Draft

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

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

***Abstract*—Traditionally, ultra-low queueing delay and capacity-seeking are considered mutually exclusive. We introduce a service that offers both: Low Latency Low Loss Scalable throughput (L4S). Therefore, it can incrementally replace best efforts as the default Internet service. It exploits the properties of ‘Scalable’ congestion controls (defined later, but Data Centre TCP is a well-known example). Under a wide range of conditions emulated on a testbed using real residential broadband equipment, it proved hard not to get remarkably low (sub-millisecond) average queuing delay, zero congestion loss and full utilization. To realize these benefits we had to solve a hard problem: how to incrementally deploy scalable congestion controls on the public Internet. The solution uses a ’DualQ Coupled Active Queue Management’ structure at the access link bottleneck. The coupled AQM enables balance (window ‘fairness’) between Scalable and Classic (Reno, Cubic, etc.) flows, without inspecting deeper than the IP header. It counterbalances the more aggressive response of Scalable flows with more aggressive marking. It identifies L4S packets using the last Explicit Congestion Notification codepoint in the IP header, which the IETF is in the process of allocating. The DualQ 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 millions of experiment runs designed to test our claims.**

## I. Introduction

With increases in bandwidth, latency is becoming the critical performance factor for many, if not most, applications, e.g. Web, voice, conversational and interactive video, interactive remote presence, finance apps, instant messaging, online gaming, cloud-based apps, remote desktop and video-assisted remote control of machinery and industrial processes. Latency is a multi-faceted problem that has to be tackled on many different fronts [10] and in all the different stages of application delivery—from data centres to access links and within end systems.

The aspect this paper addresses is the variable delay due to queuing. Queuing is intermittent, only occurring when a sufficiently long-running ‘greedy’ (capacity-seeking or TCP-like) flow happens to coincide with interactive traffic [25]. However, intermittent delays dominate experience, and many real-time apps adapt their buffering to these intermittent episodes.

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

Even state-of-the art Active Queue Management (AQM) [41], [24] can only bring queuing delay down to roughly the same order as a typical base round-trip delay. This is because bottlenecks are typically in the most geographically dispersed edge access links where statistical flow multiplexing is lowest. And, without multiplexing, a TCP flow will underutilize a link unless it can buffer about a round trip flight of data.

**Our main contribution** is to keep queueing delay extremely low and consistently low for *all* of a user’s Internet applications. As will be seen later, this typically means sub-millisecond queueing at the 99%-ile, and an order of magnitude lower queuing than state-of-the-art AQMs at any percentile.

A differentiated service (Diffserv) class such as EF [16] can only provide low delay if it is limited to a fraction of a link’s capacity. Instead, the new service that we propose accommodates capacity-seeking applications that want both full link utilization and low queuing delay. Because low delay no longer involves a throughput sacrifice, it is expected that this new service will incrementally replace the default best efforts service.

The new service is called Low Latency, Low Loss, Scalable throughput (L4S), because it effectively removes congestion loss as well as queuing delay. L4S works because senders use one of the family of ‘Scalable’ congestion controls (see § II-A for definition and rationale). In contrast, we use the term ‘Classic’ for controls like TCP Reno and Cubic, where control becomes slacker as rate scales.

For evaluation we configure the host OS to use Data Centre TCP (DCTCP [2], [3]), which is a widely available scalable control. We emphasize that the L4S service is not just intended for DCTCP, but also for a range of Scalable controls, e.g. Relentless TCP [35] and scalable variants of QUIC, SCTP, real-time protocols, etc. In order to test one change at a time, we focus this paper on network-only changes, and use DCTCP, ‘as is’, except for the ECN codepoint it uses.

Our extensive experiments over a testbed using real data-centre and broadband access equipment and models of realistic traffic strengthen confidence that DCTCP would work very well over the public Internet. However, DCTCP will need some safety (and performance) enhancements for production use over the public Internet. In 2015, a large group of DCTCP developers informally agreed the ‘Prague L4S requirements’ (§ V-B). A variant of DCTCP called TCP Prague is being developed to satisfy them, which will avoid the otherwise

confusing name.

**Our second contribution** is a solution to the deployability of Scalable controls like DCTCP, in coexistence with other traffic on the public Internet.

It is a common misconception that DCTCP is tailored for data centres, but the name merely emphasizes that it should not be deployed outside a controlled environment; it is too aggressive to coexist with existing 'Classic' traffic. DCTCP is only applicable where a single admin can upgrade all senders, receivers and bottlenecks at once,... until now.

We propose the 'Dual Queue Coupled AQM' that can be incrementally added at path bottlenecks to solve this 'coexistence' problem. It acts like a semi-permeable membrane. For delay it uses a second queue to isolate L4S traffic from Classic. But for throughput, it couples the queues to appear as a single bandwidth pool. Briefly this means that the L4S queue emits congestion signals more aggressively to counterbalance the more aggressive response of L4S sources. Then, for $n$ aggressive L4S flows and $m$ TCP-friendly Classic flows, each flow takes roughly $1/(n+m)$ of the aggregate capacity.

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

Balance between microflows can also be achieved with per-flow queuing, but that involves potential compromises of privacy, neutrality, evolvability and complexity (§ II-C). The DualQ Coupled AQM side-steps all these dilemmas by not inspecting flows (no deeper than the IP layer) and by enabling but not enforcing coexistence.

L4S faces a very similar multi-party deployment problem to classic Explicit Congestion Notification (ECN [42]). However, we have learned from the ECN experience. To overcome the risk a first mover faces in kick-starting a multi-party deployment, we have attempted to ensure that the performance gain is dramatic enough to enable valuable new applications [8], not just a marginal performance improvement.

Given access networks are invariably designed to bottleneck in one known location, the AQM does not have to be deployed in every buffer. Most of the benefit can be gained by deployment at the downstream queue into the access link, and home gateway deployment addresses the upstream.

§ V 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 DCTCP and Classic TCP in heterogeneous or interconnected data centres.

**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 the L4S service can cope with a reasonable proportion of unresponsive traffic, just as best efforts copes with reasonable levels of unresponsive streaming, VoIP, DNS etc.

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

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

## II. Rationale

### A. Why a Scalable Congestion Control?

A congestion control is defined as Scalable if it satisfies an average and a dynamic condition: as packet rate per round trip scales i) the average rate of congestion signals per round trip does not decrease; and ii) the doubling time of the flow rate does not decrease [30]. This paper primarily focuses on the first condition, which concerns coexistence with other flows and interoperability with network signalling.

TCP Cubic scales better than Reno, but it is still not fully scalable. For instance, for every 8-fold rate increase the average Cubic sawtooth duration between losses or ECN marks doubles while its amplitude in bytes increases 8-fold. For instance, for an example base RTT=20 ms, between 100 and 800 Mb/s, Cubic's sawtooth recovery time expands from 250 to 500 round trips. In contrast, DCTCP averages one ECN mark every half a round trip, which remains invariant whatever the rate.

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

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

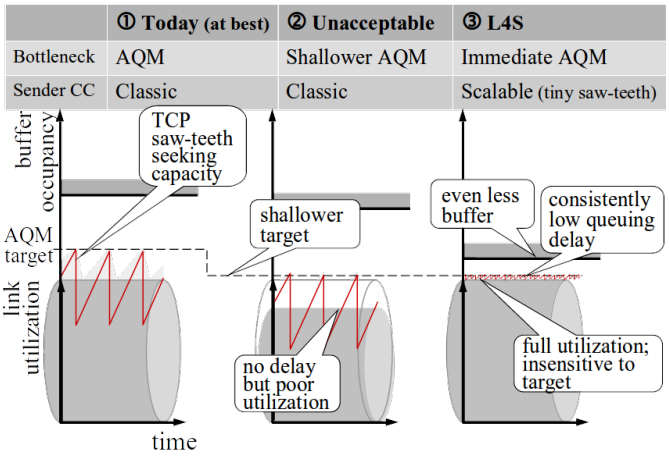



Fig. 1: Data Centre TCP: Intuition

In the steady state, the number of signals per round, $v$, is the product of segments per round $W$ and the probability $p$ that a segment carries a signal, i.e. $v = pW$. Formulas for the steady-state window, $W$, can be derived for each congestion controller (see § III-B). Each formula is of the form $W \propto 1/p^B$, where $B$ is a characteristic constant of the algorithm [5] (e.g. $B = 1/2$ for TCP Reno). So it is straightforward to state the above scalability condition in terms of $B$ by first substituting for $p$ in the above formula for $v$:

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

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

For DCTCP, $B \geq 1$, and DCTCP with probabilistic marking has $B = 1$ (see § III-B) so the steady-state signalling rate is scale-invariant.

The dynamic behaviour of DCTCP is not scalable [30] because it uses the same addition of one segment per RTT as Reno to increase its window. However, the L4S queue supports any control with a steady state that is scalable, so we can be confident that solutions to DCTCP's dynamic problems (see § V-B) will be able to evolve and co-exist with today's DCTCP, without a need for further network changes.

### B. Why ECN?

L4S uses the same protocol fields as Explicit Congestion Notification (ECN [42]), but defines new semantics for an ECN mark, breaking away from the standardized equivalence to loss [42], [7]. This allows L4S to break away from 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) ECN allows more frequent signalling, which would be untenable as loss, particularly during high load. This allows 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 it is just a sub-RTT burst, whereas it can emit ECN immediately, because it is harmless:
   - 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 cuasing 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 time constant [3] and it can choose to respond without smoothing, e.g. at flow start.
3) ECN also offers the obvious latency benefit of near-zero congestion loss, which is of most concern to short flows [43]. 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.

Because L4S requires Scalable traffic to be ECN-capable, it uses the ECN field to classify Scalable packets into the L4S queue (see § III).

### C. Why Not Per-Flow Queues?

Per-flow queuing (as in FQ-CoDel [24]) 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 TCP flow still inflicts upon *itself*.[1] This is important for the growing trend of interactive video-based apps that are both extremely latency-sensitive and capacity-hungry, e.g. interactive or conversational video, remote presence.

As we have explained, scalable congestion control is needed to address the self-inflicted delay problem. FQ can play a role in this by solving the coexistence problem; by enforcing throughput balance between Scalable and Classic flows. Indeed, an FQ classifier could be modified to detect that a flow is only using L4S-ECN packets and instantiate a shallow immediate ECN threshold.

However, per-flow scheduling involves compromises:

- It does not know whether i) 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.
- it needs to inspect transport layer headers, compromising transport evolution and privacy.
- It requires more complex classification, queuing and scheduling structures.

The DualQ Coupled AQM has been developed so that an operator can offer low delay service without having to swallow these compromises.

An operator could still enforce equal flow-rates, but that would be its own independent policy choice, not an inseparable side-effect of reducing delay.

## III. 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, specifically: how coexistence between L4S and Classic flows is achieved (§ III-B); how the low latency of the L4S service is isolated from Classic (§ III-C); how overload is handled (§ III-D); and an overview of the whole implementation in pseudocode (§ III-E).

### A. Solution Structure

L4S and Classic traffic have opposing delay requirements. The first design goal of L4S traffic is ultra-low queuing delay. In contrast Classic congestion controllers (CCs) need a significant queue to avoid under-utilization (at least they do whenever the number of flows at the bottleneck is small). 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 [42]. L4S

[1] 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 [6] or interactive video, need to maintain any self-induced send-queue locally in order to adapt how much they send. They cannot suppress lower priority data dependent on progress, if it is already in flight.

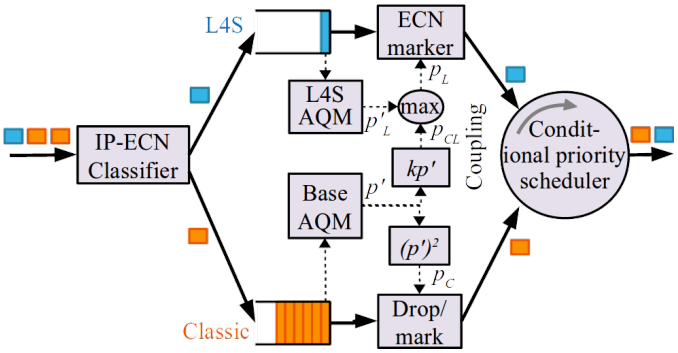



Fig. 2: Dual Queue Coupled AQM

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-A).

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

So the second design goal is coexistence between Classic and L4S congestion controllers [27], meaning rough balance between their steady-state packet rates per RTT (a.k.a. window-fairness or TCP-friendliness). Therefore, we couple the congestion signals of the two queues and reduce the signal intensity for Classic traffic to compensate for its stronger response to each signal, in a similar way to the single-queue coupled AQM in [17].

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

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

### B. Coupled AQM for Window Balance

To support co-existence between the Classic (C) and L4S (L) congestion control families, we start with the equations that characterize the steady-state window, $W$, of each as a function of the loss or ECN-marking probability $p$. Then, like [17], we set the windows to be equal to derive the coupling relationship between the congestion signals for C and L.

We use Reno and DCTCP for C and L. We use Reno because it is the worst case (weakest). We can ignore dynamics, so we use the simplified Reno equation from [36]. For L4S, we do not use the equation from the DCTCP paper [2], which is only appropriate for step marking. Instead, we use the DCTCP equation that is appropriate to our coupled AQM, where marking is probabilistic, as derived in Appendix A of [17]. For balance between the windows, $W_{\mathrm{reno}} = W_{\mathrm{dc}}$, which becomes (1) by substituting from each window equation. Then we rearrange into a generalized relationship for coupling congestion signals in the network (2):

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

where $p_{CL}$ is the signal coupled from C to L and coupling factor $k = 2\sqrt{2/3} = 1.64$ for Reno.

Appendix A of [17] shows that TCP Cubic [23] will be comfortably within its Reno compatibility mode for the 'Data Centre to the Home' scenarios that are the focus of this paper. The coupling formula in (2) also applies when the Classic traffic is TCP Cubic in Reno mode ('CReno'), except it should use $k = 2/1.68 = 1.19$.

To avoid floating point division in the kernel we round to $k = 2$. In all our experiments this proves to be a sufficiently accurate compromise for any Reno-friendly CC. It gives a slight window advantage to Reno, and a little more to CReno. However, any L4S source gives itself a counter-advantage by virtue of its shallower queue. So L4S achieves a higher packet rate with the same window because of it lower RTT. We do not expend effort countering this rate imbalance in the network—the proper place to address this is to ensure L4S sources will be less RTT-dependent (see § V-B).

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

Diversity of Base AQMs is possible and encouraged. Two have been implemented and tested [18]: a variant of RED and a proportional integral (PI) AQM. Both control queuing time not queue size, given the rate of each queue varies considerably [34], [40]. This paper uses the latter, because it performs better.

[17] also couples two AQMs to enable coexistence of different CCs, but within one queue, so there is not delay isolation. It proves theoretically and experimentally that a PI controller is a robust base AQM. It directly controls a scalable control like DCTCP (rate proportional to $1/p'$). And it shows that squaring the output of a PI controller is a more effective, more principled and simpler way of controlling TCP Reno (rate proportional to $1/\sqrt{p'}$) than PI Enhanced (PIE [41]). It shows that the piecewise lookup table of scaling values used by PIE was just a heuristic way of achieving the same effect as squaring.

### C. Dual Queue for Low Latency

Often, there will only be traffic in one queue, so each queue needs its own native AQM. The L4S queue keeps delay low using a shallow marking threshold ($T$), which has already been proven for DCTCP. Because the dequeue rate varies

considerably, $T$ is set in units of time [34], [4] with a floor of two packets. The queue to compare against the threshold is also measured in time units. On-off marking may [14] or may not [33, §5] be prone to instability. But to test one change at a time we deferred investigation of this to future research.

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

Note that the L4S AQM emits ECN marks immediately and the sender is expected to do any necessary smoothing. Whereas the Classic AQM smooths its output, which introduces delay, then Classic sources respond without delay.

To decide between the head packets of the two queues, we have found that a weighted round robin (WRR) scheduler with a high weight in favour of the L queue (e.g. 15/16) works well. The actual weight is not critical, because classic traffic gets L4S traffic to make space for it via the coupling, not the scheduler. It is only necessary to ensure Classic packets cannot be starved by unresponsive L4S traffic or long L4S bursts. This also ensures that a new Classic flow can break into a standing L4S queue.

We have also tried what we call a Time-Shifted FIFO scheduler [37]. 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 tends to allow long 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 lower priority traffic than a single queue would.

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

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

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

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

### E. Linux qdisc Implementation

**Algorithm 1** Enqueue for Dual Queue Coupled AQM

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

**Algorithm 2** Dequeue for Dual Queue Coupled AQM

```
1:  while LQ.LEN() + CQ.LEN() >0 do
2:      if 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
```

Algorithms 1 & 2 summarize the per packet enqueue and dequeue implementations of DualPI2 as pseudocode. The AQMs are applied at dequeue to minimize signalling delay. For clarity, overload and saturation logic are omitted, but they can be found in the full open-sourced implementation of the DualPI2 Linux qdisc [1].

On enqueue, packets are time-stamped and classified. The function LEN() returns the the queue in bytes, while TIME() returns the duration since a packet was time-stamped (its sojourn or service time).

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 scheduler.

If an L4S packet is scheduled, line 4 runs the native L4S AQM to output probability $p'_L$ dependent on the delay 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 and the coupling.

If a Classic packet is scheduled, line 10 decides whether to emit a congestion signal with probability $p_C$. 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 internal base signalling probability ($p'$) is kept up to date by the core PI Algorithm (3) which only needs occasional execution [26]. The change in queuing time is multiplied by the proportional gain factor $\beta$. The integral gain factor $\alpha$ is

typically smaller, to restore any persistent standing queue to the target delay. These expressions, which can be negative, are added to the previous $p'$ every $T_{\text{update}}$ (default 16 ms). Then the Coupled and Classic signalling probabilities, $p_{CL}$ and $p_C$ are derived from $p'$.

**Algorithm 3** PI core: Every $T_{\text{update}}$ $p$ is updated

```
curq = CQ.TIME()
p' = p' + α * (curq − TARGET) + β * (curq − prevq)
p_CL = k * p'
p_C = (p')^2
prevq = curq
```

## IV. Evaluation

### A. Testbed Setup

We used a testbed to evaluate the proposed DualPI2 AQM mechanism in a realistic setting, and to run repeatable experiments in a controlled environment. The testbed was assembled using carrier grade equipment in the same enviroment as for testing customer solutions. Figure 11 depicts the testbed, which consists of a classical residential service delivery network composed of Residential Gateway, xDSL DSLAM (DSL Access Multiplexer), BNG (Broadband Network Gateway), Service Routers (SR) and application servers. The Residential Gateway is connected by VDSL to a DSLAM, which is connected to the BNG through an aggregation network, representing a local ISP or access wholesaler. Traffic is routed to another network representing a global ISP that hosts the application servers and offers breakout to the Internet. The client computers in the home network and the application servers at the global ISP are Linux machines, which can be configured to use any TCP variant, start applications and test traffic. The two client-server pairs (A and B) are respectively configured with the same TCP variants and applications.

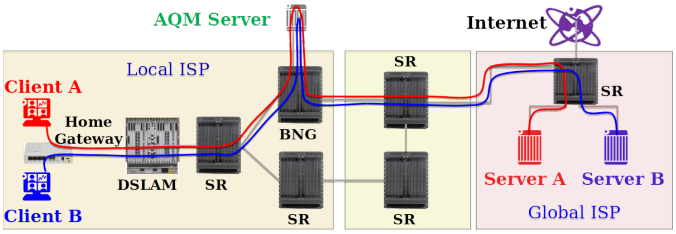



Fig. 11: Testbed configuration

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

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

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

We used DCTCP for the Scalable congestion control and Cubic for Classic. Cubic was used with default configuration, while for DCTCP, we used the implementation delivered with Linux kernel 3.19. We used an older version of DCTCP to avoid bugs introduced in later versions. Since this version of DCTCP did not correctly respond to drops, we patched it to align its drop response with Reno. We also modeified it to use the ECT(1) codepoint. For ECN-Cubic, we enabled TCP ECN negotiation. We compared DualPI2 with PIE and FQ-CoDel, all configured as in Table I. The $\alpha$: and $\beta$ values for PIE are equivalent to those used for DualPI2, but PIE scales the input parameters internally.

| | |
|---|---|
| All | Buffer: 40,000 pkt, ECN enabled |
| PIE | Target delay: 15 ms, Burst: 100 ms, TUpdate: 16 ms, $\alpha$: 1/16, $\beta$: 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, WRR C weight: 10%, $\alpha$: 0.16, $\beta$: 3.2, k: 2, Classic ECN_drop: 25% |

TABLE I: Default parameters for the different AQMs.

### B. Experimental Approach

For traffic load we used long-running flows (§§ IV-C & IV-D) and/or dynamic short flows (§ IV-C). We used long flows, not as an example of a realistic Internet traffic mix, rather to aid interpretation of various effects, such as starvation.

The set of experiments was constructed to evaluate our two main performance goals: queuing delay and window balance. We also show rate balance, link utilization and drop/mark probability, as well as flow completion times in short flow experiments. More details about which metrics we use for the evaluation are presented in § IV-F. Heavy load scenarios predominate in our setup of experiments, again not because they are typical, but because they do occur and they are the worst case.

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

In this paper we present those combinations of CC and AQM that each AQM is intended to support: DCTCP with Cubic on DualPI2 and FQ-CoDel and ECN-Cubic with Cubic on PIE.

We configured PIE with ECN and used a modified version of FQ-CoDel, where L4S support was added byusing a shallow ECN marking threshold for any ECT(1) packet, with an additional check to ensure that the threshold is only applied if there is more than 1 packet in the queue. The queue length check was added to prevent 100% marking at lower link rates, considering that packet serialisation in such cases takes longer.

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.

### C. Experiments with equal RTT

We performed a set of experiments to evaluate the performance of the AQMs in a scenario where all competing flows in each experiment had equal RTTs. We used 4 scenarios: 1) 1-1: 1 long running flow for each CC; 2) 1h:1h: 1 long running flow with added dynamic flows (high load) for each CC; 3) different combinations of flows for each CC in the range of 0-10; 4) 1h-1h + 1l-1l: 1 long running flow for each CC with high and low intensity of dynamic short flows respectively.

The first two scenarios were evaluated with 25 combinations of 5 RTTs (5, 10, 20, 50 and 100 ms) and 5 link speeds (4, 12, 40, 120 and 200 Mbps).

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

To differentiate between experiments with low and high dynamic traffic load, we denote them as '1h-1h' and '1l-1l' cases respectively, where '1' stands for one long running flow, 'h' for high and 'l' for low web traffic intensity.

For experiments with different flow combinations (third scenario, presented in Figure 6), the X-axis shows how many A and B flows were competing in each combination, while the legend states which congestion control was used for A and B flows.

### D. Experiments with different RTTs

To evaluate the RTT-dependence of the windows and rates of different CCs, we conducted additional experiments with one flow per client server pair, each having different base RTTs. In addition, we added experiments where ECN-capable flows with shortest and longest RTTs were running alone. We used different combinations of the same RTTs we used in equal RTT experiments (5, 10, 20, 50 and 100 ms). These experiments were repeated for the 5 link speeds, but in this paper, we only show results for 40Mbps link speed and 10ms RTT as a representative example. In Figure 8 and Figure 9, the X-axis denotes the RTT used for each of the two flows (A and B), while the legend states which congestion control was used for each of the A and B flows respectively.

### E. Overload experiments

To validate the correct overload behaviour, we added an unresponsive UDP flow to 5 long-running flows of each congestion control type (ECN and non-ECN) over a 100 Mbps bottleneck link with 10 ms base RTT. For each AQM we ran 2 sets of tests with the UDP traffic marked as either ECN/L4S or non-ECN. Each set tested 5 different UDP rates (50, 70, 100, 140 and 200 Mbps. Both UDP class (ECN/L4S or non-ECN) and UDP rate are shown on X-axis in Figure 10)

### F. Evaluation metrics

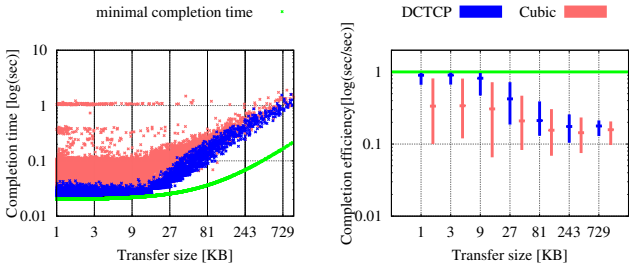



Fig. 12: Completion time against efficiency representation for 1 long flow and high dynamic load each on a 40 Mbps link with 10 ms base RTT.

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

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, $25^{th}$ and $99^{th}$ percentiles.

To evaluate the stability of rate and window, we measured throughput for each long-running flow by capturing all traffic at the AQM node. Since some experiments had 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. All rate and window measurements were sampled per second, and all samples were further processed to derive mean, $1^{st}$ and $99^{th}$ percentiles. Rate and window balance ratio was calculated by diving average rate/window per flow of ECN-capable CC by average rate/window per flow of Classic CC for the entire experiment.

Utilisation 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, $1^{st}$ and $99^{th}$ 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.

The left-hand side of Figure 12 shows a log-log scatter plot of the completion time to item size relation for the high load DualPI2 AQM test case on a 40 Mbps link with 10 ms base RTT. The green line is the theoretically achievable completion time, taking the RTT into account but downloading at full link speed from the start.

To better quantify the average and percentiles of the completion times, we used the Completion Efficiency representation on the right of Figure 12. To calculate its completion efficiency for each item we divided actual by theoretical completion time. We then binned the samples in log scale bins (base 3) and calculated the average, $1^{\text{st}}$ and $99^{\text{th}}$ percentiles. The green theoretical completion time is now at 1 (maximum efficiency).

### G. Results

{*Editorial note: Originally, this paper contained a selection of plots to fit a page limit. Instead, the full set of results has been included at the end. The commentary explaining the selected results has been suppressed to avoid confusion.*}

## V. Deployment Considerations

### A. Standardization Requirements

The IETF has taken on L4S standardization work [13]. [19] considers the pros and cons of various candidate identifiers for L4S and finds that none are without problems, but proposes ECT(1) as the least worst. 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 [7].

The main issue is that there is only one spare codepoint. So, if it is used to distinguish L and C packets, congestion marking has to use the same Congestion Experienced (CE) codepoint for both L & C packets. This is not a problem for hosts. However, in the (unusual) case of multiple ECN bottlenecks along one path, any packet already marked CE upstream will have to be classified into the L queue, irrespective of whether it was originally C or L. This could lead to an unusual form of reordering where a few packets arrive early. The IETF considers this acceptable because subsequent packets still advance the ACK counter [19] so TCP normally handles it without any spurious retransmissions—except in the unlikely case where a Classic ECN flow experiences a solid run of CE marks longer than the reordering window.

The IETF has defined the semantics of the new identifier. The 'Classic' ECN standard [42] defines a CE mark as equivalent to a drop, so queuing delay with Classic ECN cannot be better than with drop (this may be why operators have not deployed Classic ECN [45, §5]). The square relationship between an L4S mark and a drop in this paper (Eqn. (2)) has been proposed for experimental standardization [19]. It has been proposed to recommend rather than standardize a value for the coupling factor, $k$, given differences would not prevent interoperability.

The IETF has also specified how operators could classify L4S on additional identifiers as well as ECN (e.g. address range or VLAN ID), which they might use for initial exclusivity, without compromising long-term interoperability.

The IETF has also adopted a specification of the dualQ coupled AQM mechanism [18] so that multiple implementations can be built, tested and compared, possibly using different base AQMs internally.

### B. Congestion Control Roadmap

This paper uses DCTCP unmodified (except for the codepoint)in all experiments i) to focus the parameter space of our experiments on the network mechanism, without which end-system performance improvements would be moot; and ii) to emphasize that the end-system side of the multi-party deployment is already available (in windows, Linux and FreeBSD), at least for testing purposes. Nonetheless, numerous improvements to DCTCP can be envisaged for this new public Internet scenario. They are listed below in priority order starting with those necessary for safety, and ending with performance improvements. They are a summary of the L4S transport layer behaviours identified by the IETF [19], which are in turn are adapted from the "Prague L4S requirements", named after the meeting in Prague of a large group of DCTCP developers that informally agreed them [9]. Since, a variant of DCTCP called TCP Prague has been implemented for Linux [11] to address them:

1) Fall back to Reno/Cubic on loss;
2) Negotiate altered feedback semantics [31], [12];
3) Use of a standardized packet identifier [19];
4) Handle a window of less than 2, rather than grow the queue if base RTT is low [11, §3.1.6];
5) Smooth ECN feedback over a timescale measured in the flow's own round trips [3, §5];
6) Fall back to Reno/Cubic if increased delay of classic ECN bottleneck detected;
7) Faster-than-additive increase, e.g. Adaptive Acceleration ($A^2$DTCP) [48] or Paced Chirping [38];
8) Less drastic exit from slow-start, similar goal to Flow-Aware (FA-DCTCP) [28] or Paced Chirping [39];
9) Reduce RTT-dependence of rate [3, §5] (see below).

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

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

[2]Note that access buffers cannot exploit such high flow aggregation as in the core [21]

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

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

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

### C. Deployment Scenarios

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

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

## VI. Related Work

In 2002, Gibbens and Kelly [22] 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 [33, §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 [20].

Wu et al. [46] investigates a way to incrementally deploy DCTCP within data centres, marking ECN when the temporal queue exceeds a shallow threshold but using standard ECN [42] on end-systems. Kuhlewind et al. [32] showed that DCTCP and Reno could co-exist in the same queue configured with a form of WRED [15] classifying on ECN rather than Diffserv. Judd [29] 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) [47] is an AQM for DC switches building on CoDel [40]. Unlike the DualQ, FaLL inspects the transport layer of sample packets to focus more marking onto faster flows while keeping the queue short.

## VII. Conclusion

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

This paper moves on from debating where the 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 kick out their old TCP balloon.

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

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

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

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

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

## 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., 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.

[3] ALIZADEH, M., JAVANMARD, A., AND PRABHAKAR, B. Analysis of DCTCP: Stability, Convergence, and Fairness. *Proc. ACM SIGMETRICS'11* (2011).

[4] 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.

[5] BANSAL, D., AND BALAKRISHNAN, H. Binomial Congestion Control Algorithms. In *Proc. IEEE Conference on Computer Communications (Infocom'01)* (Apr. 2001), IEEE, pp. 631–640.

[6] BELSHE, M., PEON, R., AND THOMSON (ED.), M. Hypertext Transfer Protocol version 2 (HTTP/2). Request for Comments 7540, RFC Editor, May 2015.

[7] BLACK, D. Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation. Request for Comments RFC8311, RFC Editor, Jan. 2018.

[8] BONDARENKO, O., DE SCHEPPER, K., TSANG, I.-J., BRISCOE, B., PETLUND, A., AND GRIWODZ, C. Ultra-Low Delay for All: Live Experience, Live Analysis. In *Proc. ACM Multimedia Systems; Demo Session* (New York, NY, USA, May 2016), ACM, pp. 33:1–33:4.

[9] 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.

[10] 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.

[11] 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).

[12] BRISCOE, B., KÜHLEWIND, M., AND SCHEFFENEGGER, R. More Accurate ECN Feedback in TCP. Internet Draft draft-ietf-tcpm-accurate-ecn-08, Internet Engineering Task Force, Mar. 2019. (Work in Progress).

[13] BRISCOE (ED.), B., DE SCHEPPER, K., AND BAGNULO, M. Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture. Internet Draft draft-ietf-tsvwg-l4s-arch-03, Internet Engineering Task Force, Oct. 2018. (Work in Progress).

[14] CHEN, W., CHENG, P., REN, F., SHU, R., AND LIN, C. Ease the Queue Oscillation: Analysis and Enhancement of DCTCP. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on* (July 2013), pp. 450–459.

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

[16] DAVIE, B., ET AL. An Expedited Forwarding PHB (Per-Hop Behavior). Request for Comments 3246, Internet Engineering Task Force, Mar. 2002.

[17] 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.

[18] DE SCHEPPER, K., BRISCOE (ED.), B., ALBISSER, O., AND TSANG, I.-J. DualQ Coupled AQM for Low Latency, Low Loss and Scalable Throughput (L4S). Internet Draft draft-ietf-tsvwg-aqm-dualq-coupled-08, Internet Engineering Task Force, Nov. 2018. (Work in Progress).

[19] DE SCHEPPER, K., BRISCOE (ED.), B., AND TSANG, I.-J. Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S). Internet Draft draft-ietf-tsvwg-ecn-l4s-id-06, Internet Engineering Task Force, Mar. 2019. (Work in Progress).

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

[21] GANJALI, Y., AND MCKEOWN, N. Update on Buffer Sizing in Internet Routers. *ACM SIGCOMM Computer Communication Review 36* (Oct. 2006).

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

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

[24] HOEILAND-JOERGENSEN, T., MCKENNEY, P., TÄHT, D., GETTYS, J., AND DUMAZET, E. The FlowQueue-CoDel Packet Scheduler and Active Queue Management Algorithm. Request for Comments RFC8290, RFC Editor, Jan. 2018.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] KELLY, T. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *ACM SIGCOMM Computer Communication Review 32*, 2 (Apr. 2003).

[31] KÜHLEWIND, M., SCHEFFENEGGER, R., AND BRISCOE, B. Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback. Request for Comments 7560, RFC Editor, Aug. 2015.

[32] 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.

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

[34] 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.

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

[36] 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).

[37] 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.

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

[39] MISUND, J., AND BRISCOE, B. Paced Chirping: Rapid flow start with very low queuing delay. In *Proc. IEEE Global Internet Symp.* (May 2019), IEEE.

[40] NICHOLS, K., AND JACOBSON, V. Controlling Queue Delay. *ACM Queue 10*, 5 (May 2012).

[41] 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.

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

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

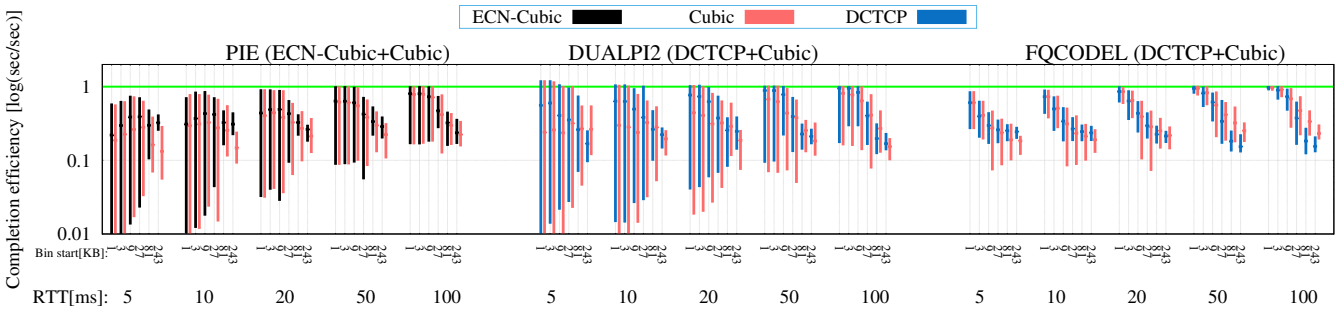
[44] STEEN, H. Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management. Masters thesis, Department of Informatics, University of Oslo, Spring 2017.

[45] WELZL, M., AND FAIRHURST, G. The Benefits of using Explicit Congestion Notification (ECN). Request for Comments RFC8087, RFC Editor, Mar. 2017.

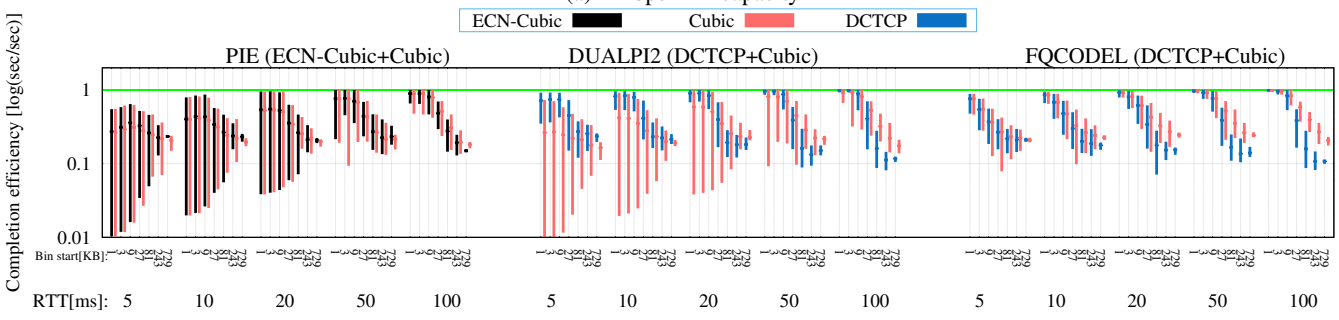
[46] WU, H., JU, J., LU, G., GUO, C., XIONG, Y., AND ZHANG, Y. Tuning ECN for Data Center Networks. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2012), CoNEXT '12, ACM, pp. 25–36.

[47] 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.

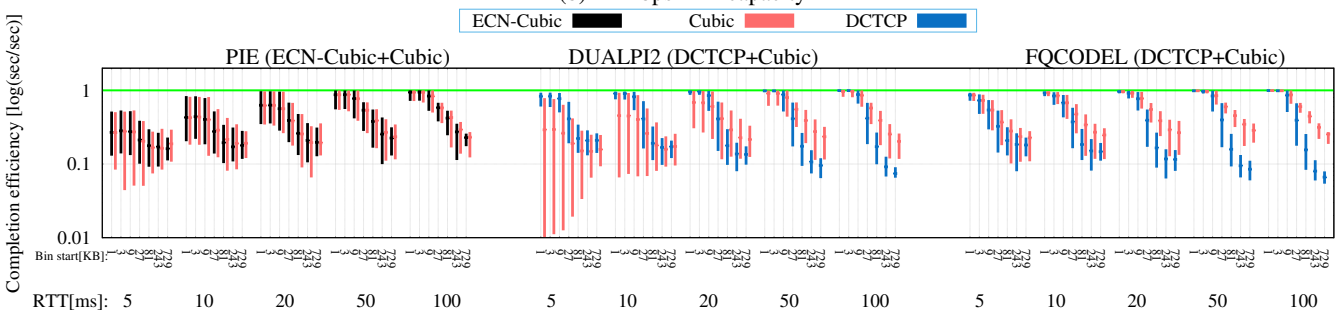
[48] ZHANG, T., WANG, J., HUANG, J., HUANG, Y., CHEN, J., AND PAN, Y. Adaptive-Acceleration Data Center TCP. *IEEE Transactions on Computers 64*, 6 (June 2015), 1522–1533.



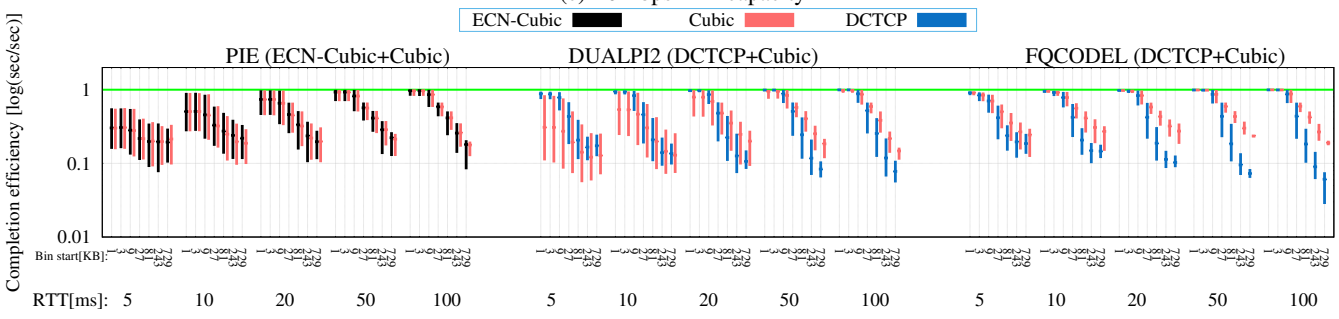

(a) 4Mbps link capacity



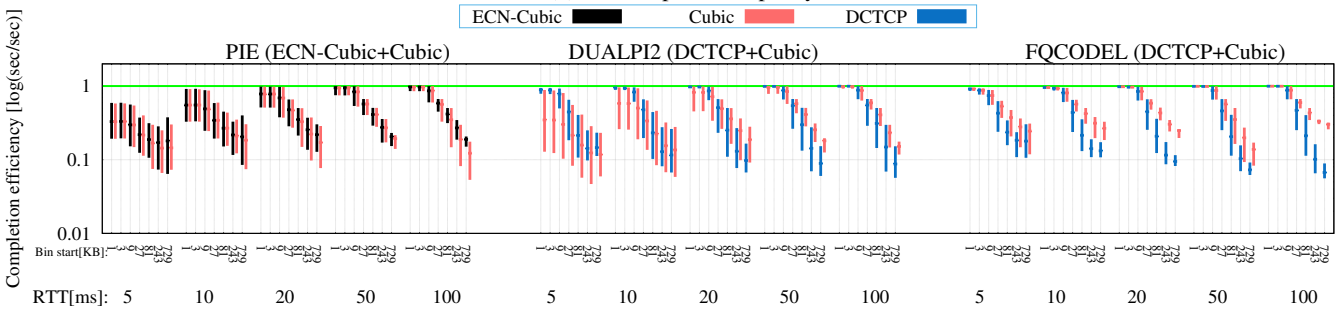

(b) 12Mbps link capacity




(c) 40Mbps link capacity




(d) 120 Mbps link capacity




(e) 200 Mbps link capacity

Fig. 3: Equal RTT (1h-1h)

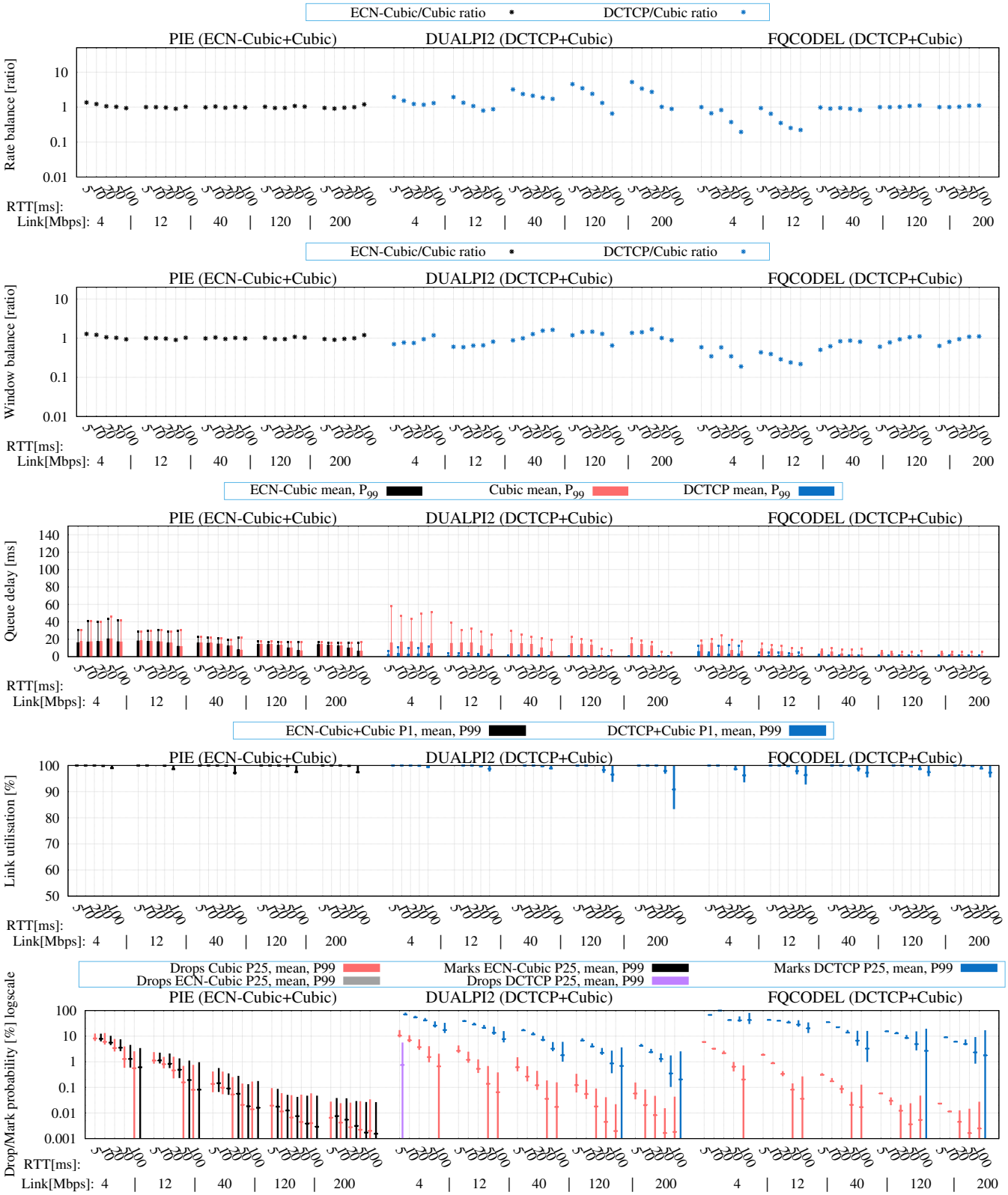



Fig. 4: Equal RTT (1-1)

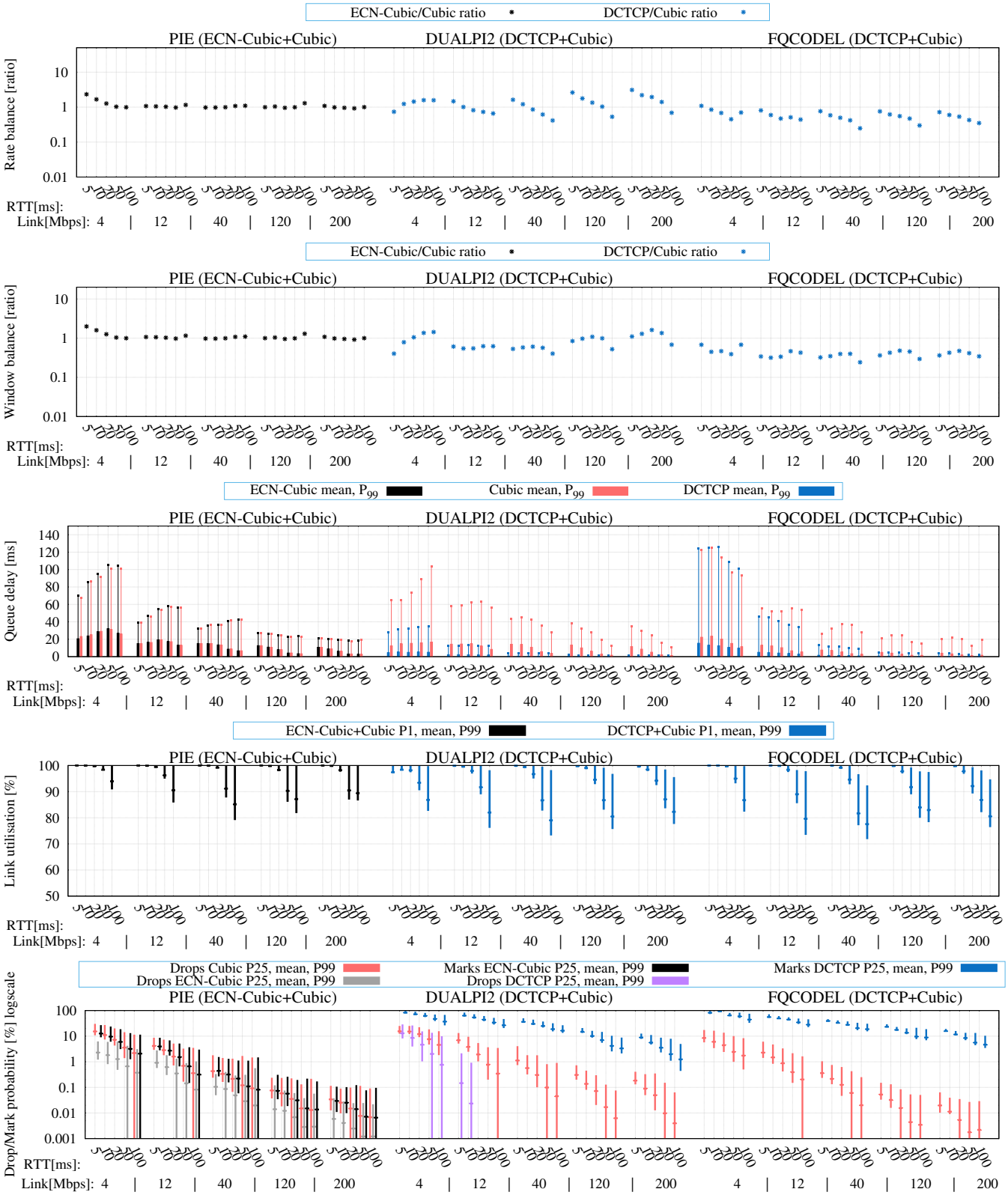



Fig. 5: Equal RTT (1h-1h)

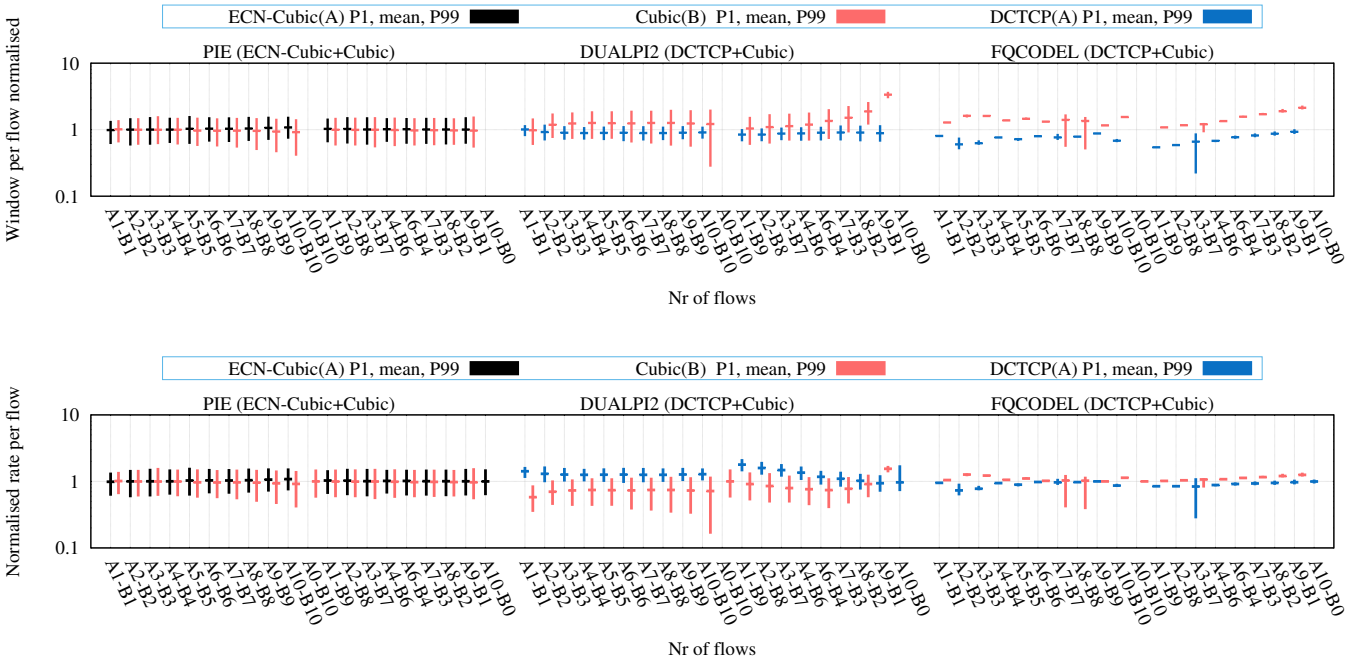



Fig. 6: Normalised rate and window size per flow. 40Mbps link capacity, 10ms RTT. Equal RTT

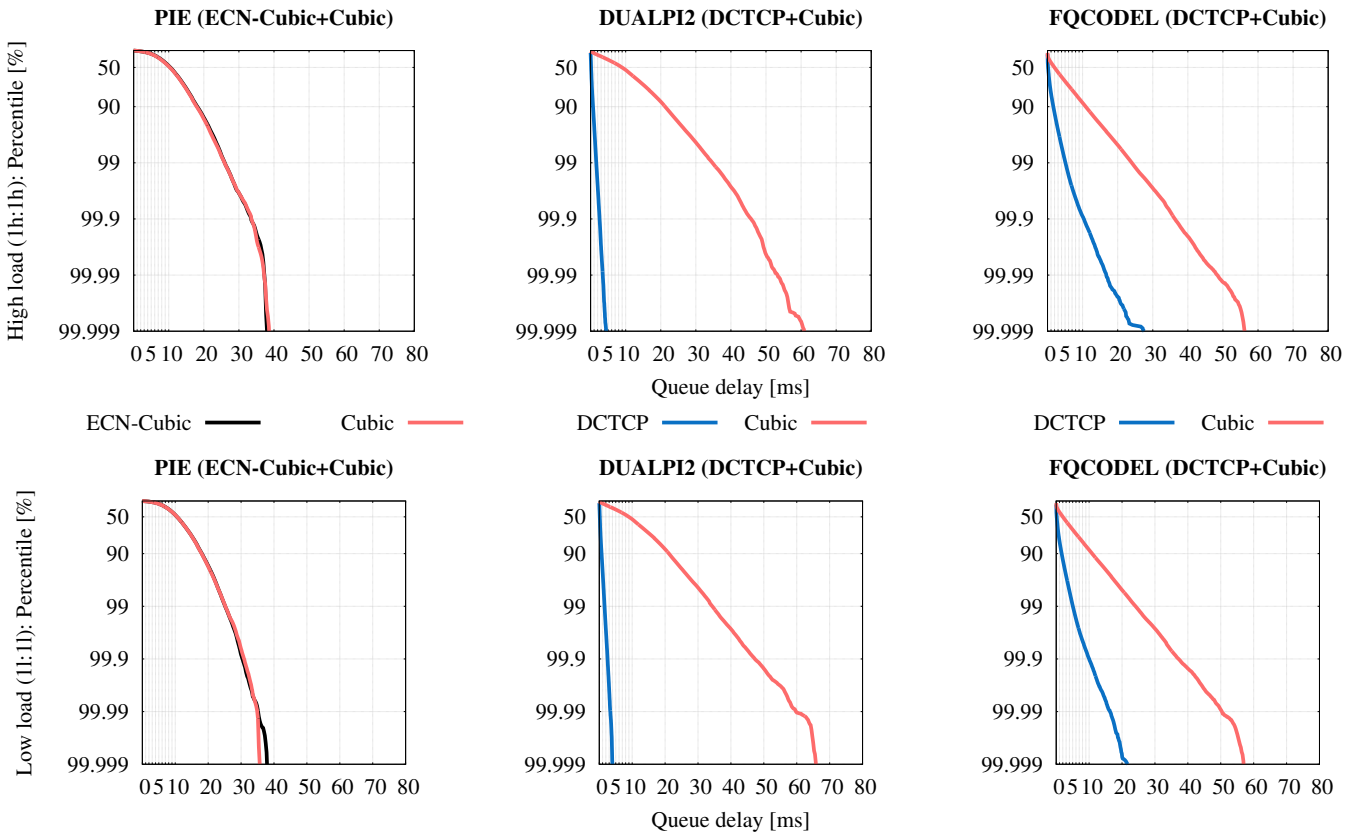



Fig. 7: Queue delay percentiles. 120Mbps link capacity, 10ms RTT. Equal RTT

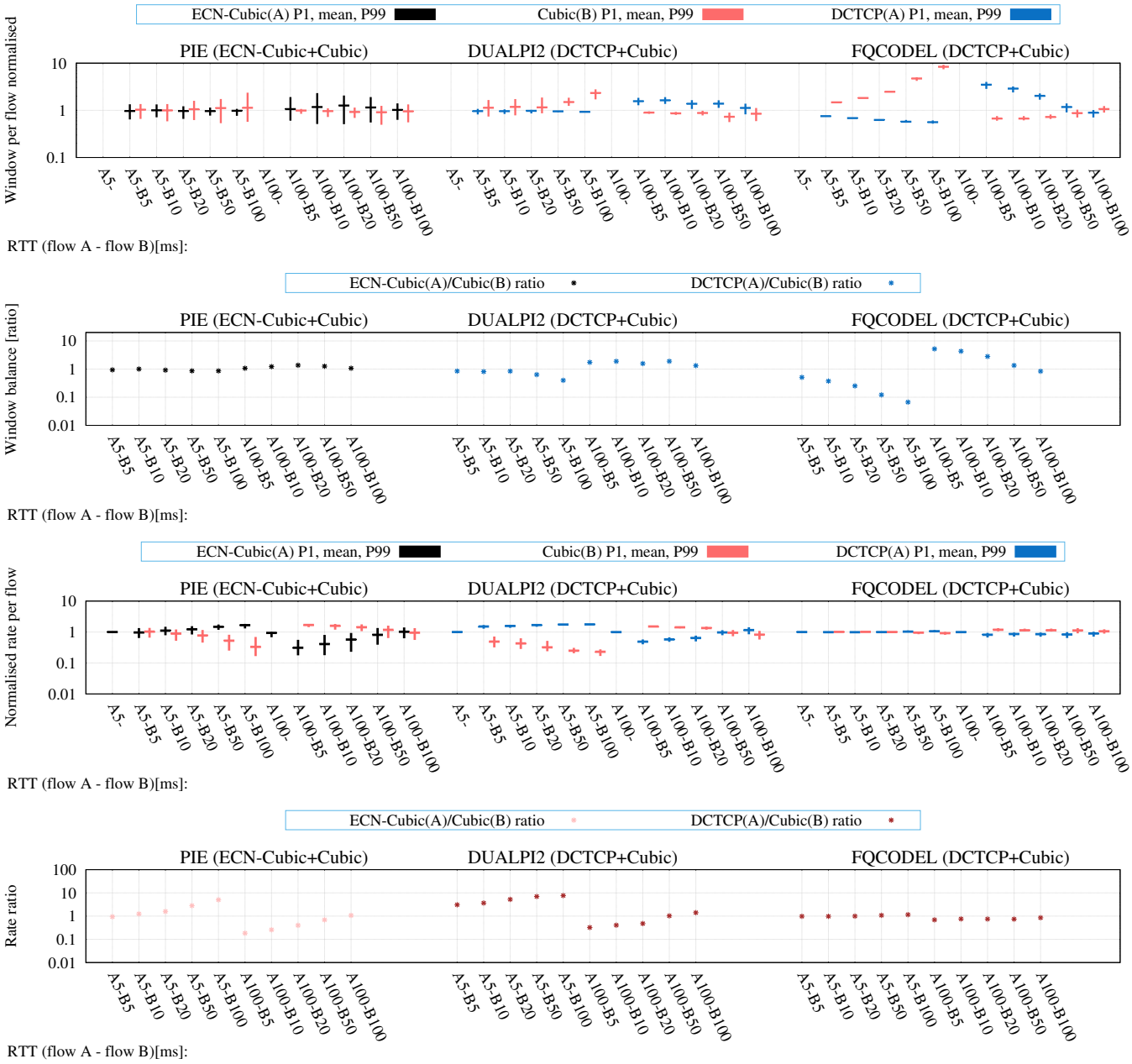



Fig. 8: 1 flow for each CC. Mixed RTT

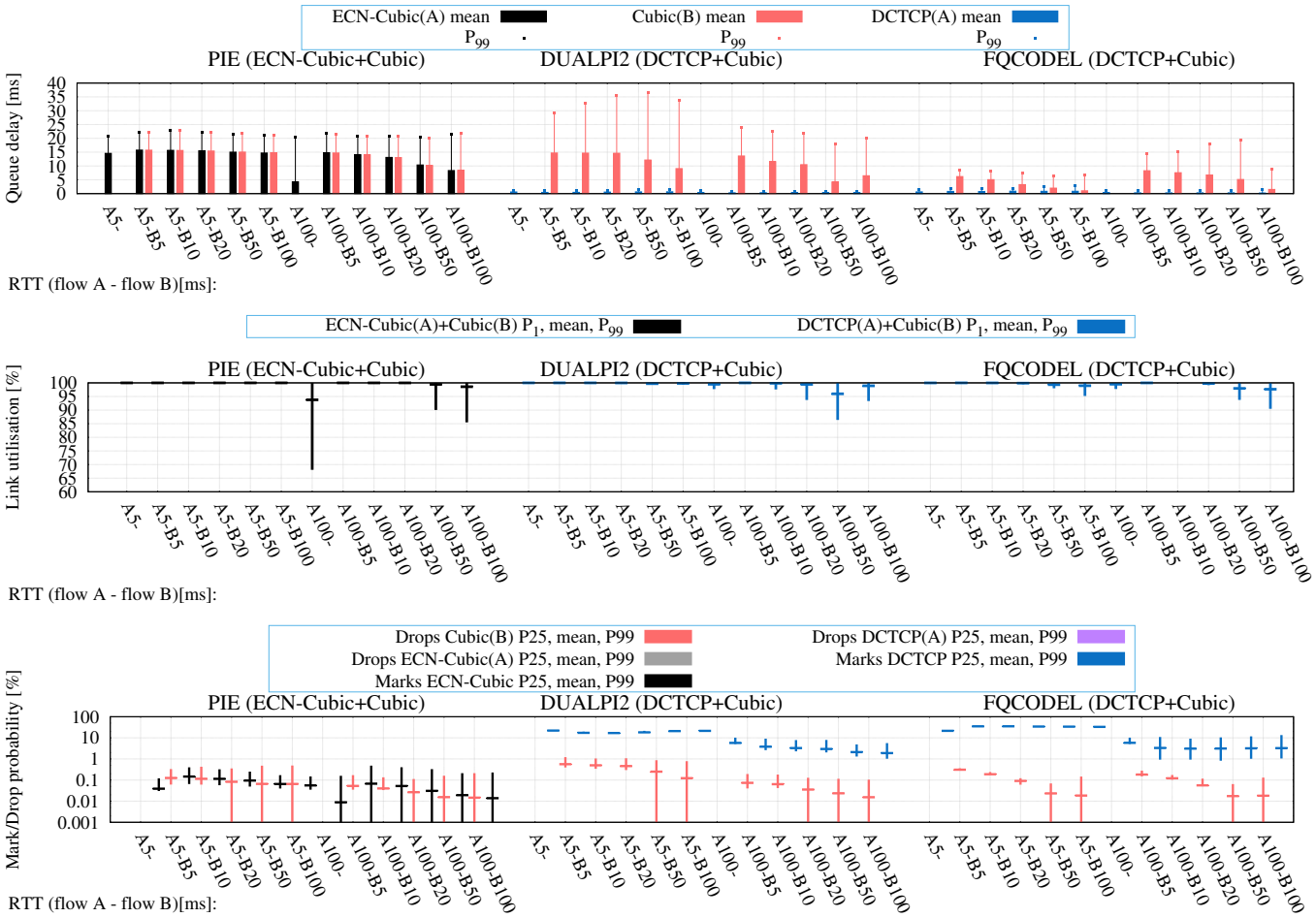



Fig. 9: 1 flow for each CC. Mixed RTT

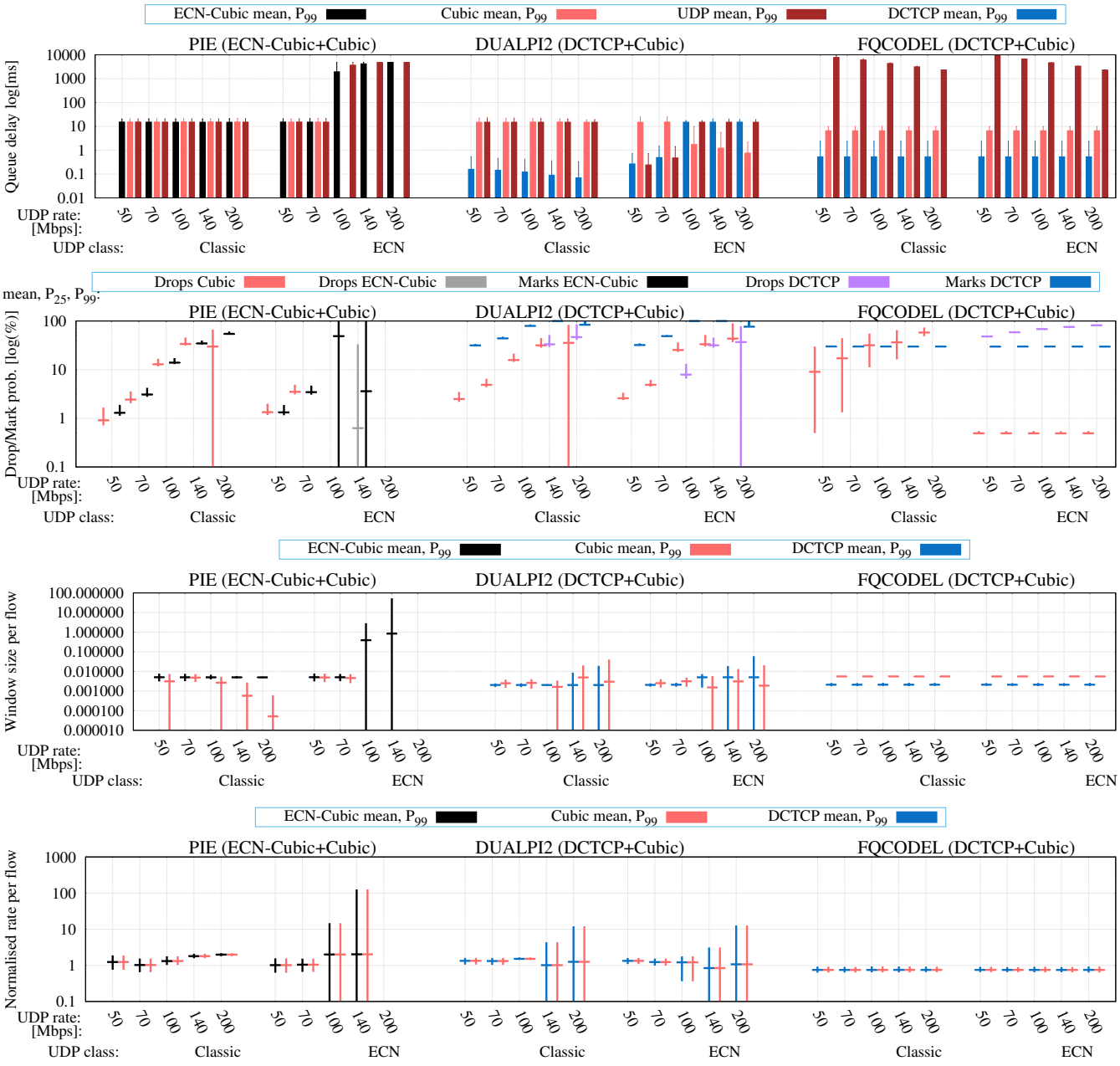



Fig. 10: Overload experiments. 1 flow for each CC