now, DCTCP could only be deployed where a clean-slate environment could be arranged, such as in private data centres. This paper proposes ‘Coupled Active Queue Management (AQM)’ to allow scalable congestion controls like DCTCP to safely co-exist with classic Internet traffic. In extensive tests within the edge gateway of a realistic broadband access testbed, the Coupled AQM ensures that a flow runs at about the same rate whether it uses DCTCP or TCP Reno/Cubic, but without inspecting transport layer flow identifiers. DCTCP achieves sub-millisecond average queuing delay and zero congestion loss under a wide range of mixes of DCTCP and ‘classic’ broadband Internet traffic, without compromising the performance of the classic traffic. The solution also reduces network complexity and eliminates network configuration.

1. INTRODUCTION

Latency is becoming the critical performance factor for many (most?) applications on the public Internet, e.g. Web, voice, conversational video, gaming and finance apps. In the developed world, further increases in access network bit-rate offer diminishing returns, whereas latency is still a multi-facetted problem. In the last decade or so, much has been done to reduce propagation time by placing caches or servers closer to users. However, queuing remains a major component of latency.

The Diffserv architecture provides Expedited Forwarding, so that low latency traffic can jump the queue of other traffic. However, on access links dedicated to individual sites (homes, small enterprises or mobile devices), often *all* traffic at any one time will be latency-sensitive. Then Diffserv is of little use. Instead, we need to remove the causes of any unnecessary delay.

The bufferbloat project has shown that excessively-large buffering (‘bufferbloat’) has been introducing significantly more delay than the underlying propagation time. These delays appear only intermittently—only when a capacity-seeking (e.g. TCP) flow is long enough for the queue to fill the buffer, making every packet in other flows sharing the buffer sit through the queue.

Active queue management (AQM) was originally developed to solve this problem (and others). Unlike Diffserv, AQM controls latency for *all* traffic in a class. In general, AQMs introduce an increasing level of discard from the buffer the longer the queue persists above a shallow threshold. This gives sufficient signals to capacity-seeking (aka. greedy) flows to keep the buffer empty for its intended purpose: absorbing bursts. However, RED and other algorithms from the 1990s were sensitive to their configuration and hard to set correctly. So, AQM was not widely deployed. More recent state-of-the-art AQMs, e.g. fq_CoDel [8], PIE [13], Adaptive RED, define the threshold in time not bytes, so it is invariant for different link rates.

It seems that further changes to the network alone will now yield diminishing returns. Data Centre TCP (DCTCP [1]) teaches us that a small but radical change to TCP is needed to cut two major outstanding causes of queuing delay variability:

1. the ‘sawtooth’ varying rate of TCP itself;
2. the smoothing delay deliberately introduced into AQMs to permit bursts without triggering losses.

The former causes a flow’s round trip time (RTT) to vary from about 1 to 2 times the base RTT between the machines in question. The latter delays the system’s response to change by a worst-case (transcontinental) RTT, which could be hundreds of times the actual RTT of typical traffic from localised CDNs.

Latency is not our only concern.

3. It was known when TCP was first developed that it would not scale to high bandwidth-delay products.

Given regular broadband bit-rates over WAN distances are already beyond the scaling range of ’classic‘ TCP Reno, ‘less unscalable’ Cubic [7] and Compound [15] variants of TCP have been successfully deployed. How-

[*]The first two authors contributed equally

ever, these are now approaching their scaling limits. Unfortunately, fully scalable TCPs cause 'classic' TCP to starve itself, which is why they have been confined to private data centres or research testbeds (until now).

Our contribution is a 'Coupled AQM' that solves the problem of coexistence between DCTCP and classic flows, without having to inspect flow identifiers. It needs fewer operations per packet than RED uses. Also, no network configuration is needed for a wide range of scenarios.

It uses two queues for two services: "Classic" and "Low-Latency, Low-Loss and Scalable" (L4S), denoted resp. by subscripts 'C' and 'L'. The 'Classic' service is intended for all the behaviours that currently co-exist with TCP Reno (TCP Cubic, Compound, SCTP, etc). The 'L4S' service is intended for DCTCP traffic but it is also more general—it will allow a set of congestion controls similar to DCTCP (e.g. Relentless) to evolve.

The AQM couples marking and/or dropping across the two queues such that a flow will get roughly the same throughput whichever it uses. This enables scalable congestion controls like DCTCP to give stunningly low and predictably low latency, without compromising the performance of competing 'classic' Internet traffic.

In our tests, we use DCTCP unmodified, despite its known deficiencies (listed as further work in Section 10). Our focus is on getting the network service in place. Then, without any management intervention, applications can exploit it by migrating to scalable controls like DCTCP, which can then evolve *while* their benefits are being enjoyed by everyone on the Internet.

We also conducted subjective testing with a demanding panoramic interactive video application run over a stack with DCTCP enabled and deployed on the testbed. Each user could pan or zoom their own HD sub-window of a larger video scene from a football match (Figure 1). Even though the user was also downloading large amounts of L4S and Classic data, latency was so low that the picture appeared to stick to their finger on the touchpad (all the L4S data achieved the same ultra-low latency). With an alternative AQM, the video noticeably lagged behind the finger gestures.

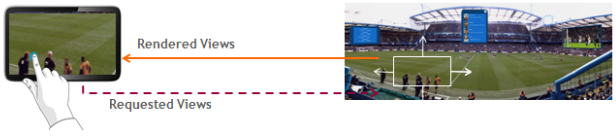

Figure 1: Panoramic interactive video application

Document Outline. Section 2 gives the intuition for our design choices, deferring the mathematical rationale to Section 3. We then present our two main contributions: the 'Coupled AQM' that addresses throughput equivalence (Section 4) then a Dual Queue structure that provides latency separation (Section 5).

To evaluate the new AQM's performance, we have built it into the downstream of a realistic end-to-end broadband testbed (Section 6). Section 7 gives a condensed report of thousands of experiments to quantify its performance against RED, PIE and fq_CoDel.

The paper ends with discussion of standardisation aspects (Section 8) before the usual tailpieces.

2. RATIONALE

2.1 Intuition: Why DCTCP and ECN?

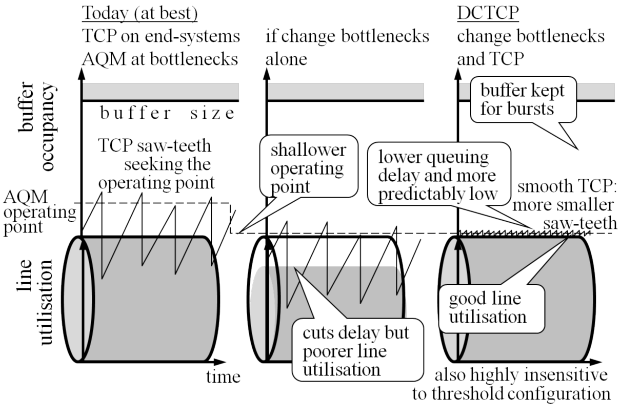

Figure 2: Data Centre TCP: Intuition

Figure 2 shows how DCTCP resolves the dilemma caused by TCP's large sawteeth (problem #1 in the Introduction). DCTCP's smaller sawteeth allow both maximal utilisation of the link (the grey cylinder) and minimal occupancy of the buffer (above the cylinder). The size of DCTCP's sawteeth also remains invariant with increasing bit-rate, which addresses problem #3.

DCTCP also solves problem #2 above, by requiring the network to signal congestion immediately, without any smoothing delay (not illustrated in the figure).

The underlying reason that DCTCP can solve problems #1 and #2 is that it signals congestion with explicit congestion notification (ECN [14]). ECN is purely a signal, whereas drop is both an impairment and a signal, which compromises signalling flexibility:

1. DCTCP's finer sawteeth imply a higher signalling rate, which would be untenable as loss;
2. If the queue grows, an AQM holds back from introducing loss in case it is just a sub-RTT burst, whereas emitting ECN immediately is harmless.

This last point is most significant, because the sender knows its own RTT, which DCTCP could use as the time constant to smooth incoming ECN marks. Whereas, with drop, the network has to smooth but it does not know each flow's RTT, so it has to use a worst-case (transcontinental) RTT, to avoid instability if a flow does have such a long RTT.

ECN also offers the obvious latency benefit of near-zero congestion loss, of most concern to short flows. 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.

To be concrete, in our testbed, we identify classic traffic by a cleared ECN field in the IP header (not ECN-capable). In Section 8, we discuss an alternative but non-preferred compromise that would allow Classic traffic to use ECN as well.

The coexistence mechanism has nothing to do with flow identifiers; it solely contrives the two aggregate signalling levels to compensate for the different responses to the signals exhibited by flows in each aggregate.

2.2 Delay vs. Drop

Evaluating our Coupled AQM at high load, we noticed that PIE and fq_Codel produced unusually high drop, which took over from queuing as the dominant cause of delay for short Web flows. Thus, the goal of AQMs like PIE and fq_CoDel—to cap queuing delay at a fixed target—is misguided. If TCP cannot increase queuing delay, it will increase drop instead.

Our coupled AQM constrains classic traffic with a softened delay target, but not as soft as RED. We call the mechanism Curvy RED. The original RED algorithm increases drop probability linearly with queue growth. With a curviness parameter of 2, the Curvy RED algorithm increases drop with the square of the queue, so it pushes back increasingly aggressively the more the queue grows. This is implemented by simply comparing the queue to two random numbers, not one. It also uses queuing time, not queue length.

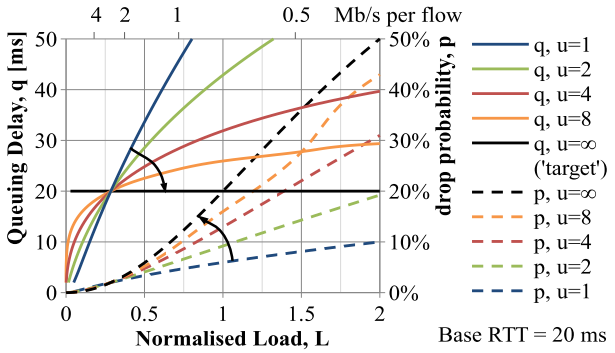

Figure 3: The Delay-Loss Dilemma: Sensitivity to Load for Curvy RED AQMs with Increasing Curviness, u.

The order of the legend follows that of the curves. Normalised load, $L \propto 1/$ bit-rate per Reno flow (top).

Conceptually, as the curviness parameter tends to infinity (shown by the two pincer arrows in Figure 3 from [2]), it approximates the hard delay target of PIE or CoDel, shown as a flat step configured at 20 ms. To clamp delay to a hard constraint, the corresponding dashed drop curve has to increase aggressively with load. In this paper, we show that intermediate curviness can still keep queuing delay under control whatever the load, but without having to introduce so much loss that loss itself becomes the dominant cause of delay.

2.3 Per-Flow Queuing and Self-Harm

Superficially, it might seem that per-flow queuing (as in fq_CoDel) would address the problems in Section 1; 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 saw-toothing, which the flow still inflicts upon *itself*.

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 Web applications multiplex streams into a single flow, e.g. SPDY or HTTP/2; then they alter which stream they prioritise, depending on the progress of each of the streams already sent. Our panoramic interactive video app is another example, where the amount of data sent (the frame rate) at any instant depends on how much prior data was delivered. For such applications, a remote queue is not useful if their self-induced queue is remote; once optional data is in flight, they cannot suppress it. To adapt how much they send, they need to maintain their send-queue locally.

Per-flow queuing was not an appropriate solution, given it i) does not solve the self-induced latency problem; ii) needs to inspect transport layer headers (preventing transport evolution); and iii) requires many more queues and supporting scheduling structures, whereas our primary goal was to *reduce* cost and complexity.

3. STEADY STATE RATE

An L4S congestion controller such as DCTCP achieves low latency, low loss and low throughput variations by driving the network to give it a more responsive signal with a higher resolution. Therefore, a solution must be found to reduce the congestion signal intensity for Classic congestion controllers (TCP Cubic and Reno), but not for L4S (DCTCP).

Our first concern is to support rough equality in terms of long term throughput (other factors being equal). In terms of throughput for short flows, we want to allow differentiation to support lower latency, less loss and less throughput variation for the improved class.

For our purposes, it is sufficient to ignore dynamic aspects, and apply the simplified models in (1) and (2) for TCP Reno and Cubic paket rate as in [12] and [7]:

$$r_{reno} = \frac{1.22}{p^{1/2}T} \quad (1) \qquad r_{cubic} = \frac{1.17}{p^{3/4}T^{1/4}} \quad (2)$$

where p is drop probabillity and T is RTT.

The Cubic implementation in Linux provides a fallback to TCP Reno when RTT is small. Due to the

different decrease factor, the steady state rate will deviate from (1) with a slightly higher constant (3).

The implicit switchover RTT can be derived from (2) and (3). Pure Cubic behaviour (as defined in (2)) will become active when (4) is false.

$$r_{cubic} = \frac{1.68}{p^{1/2}T} \quad (3) \qquad r * T^{5/2} < 3.5 \quad (4)$$

For typical user access from local data centres ($T <$ 20 ms), pure Cubic behavior can only be expected for flows faster than 500Mbps. Therefore, at least for AQMs in access networks, we can focus on the relationship between Reno and DCTCP and assume that Cubic will be in Reno mode, or at least not too far outside it.

We have derived a similar model for DCTCP, assuming an idealised uniform deterministic marker, which marks every $1/p$ packets. A DCTCP congestion controller has an incremental window increase per RTT $\alpha = 1$ and a multiplicative decrease factor $\beta = p/2$ (with p being estimated). So, every RTT, W is increased by $W \leftarrow W + 1$, meaning that under steady state, this must be compensated every RTT by (5). This decrease is steered by marks, as defined in (6).

$$W \leftarrow \left(1 - \frac{1}{W}\right) W \quad (5) \qquad W \leftarrow \left(1 - \frac{p}{2}\right) W \quad (6)$$

From (5) and (6), we see that to preserve this balance, (7) must be true, which determines the window and therefore the steady state packet rate in (8).

$$\frac{p}{2} = \frac{1}{W} \quad (7) \qquad r_{dc} = \frac{2}{pT} \quad (8) \qquad r_{dcth} = \frac{2}{p^2 T} \quad (9)$$

Note that (9) derived in the DCTCP paper [1] has a different exponent of p compared to (8). The reason is that (9) is defined for a step threshold, which causes an on-off marking pattern. When DCTCP marking is coupled to the drop probability of Classic traffic, it uses fractional marking probability, so (8) will be applicable.

4. COUPLED AQM FOR EQUAL RATE

Knowing the relation between network congestion signal, (mark or drop) probability and the flow rate, we can adjust the feedback from the network to each type of congestion control. For TCP Reno and DCTCP, we substitute (1) and (8) in $r_{reno} = r_{dc}$:

$$\frac{1.22}{p_{reno}^{1/2} T_{reno}} = \frac{2}{p_{dc} T_{dc}} \quad (10)$$

If the RTTs are equal (they may not be; see Section 5), we can arrange the rates to be equal using the simple relation between the probabilities, defined in (11). This relation could be achieved by a modified AQM in the network, shown in Figure 4.

$$p_{reno} = \left(\frac{p_{dc}}{1.63}\right)^2 \quad (11)$$

Probabilistic mark/drop is typically implemented by comparing the probability p with a pseudo-random generated value R per packet. A signal is applied for a packet when $p > R$. The advantage of using relation (11) is that p^2 can easily be acquired by comparing p with 2 psuedo-random generated values and signal only if both random values are smaller than p: $p > max(R_1, R_2)$.

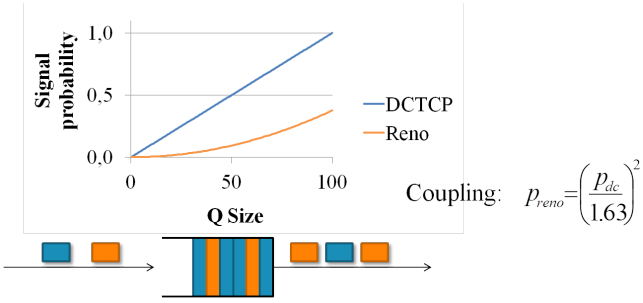

Figure 4: Equal rate AQM for TCP Reno and DCTCP

A square curve is a simple way to emulate the three piecewise-linear parts of RED. Drop probability hugs the zero axis, while the queue is shallow. Then, as load increases, it introduces a growing 'barrier' to higher delay, but with only one parameter, not three.

To configure our evaluations, we used DCTCP (8) and Reno (1) behaviour as the gold standards for the L4S and Classic classes respectively, so we used (11) as the main coupling relation.

The approach is encapsulated by the phrase "Think once to mark, twice to drop". This concept was first verified by simulation. The results are not shown due to space limitations, but confirmed that for any number of flows with any combination of RTT's, the steady state throughput was independent of the congestion controller (DCTCP or Reno) if the signal probability was coupled, as in Figure 4, and the window sizes of flows were above four packets.

In addition, further analysis revealed a different sensitivity to multiple bottlenecks. In worst case (all bottlenecks signalling the same probability), DCTCP's throughput depends on the accumulated probabilities, while Reno's by the square root of it. In a realistic situation with 2 dominant bottlenecks, DCTCP would have maximum $\sqrt{2}$ times less throughput.

Coupling signals supports rate equalisation, but as long as there is only one queue, Classic traffic spoils the consistent low queueing latency of DCTCP traffic (unless utilization is sacrificed).

5. DUAL QUEUE FOR LOW LATENCY

To achive low latency for L4S traffic in the presence of Classic traffic, we still use a coupled AQM, but across two queues as shown in Figure 5. In our experiments, if the ECN field in the IP header is not cleared, we classify the packet into the L4S (L) queue, otherwise into Classic (C). More sophisticated classification might be necessary (see Section 8).

To minimise latency for L4S traffic, we schedule the

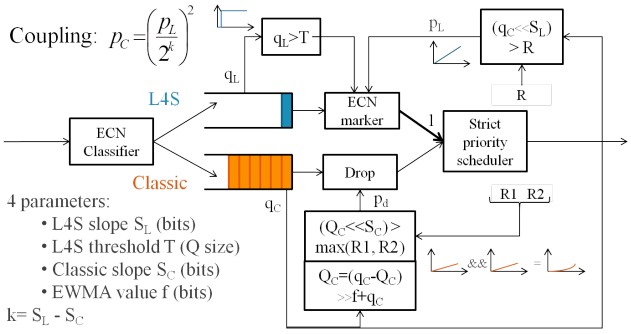

Figure 5: Dual Queue Coupled AQM

L4S queue with strict priority, and the Classic queue only when the L4S queue is empty.

The ECN signal from the L4S queue is coupled to the queuing time of the Classic queue (also known as sojourn, waiting or service time). The coupled feedback ensures that flows share capacity correctly across the two queues. The L4S queue size remains low enough to keep the Classic queue flowing. Policing unresponsive flows is a policy issue that needs to be separate from a basic AQM, but the scheme does need to handle overload[1].

Whenever there are packets in the Classic queue, the coupled ECN feedback that the L4S queue emits already depends on its own utilisation (via the Classic queue). However, the L4S queue needs to be able to emit ECN signals if L4S load causes the L4S queue itself to grow, particularly if there is no Classic traffic to generate any coupled feedback. For now, in addition to any coupled feedback, the L4S queue applies a shallow step function without any smoothing delay, as used for DCTCP in data centres (see [5, §10] for other possibilities).

When there is traffic in both queues, (11) gives the desired coupling between the drop and marking probabilities in the two classes that should achieve our objective of roughly equal flow rates (other factors being equal). For implementation efficiency, we approximate the denominator by an integer power of 2, giving:

$$\boxed{p_C = \left(\frac{p_L}{2^k}\right)^2}. \tag{12}$$

The resulting coupled AQM needs just 2 parameters for each queue. For the Classic queue:

S_C To convert the current Classic queue sojourn time into a dropping probability in the range $[0, 1)$ requires a scaling parameter. To make multiplication efficient, we use an integer power of two, so we define the slope of the AQM's square curve, relating Classic queueing time to drop probability from the Classic queue as 2^{S_C};

f To support smoothing the sojourn time of the Classic queue and make multiplication efficient, we will use an integer power of two for the EWMA constant, which we define as 2^{-f}.

For the L4S queue:

S_L As for the Classic queue, we define the slope of the AQM's marking function as 2^{S_L};

T The queue size at which step threshold marking starts in the L4S queue.

The average queue sojourn time Q_C is calculated every time a packet is dequeued from any of the queues for scheduling. The calculation is done according to (13), with q_C the sojourn time of the latest packet in the Classic queue and f the smoothing exponent.

$$\begin{aligned} Q_C &\leftarrow 2^{-f} q_C + (1 - 2^{-f}) Q_C \\ &\leftarrow Q_C + (q_C - Q_C) >> f, \end{aligned} \tag{13}$$

where $>>$ is a right bit shift.

The queuing time of the Classic queue then drives the marking probability of L4S packets and dropping probability of Classic packets as follows:

$$p_L = 2^{S_L} q_C, \tag{14}$$

$$p_C = (2^{S_C} Q_C)^2. \tag{15}$$

For either case, 2^{-S_L} or 2^{-S_C} represent the queuing time in the Classic queue at which marking or dropping probability reaches 100%. Assuming a steady state and no under-utilisation, $Q_C = q_C$. So, substituting (14) & (15) into (12) we have:

$$k = S_L - S_C. \tag{16}$$

Therefore, k represents the strength of coupling between the two queues, but it is not an additional parameter.

Using the dual queue AQM with coupled feedback will ensure that the L4S traffic leaves sufficient capacity unused as long as there are packets in the Classic queue. However, the rate ratio between DCTCP and Cubic(Reno) traffic in equation (11) was derived assuming the RTTs of the two types of flow were equivalent. Typically, there would be hardly any delay in the L4S queue, but substantial delay in the Classic queue. The resulting increase in a Classic flow's RTT could be substantial such that (11) no longer held, then (17) would have to be used instead.

$$\frac{r_{reno}}{r_{dc}} = \frac{1.22}{2} \frac{p_{dc}}{p_{reno}^{1/2}} \frac{T_{dc}}{T_{reno}}, \tag{17}$$

with T_{reno} and T_{dc} being the total RTT composed of the delay in their respective queue and a common base RTT T_0 for the rest of the network.

Some value has to be chosen for k in (12) to relate the rates of L4S and Classic flows, given the possible discrepany between their RTTs. This is a policy decision for the network operator, which could decide to use:

[1] A tradeoff needs to be made between complexity and the risk harming Classic flows. It is an operator policy to define what must happen if the service time of the classic queue becomes too big. Actions can include delay based scheduling, common drop, etc...

- $2^k \approx 1.63$ from (11) on the basis that Classic flows have only themselves to blame if the queue they build reduces their rate;
- $2^k \approx 1.19$ on the same basis, but if it judged that most Classic traffic would be in Cubic mode.
- $2^k \approx 1.63(T_0+q_C)/T_0$ to compensate for the extra queue delay q_C for exact throughput equality.[2]

For the first two policies respectively $k = 1$ and $k = 0$ would be the closest values. In the the last case, the Classic queue delay q_C depends on the slope of the Classic AQM S_C. For the slope used in our experiments, $q_C \approx 4 * T_0$, therefore $2^k \approx 1.63 * (4+1) \approx 8$, which implies $k = 3$. Of course, given T_0 is bigger towards third party servers, the RTT ratio will become smaller, and any overcompensated factor will be to the disadvantage of L4S flows. In our testbed experiments, we adopted the throughput equality policy to allow an easy comparison of the results. However, there is no implication that this policy is recommended.

The pseudocode below summarises the implementation of the above analysis used for all experiments.

Algorithm 1 Dequeue for Dual Queue Coupled AQM

```
1: if LQ.DEQUEUE(pkt) then
2:     if (LQ.LEN() > T) ∨ ((CQ.TIME() << S_L) > RND()) then
3:         MARK(pkt)
4:     end if
5:     RETURN(pkt)        ▷ return the packet and stop here
6: end if
7: while CQ.DEQUEUE(pkt) do
8:     Q_C += (PKT.TIME() − Q_C) >> f_C        ▷ C EWMA
9:     if (Q_C << S_C) > MAX(RND(), RND()) then
10:        DROP(pkt)                ▷ Squared drop, redo loop
11:    else
12:        RETURN(pkt)  ▷ return the packet and stop here
13:    end if
14: end while
```

6. TESTBED SETUP

We have 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 from carrier grade equipment used for testing customer solutions. Figure 6 depicts the testbed, which consists of a classical residential service delivery network composed of Residential Gateway, xDSL DSLAM (DSL Access Multiplexers), BNG (Broadband Network Gateway), Service Routers (SR) and application servers. A residential user's 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.

[2]Assuming the queueing delay of the L4S queue is negligible and T_0 is the same for Reno and DCTCP flows.

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 and start applications and test traffic. The two client-server pairs (A and B) are respectively configured with the same TCP variants and applications.

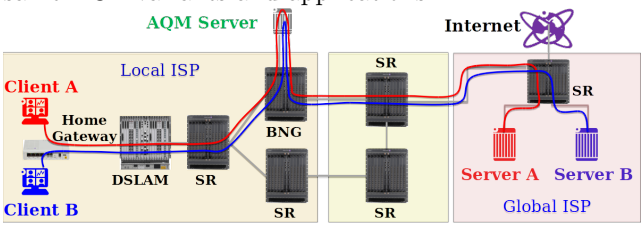

Figure 6: Testbed configuration

Within a BNG, per-customer queues form the leaves of a hierarchical scheduling tree. In a production access network, the BNG is deliberately arranged as the downstream bottleneck for the per customer queues. A Linux server (AQM server) is used to create this bottleneck and to configure the different AQMs that were evaluated. Traffic from the client-server pairs is routed from the BNG through this Linux box so as to simulate the function proposed for the BNG. This server also controls the experiments and captures and analyses the traffic. In practice it would also be important to deploy AQM in the residential gateway, but to minimise side-effects we kept upstream traffic below capacity.

The following setup was used for the evaluations (Section 7): Two client computers were connected to a modem using 100Mbps Fast Ethernet; the xDSL line was configured at 48Mbps downstream and 12Mbps upstream; the links between network elements consisted of at least 1GigE connections. The AQM server at the BNG created a 40Mbps bottleneck, before the configured AQM for the downstream traffic. No bottlenecks were explicitly configured for the upstream traffic. All Linux computers were Ubuntu 14.04 LTS with kernel 3.18.9, which contained the implementation of the TCP variants and AQMs. The base RTT (T_0) between the clients and servers was 7 ms, which primarily originated from the xDSL interleaved FEC configuration.

The experiments used DCTCP, Cubic and Reno with their default values. For ECN-Cubic, we additionally enabled TCP ECN negotiation on the relevant client and server. The AQM configurations used the options as described in Table 1, unless otherwise stated.

All	Buffer: 1000 pkt (320ms @40Mbps), ECN
RED	Min thres: 80 pkt (24 ms @40Mbps) Max thres: 240 pkt (72 ms @40Mbps) Burst: 220 pkt (66 ms @40Mbps)
PIE	Target delay: 20 ms, Burst: 100 ms
fq_CoDel	Target delay: 5 ms, Burst: 100 ms
DualQ	$S_C = 1$, $f = 5$, $S_L = 4$, $T = 5$

Table 1: Default parameters for the different AQMs.

7. EVALUATION

To assess our primary objective (long term throughput equivalence), we performed experiments on different AQMs with long-running flows. We used steady flows not as an example of a realistic Internet traffic mix, rather as a situation where starvation can typically occur. We also evaluated the AQMs under dynamic load to verify the impact of low latency and loss on completion time of short flows.

Each experiment (lasting 250 s) was performed between client-server pairs A and B, with a specified TCP variant configured on each client-server pair and a specified AQM on the AQM server.

For all long-running flow experiments, each client started 0 to 10 file downloads on its matching server, resulting in 120 combinations. To preserve details, including variability over time, while visualising the overall outcome, we show flow thoughtput, queue delay, marking and dropping probability in an overview matrix for all 120 combinations.

The row and column labels indicate the TCP variant (C:Cubic, E:ECN-Cubic, D:DCTCP) followed by the number of active flows on the respective client-server pairs. The left matrix label shows the AQM used and the X and Y ranges.

For all dynamic behaviour experiments, 25 load combinations were tested on each client-server pair. Again, the row and column labels (X0+L; X0+H, X1; X1+L; X1+H) indicate TCP variant (X=C,E,D), the number of long-running flows (0 to 1), and the dynamic load level which is an exponential arrival process with 100ms (L=low load) or 10ms (H=high load) average interarrival times (or none), requesting a Pareto ditributed download size $\alpha = 0,9$ with a minimum size of 1KB. Every request opened a new TCP connection, closed by the server after sending the data.

Plotted values of flow or class throughput and marking or dropping probabilities are always measured over each second. Flow completion time plots show a dot per download size and time between opening the connection and receiving the data. A reference line shows the completion time that a perfect lone download would have achieved at full line rate after a 2-RTT handshake. The Queue delay CDF plots the sojourn time of every packet for each traffic class. For flow throughput, the Y range was adapted to locate the expected throughput of the N dominant flows across the middle of the graph ($80/N$ Mbps). For example, if 4 Cubic flows compete over the 40Mbps bottleneck, the number of dominant flows is 4, the expected throughput is 10Mbps and the scale 20Mbps. If Cubic flows are starved when 5 Cubic and 2 DCTCP flows compete, then only $N = 2$ flows are dominant, and the upper limit of Y will be 40Mbps. This results in overall comparable plots, scaling the visualisation optimally.

7.1 Long term throughput equivalence

To demonstrate the starvation problem that the DualQ aims to solve, the first evaluation uses DCTCP and Cubic over RED, PIE, fq_CoDel and DualQ AQMs.

The results are plotted in Figures 7 & 8. As expected, the DCTCP flows take most of the available bandwidth from Cubic on the single Q AQMs (RED and PIE).

The queue delay in the RED AQM is very high, but with moderate dropping and marking probability. PIE[3] dropping probability exceeded the 10% limit at which it switches to dropping ECN traffic, and the DCTCP response to drop was incorrectly implemented (see [3] for details).

fq_CoDel seems to handle DCTCP quite well, providing every flow with an almost perfectly stable and equal rate, except when the statistical buffer assignment fails to use a unique buffer per flow. If flows of the same class land in the same Q, the throughput deviation from the equal rate is only 50%. If flows of different classes are assigned to the same buffer, the Cubic flow starves (as in Figure 7 D:8-C:7). This behaviour results in sporadic and hard to reproduce random failure of applications, with potential frustration for users and service support. From a queuing delay perspective, unlike PIE, fq_CoDel is not able to keep the DCTCP flows at its (smaller) target delay, but delay is still low. However, as predicted, the drop probability of fq_CoDel rises quickly with load.

Our DualQ Coupled AQM is able to guarantee equal throughput between DCTCP and Cubic flows. It deviates slightly due to the Classic Q size, which grows on higher load, resulting in less throughput for the Classic flows, and is smaller at lower load, resulting in a higher throughput for the Classic flows. The queuing delay for the L4S traffic is stunningly low—so low that the CDF plots are nearly perfect step functions. In the D:10-C:10 combination, the marking probability approaches 100%. Combinations with larger number of flows revealed a previously unnoticed limit to TCP's ability to scale to low queuing delays, which needs to be fixed, at least in DCTCP. We have explained this in a separate technical report [3]. Essentially, DCTCP or TCP will override any AQM and increase queuing delay to keep at least 2 segments in flight. For Classic traffic, compared to fq_CoDel, loss levels are kept to reasonable levels by relaxing the delay constraint somewhat (see Section 2.2).

Further experiments with adding a 10 or 20 Mbps unresponsive UDP CBR flow[3] also showed an important difference beween fq_CoDel and DualQ. fq_CoDel assumes that capping the CBR flow rate to an equal share is always the correct policy. Whereas the DualQ AQM allows applications to determine their flow rate and the responsive flows to share the remaining

[3]See complementary technical report [3] for these plots.

30 or 20 Mbps evenly. For instance, DualQ would support unresponsive multicast TV up to the link capacity whereas fq_CoDel would divide this by the current number of large flows. Further fq_CoDel treats a VPN tunnel containing many flows as one, and can prevent background/delay-based flows from yielding their throughput to others. Worth noting that unfortunately, in our fq_CoDel D:1-C:1 combination, the 20Mbps flow got classified in the same buffer as the CBR flow, resulting in a starving DCTCP flow. fq_CoDel's buffer collisions were more frequent than expected.

7.2 Evaluation under dynamic load

Figure 9 shows the results for 6 dynamic experiments. We did the (ECN) Cubic experiments as a benchmark to distinguish how much improvement was due to ECN, DCTCP or DualQ. RED[3] and PIE show similar results. fq_CoDel approximates the perfectly equal share for completion times. The small queuing delay (5 ms) just accomodates Cubic's needs preventing underutilization.

In completion time results for Cubic, we see 2 levels of timeouts: at 1 s (due to lost SYN) and 300 ms (due to lost FIN). Using ECN-Cubic does not reduce the number of lost SYN/ACK/FIN packets, since they don't carry the ECN capability in the IP header, the first two in compliance with the ECN standard (and see below regarding FINs). Interpreting the results, we found an anomaly in the Linux implementation. Flows with 1 or 2 packets of data (below 3KB of data) keep experiencing lost packet timeouts of 200ms. The reason is that 2 packets are sent without delay, and a later close connection call results in a separate FIN packet sent without ECN. The current DCTCP implementation uses ECN flags in the IP header of all packets.

In Experiment 4 (PIE), the dots at 300ms (due to drop of final packets) indicate that burst allowance is not effective if other long flows are present. The results for RED[3] were almost identical. Using ECN clearly has an advantage, resulting in shorter completion times as drop is partially avoided. fq_CoDel's burst allowance is effective as it works per flow, and ECN provides no significant improvement (Experiment 5). Sporadic occurances can be attributed to queue collision with an ongoing flow. Also, lower completion times for ECN-Cubic due to less retransmissions of other dropped packets can be identified. ECN has no significant impact on queuing delay.

In Experiment 6 we configured the DualQ AQM with parameters adapted to ECN-Cubic. We halved the L4S slope, additionally applying a squared probability to the ECN-Cubic. As a result, we see significant improvement for the completion times, similar to fq_CoDel. Also, we see near zero queuing delay for the L4S traffic,

but as a downside, a significant reduction in utilisation when competing with many short flows.

Comparing Experiment 5 to 7, Cubic has more completion time outliers when a long running DCTCP flow is active, probably due to fq_CoDel's buffer collisions. Additionally, long running DCTCP flows have a much larger queue, explaining the full utilisation.

Comparing Experiments 5, 7 with 6, 8, we can again conclude that our DualQ AQM very much approximates the fq_CoDel AQM without the need for flow identification and more complex processing. The main advantage is DualQ's lower queuing delay for L4S traffic. Compared to Cubic, DCTCP improves utilisation, as it reduces the throughput more appropriately on congestion signals, but can only regain the available capacity incrementally when a short flow ends. One issue with DCTCP also becomes apparent for flows bigger than the initial windows size of 10. As marking probablity is much higher, slow start will get consistently prematurely interrupted. A good result is that no slow start overshoot is detected (zero Q delay), but it leads to unnecessarily longer completion times. The outcome suggests that a gradual slow start exit scheme is possible. Again, Dual Q queuing delay is nearly perfect for DCTP traffic, and even for Classic traffic its delay is nearly as good as fq_CoDel.

8. STANDARDISATION REQUIREMENTS

An identifier will need to be standardised to distinguish L4S and C packets. In our tests we used a cleared ECN field to indicate C packets and L4S otherwise.

The ECN standard [14] currently defines a mark as equivalent to a drop, but discussions have started at the IETF on changing its meaning [16, § 5]. It is being questioned whether merely preventing drop offers enough performance improvement for an operator to countenance the cost and risk of deployment.

For those who have managed to get classic ECN widely deployed on servers, moving the goalposts at this stage would be harsh. However, despite widespread server deployment there is no evidence that any public network operator is considering or has deployed ECN, even though it was standardised in 2001. Whereas private data centre operators do redefine the ECN field for the predictable latency of DCTCP.

If the meaning of ECN cannot be changed from "equivalent to drop", it would be possible to identify the L4S service in another way, e.g. a combination of ECN and Diffserv, or the ECT(1) codepoint. However, the Diffserv codepoint is not preserved end-to-end and it may be argued that the last ECN codepoint should not be burned when the current one is not being used.

The square relationship between L4S marking and Classic drop (Eqn. (12)) would need to be standardised, but it would be better to recommend rather than standardise a value for k, given differences would not prevent interoperability.

9. RELATED WORK

In 2002, Gibbens and Kelly [6] 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 [11, §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. Wu et al. [17] investigates a way to incrementally deploy DCTCP within data centres, marking ECN when the temporal queue exceeds a shallow threshold but using standard [14] ECN on end-systems. Kuhlewind et al. [10] showed that DCTCP and Reno could co-exist in the same queue configured with a form of WRED classifying on ECN not Diffserv. Judd [9] uses Diffserv scheduling to partition data centre switches between DCTCP and classic traffic in a financial data centre scenario. The technical report complementing this paper gives fuller reviews of each of these sources and more [5].

10. CONCLUSION

Extensive tests of our novel Coupled Dual Queue AQM within the edge gateway of a broadband access testbed have verfied that it elegantly solves the problem of co-existence between traffic like Data Centre TCP (DCTCP) and classic TCP traffic. This means that ISPs can offer a new form of unmanaged Internet service that we call Low Latency Low Loss and Scalable (L4S) without existing 'Classic' traffic losing per-flow throughput. In our tests the 99th %-ile queuing delay of L4S was 1 ms; more than an order of magnitude lower than that of PIE or fq_CoDel. We identified self-delay as a new concern for a novel breed of rapidly adaptive applications. L4S will also allow TCP throughput to scale indefinitely, which will otherwise soon become problematic.

In the Linux implementation of our AQM, L4S and Classic together consume fewer instructions per packet than even the simplest form of RED. No flow-ID inspection or per-flow queuing is needed.

Further work is needed to verify the parameter insensitivity of Dual Queue in a wide range of settings and to determine the best AQM for Classic traffic that drops with the square of the probability of L4S traffic. Curvy RED shows promise, and we plan to explore it more fully, particularly with more convexity, but taking care with stability. We will also determine whether a PI controller might be better.

DCTCP needs numerous improvements: fall-back to Reno on loss; loss-resilient feedback [4]; smoothing incoming feedback over the flow's own RTT estimate; smoothing the exit from slow-start; faster than additive increase; and pacing a fractional window [3].

11. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[2] B. Briscoe. Insights from Curvy RED (Random Early Detection). Technical report TR-TUB8-2015-003, BT, May 2015. http://riteproject.eu/publications/.

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

[4] B. Briscoe, R. Scheffenegger, and M. Kuehlewind. More accurate ECN feedback in TCP. IETF Internet Draft draft-kuehlewind-tcpm-accurate-ecn-03, work in progress, July 2014.

[5] K. de Schepper, O. Bondarenko, I. Tsang, and B. Briscoe. Data Center to the Home. Technical report, RITE Project, June 2015. http://riteproject.eu/publications/.

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

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

[8] T. Hoeiland-Joergensen, P. McKenney, D. Täht, J. Gettys, and E. Dumazet. Flowqueue-codel. Internet Draft draft-hoeiland-joergensen-aqm-fq-codel, work in progress, June 2014.

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

[10] M. Kühlewind, D. Wagner, J. Espinosa, and B. Briscoe. Using data center TCP (DCTCP) in the Internet. In *Globecom Workshops (GC Wkshps), 2014*, pages 583–588, Dec 2014.

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

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

[13] R. Pan et al. PIE: A lightweight control scheme to address the bufferbloat problem. In *Proc. IEEE Int'l Conf. on High Performance Switching and Routing (HPSR)*, pages 148–155, 2013.

[14] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. RFC 3168 (Proposed Standard), Sept. 2001.

[15] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP approach for high-speed and long distance networks. In *Proc. IEEE Conference on Computer Communications*, pages 1–12, 2006.

[16] M. Welzl and G. Fairhurst. The Benefits to Applications of using Explicit Congestion Notification (ECN). Internet Draft draft-welzl-ecn-benefits-02, IETF, Mar. 2015. (Work in Progress).

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

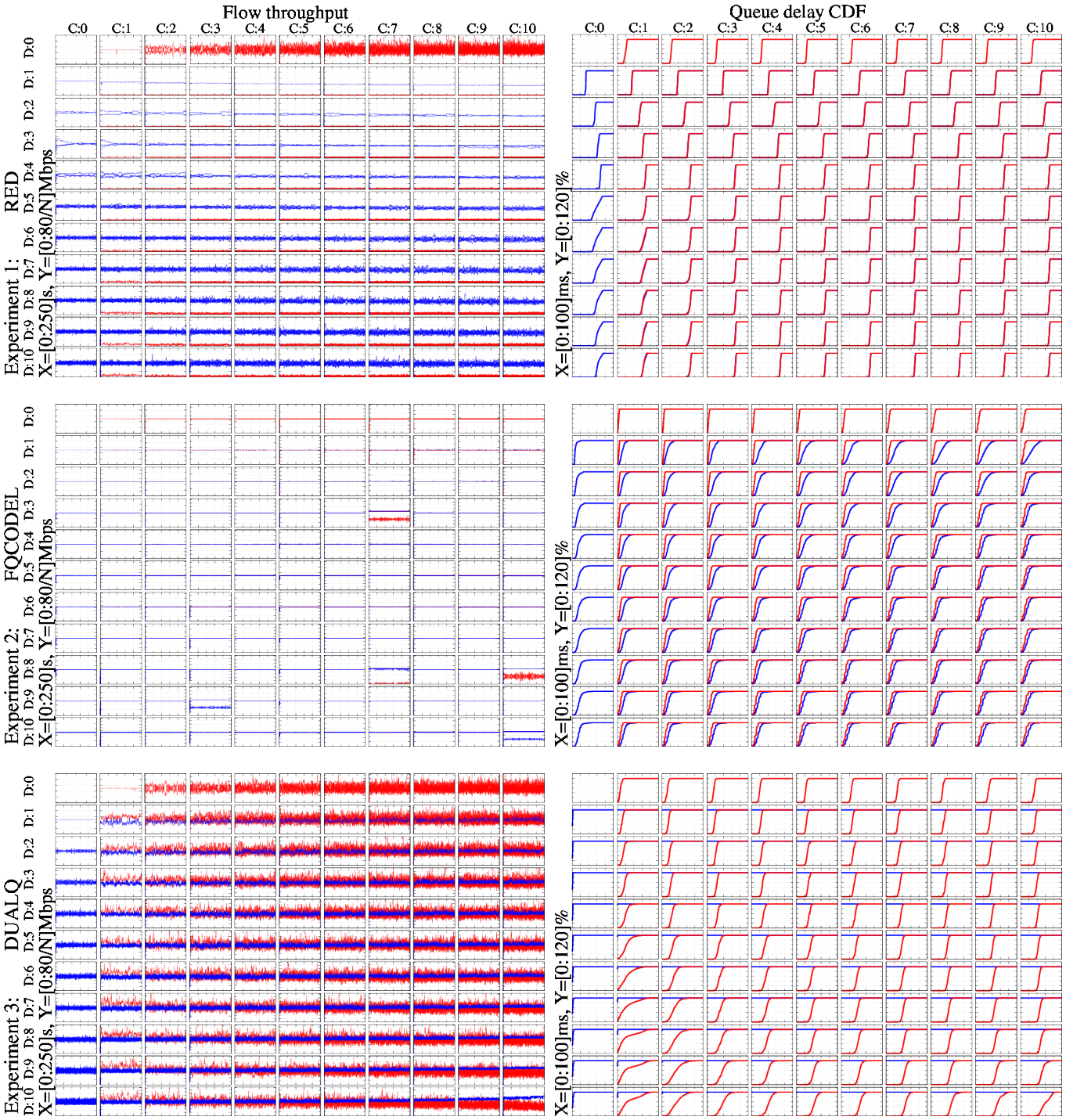

Figure 7: AQM comparison for long flows
Showing the coexistence problem (Exp 1) and potential solutions (Exps 2 & 3).
D: Number of DCTCP flows (blue), C: Number of Cubic flows (red), N: the number of expected dominant flows.

Note: With Dual Q, the queue delay CDFs for DCTCP (blue) are hard to see, because they are all nearly perfect step-functions.

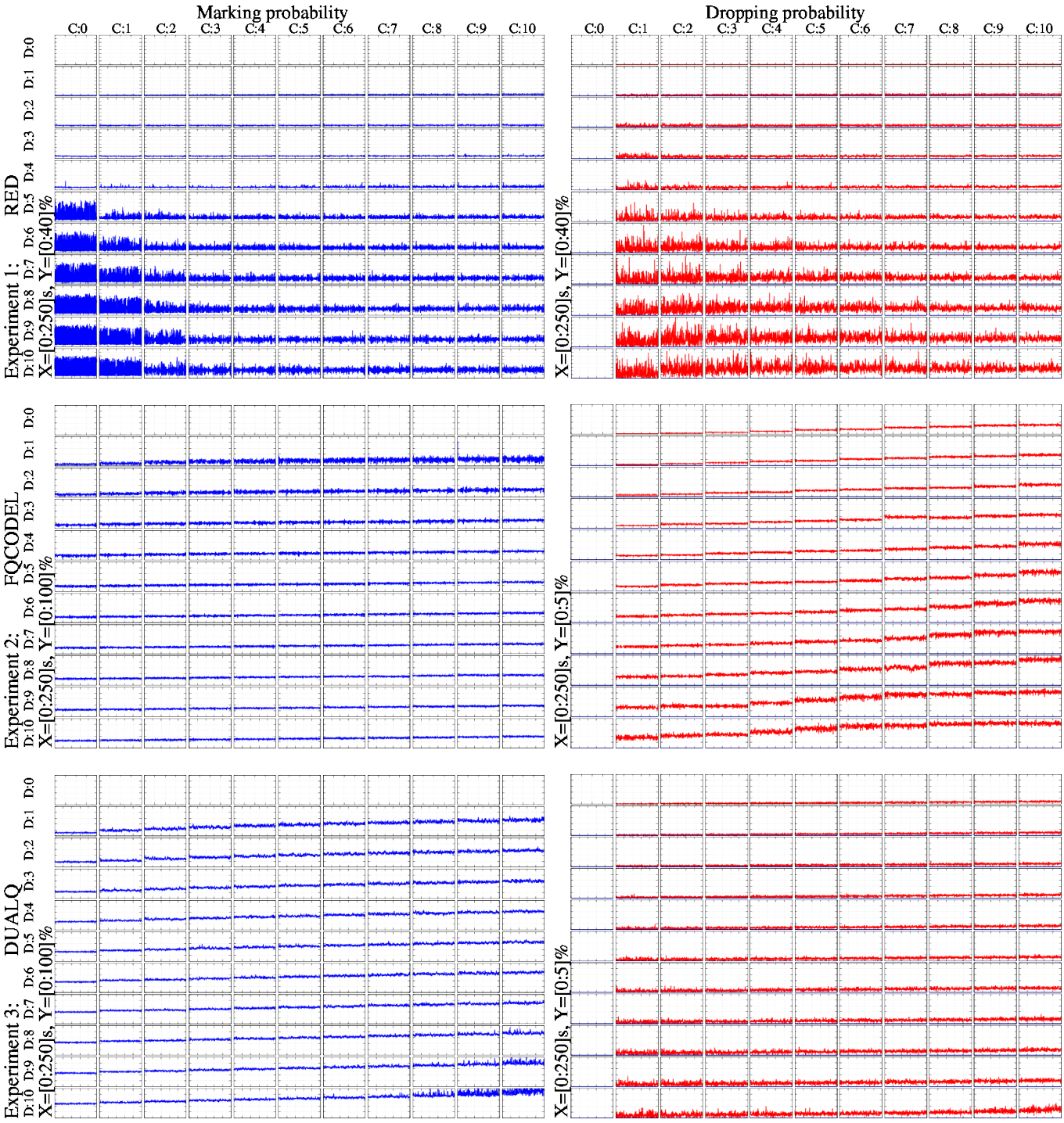

Figure 8: AQM comparison for long flows (cont.)
D: Number of DCTCP flows (blue), C: Number of Cubic flows (red).

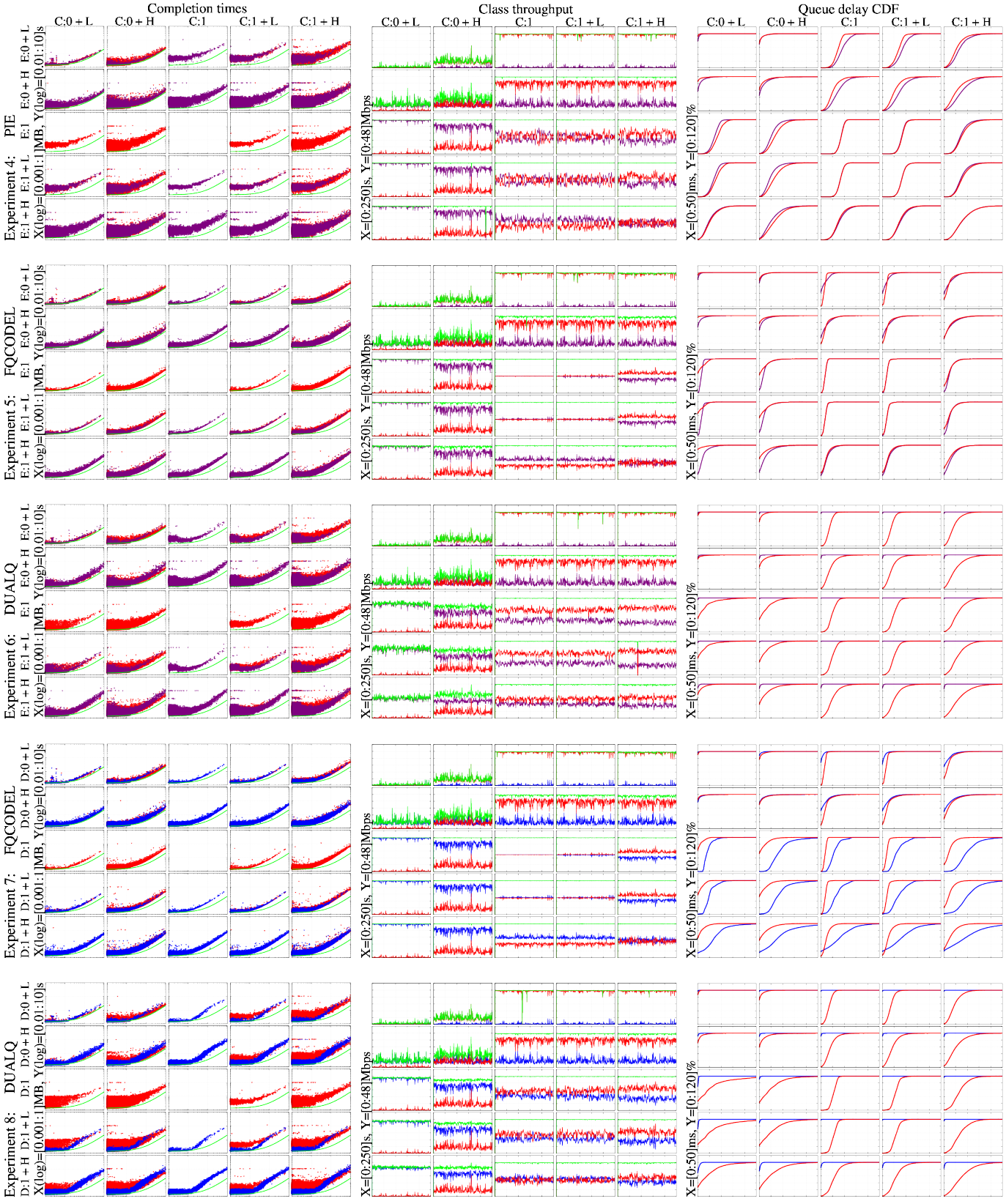

Figure 9: Dynamic load for different AQMs for ECN-Cubic & Cubic (Exps 4–6) or DCTCP & Cubic (Exps 7 & 8). E: Number of ECN-Cubic flows (purple), D: Number of DCTCP flows (blue), C: Number of Cubic flows (red), Green = total throughput, showing utilisation. L: 100 ms exponential load, H: 10 ms exponential load.
Note: With Dual Q, the queue delay CDFs for DCTCP (blue) are hard to see, because they are all nearly perfect step-functions.