Workgroup: Transport Area Working Group
Internet-Draft: draft-ietf-tsvwg-l4s-arch-20
Published: 29 August 2022
Intended Status: Informational
Expires: 2 March 2023
Authors: B. Briscoe, Ed. K. De Schepper
Independent Nokia Bell Labs
M. Bagnulo Braun G. White
Universidad Carlos III de Madrid CableLabs

# Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture

## Abstract

This document describes the L4S architecture, which enables Internet applications to achieve Low queuing Latency, Low Loss, and Scalable throughput (L4S). L4S is based on the insight that the root cause of queuing delay is in the capacity-seeking congestion controllers of senders, not in the queue itself. With the L4S architecture all Internet applications could (but do not have to) transition away from congestion control algorithms that cause substantial queuing delay, to a new class of congestion controls that can seek capacity with very little queuing. These are aided by a modified form of explicit congestion notification (ECN) from the network. With this new architecture, applications can have both low latency and high throughput.

The architecture primarily concerns incremental deployment. It defines mechanisms that allow the new class of L4S congestion controls to coexist with 'Classic' congestion controls in a shared network. The aim is for L4S latency and throughput to be usually much better (and rarely worse), while typically not impacting Classic performance.

## Status of This Memo

This Internet-Draft will expire on 2 March 2023.

## Copyright Notice



## Table of Contents

## 1. Introduction

At any one time, it is increasingly common for all of the traffic in a bottleneck link (e.g. a household's Internet access) to come from applications that prefer low delay: interactive Web, Web services, voice, conversational video, interactive video, interactive remote presence, instant messaging, online gaming, remote desktop, cloud-based applications and video-assisted remote control of machinery and industrial processes. In the last decade or so, much has been done to reduce propagation delay by placing caches or servers closer to users. However, queuing remains a major, albeit intermittent, component of latency. For instance spikes of hundreds of milliseconds are not uncommon, even with state-of-the-art active queue management (AQM) [COBALT], [DOCSIS3AQM]. Queuing in access network bottlenecks is typically configured to cause overall network delay to roughly double during a long-running flow, relative to expected base (unloaded) path delay [BufferSize]. Low loss is also important because, for interactive applications, losses translate into even longer retransmission delays.

It has been demonstrated that, once access network bit rates reach levels now common in the developed world, increasing link capacity offers diminishing returns if latency (delay) is not addressed [Dukkipati06], [Rajiullah15]. Therefore, the goal is an Internet service with very Low queueing Latency, very Low Loss and Scalable throughput (L4S). Very low queuing latency means less than 1 millisecond (ms) on average and less than about 2 ms at the 99th percentile. End-to-end delay above 50 ms [Raaen14] or even above 20 ms [NASA04] starts to feel unnatural for more demanding interactive applications. So removing unnecessary delay variability increases the reach of these applications (the distance over which they are comfortable to use). This document describes the L4S architecture for achieving these goals.

Differentiated services (Diffserv) offers Expedited Forwarding (EF [RFC3246]) for some packets at the expense of others, but this makes no difference when all (or most) of the traffic at a bottleneck at any one time requires low latency. In contrast, L4S still works well when all traffic is L4S - a service that gives without taking needs none of the configuration or management baggage (traffic policing, traffic contracts) associated with favouring some traffic flows over others.

Queuing delay degrades performance intermittently [Hohlfeld14]. It occurs when a large enough capacity-seeking (e.g. TCP) flow is running alongside the user's traffic in the bottleneck link, which is typically in the access network. Or when the low latency application is itself a large capacity-seeking or adaptive rate (e.g. interactive video) flow. At these times, the performance improvement from L4S must be sufficient that network operators will be motivated to deploy it.

Active Queue Management (AQM) is part of the solution to queuing under load. AQM improves performance for all traffic, but there is a limit to how much queuing delay can be reduced by solely changing the network; without addressing the root of the problem.

The root of the problem is the presence of standard congestion control (Reno [RFC5681]) or compatible variants (e.g. CUBIC [RFC8312]) that are used in TCP and in other transports such as QUIC [RFC9000]. We shall use the term 'Classic' for these Reno-friendly congestion controls. Classic congestion controls induce relatively large saw-tooth-shaped excursions up the queue and down again, which have been growing as flow rate scales [RFC3649]. So if a network operator naively attempts to reduce queuing delay by configuring an AQM to operate at a shallower queue, a Classic congestion control will significantly underutilize the link at the bottom of every saw-tooth.

It has been demonstrated that if the sending host replaces a Classic congestion control with a 'Scalable' alternative, when a suitable AQM is deployed in the network the performance under load of all the above interactive applications can be significantly improved. For instance, queuing delay under heavy load with the example DCTCP/DualQ solution cited below on a DSL or Ethernet link is roughly 1 to 2 milliseconds at the 99th percentile without losing link utilization [DualPI2Linux], [DCttH19] (for other link types, see Section 6.3). This compares with 5-20 ms on *average* with a Classic congestion control and current state-of-the-art AQMs such as FQ-CoDel [RFC8290], PIE [RFC8033] or DOCSIS PIE [RFC8034] and about 20-30 ms at the 99th percentile [DualPI2Linux].

L4S is designed for incremental deployment. It is possible to deploy the L4S service at a bottleneck link alongside the existing best efforts service [DualPI2Linux] so that unmodified applications can start using it as soon as the sender's stack is updated. Access networks are typically designed with one link as the bottleneck for each site (which might be a home, small enterprise or mobile device), so deployment at either or both ends of this link should give nearly all the benefit in the respective direction. With some transport protocols, namely TCP and SCTP, the sender has to check that the receiver has been suitably updated to give more accurate feedback, whereas with more recent transport protocols such as QUIC and DCCP, all receivers have always been suitable.

This document presents the L4S architecture. It consists of three components: network support to isolate L4S traffic from classic traffic; protocol features that allow network elements to identify L4S traffic; and host support for L4S congestion controls. The protocol is defined separately [I-D.ietf-tsvwg-ecn-l4s-id] as an experimental change to Explicit Congestion Notification (ECN). This document describes and justifies the component parts and how they interact to provide the scalable, low latency, low loss Internet service. It also details the approach to incremental deployment, as briefly summarized above.

### 1.1. Document Roadmap

This document describes the L4S architecture in three passes. First this brief overview gives the very high level idea and states the main components with minimal rationale. This is only intended to give some context for the terminology definitions that follow in Section 3, and to explain the structure of the rest of the document. Then Section 4 goes into more detail on each component with some rationale, but still mostly stating what the architecture is, rather than why. Finally, Section 5 justifies why each element of the solution was chosen (Section 5.1) and why these choices were different from other solutions (Section 5.2).

Having described the architecture, Section 6 clarifies its applicability; that is, the applications and use-cases that motivated the design, the challenges applying the architecture to various link technologies, and various incremental deployment models: including the two main deployment topologies, different sequences for incremental deployment and various interactions with pre-existing approaches. The document ends with the usual tailpieces, including extensive discussion of traffic policing and other security considerations in Section 8.

## 2. L4S Architecture Overview

Below we outline the three main components to the L4S architecture; 1) the scalable congestion control on the sending host; 2) the AQM at the network bottleneck; and 3) the protocol between them.

But first, the main point to grasp is that low latency is not provided by the network - low latency results from the careful behaviour of the scalable congestion controllers used by L4S senders. The network does have a role - primarily to isolate the low latency of the carefully behaving L4S traffic from the higher queuing delay needed by traffic with pre-existing Classic behaviour. The network also alters the way it signals queue growth to the transport - It uses the Explicit Congestion Notification (ECN) protocol, but it signals the very start of queue growth -

immediately without the smoothing delay typical of Classic AQMs. Because ECN support is essential for L4S, senders use the ECN field as the protocol that allows the network to identify which packets are L4S and which are Classic.

1) **Host:** Scalable congestion controls already exist. They solve the scaling problem with Classic congestion controls, such as Reno or Cubic. Because flow rate has scaled since TCP congestion control was first designed in 1988, assuming the flow lasts long enough, it now takes hundreds of round trips (and growing) to recover after a congestion signal (whether a loss or an ECN mark) as shown in the examples in Section 5.1 and [RFC3649]. Therefore, control of queuing and utilization becomes very slack, and the slightest disturbances (e.g. from new flows starting) prevent a high rate from being attained.

   With a scalable congestion control, the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. This maintains the same degree of control over queueing and utilization whatever the flow rate, as well as ensuring that high throughput is more robust to disturbances. The scalable control used most widely (in controlled environments) is Data Center TCP (DCTCP [RFC8257]), which has been implemented and deployed in Windows Server Editions (since 2012), in Linux and in FreeBSD. Although DCTCP as-is functions well over wide-area round trip times, most implementations lack certain safety features that would be necessary for use outside controlled environments like data centres (see Section 6.4.3). So scalable congestion control needs to be implemented in TCP and other transport protocols (QUIC, SCTP, RTP/RTCP, RMCAT, etc.). Indeed, between the present document being drafted and published, the following scalable congestion controls were implemented: TCP Prague [PragueLinux], QUIC Prague, an L4S variant of the RMCAT SCReAM controller [SCReAM] and the L4S ECN part of BBRv2 [BBRv2] intended for TCP and QUIC transports.

2) **Network:** L4S traffic needs to be isolated from the queuing latency of Classic traffic. One queue per application flow (FQ) is one way to achieve this, e.g. FQ-CoDel [RFC8290]. However, using just two queues is sufficient and does not require inspection of transport layer headers in the network, which is not always possible (see Section 5.2). With just two queues, it might seem impossible to know how much capacity to schedule for each queue without inspecting how many flows at any one time are using each. And it would be undesirable to arbitrarily divide access network capacity into two partitions. The Dual Queue Coupled AQM was developed as a minimal complexity solution to this problem. It acts like a 'semi-permeable' membrane that

partitions latency but not bandwidth. As such, the two queues are for transition from Classic to L4S behaviour, not bandwidth prioritization.

Section 4 gives a high level explanation of how the per-flow-queue (FQ) and DualQ variants of L4S work, and [I-D.ietf-tsvwg-aqm-dualq-coupled] gives a full explanation of the DualQ Coupled AQM framework. A specific marking algorithm is not mandated for L4S AQMs. Appendices of [I-D.ietf-tsvwg-aqm-dualq-coupled] give non-normative examples that have been implemented and evaluated, and give recommended default parameter settings. It is expected that L4S experiments will improve knowledge of parameter settings and whether the set of marking algorithms needs to be limited.

**3) Protocol:** A sending host needs to distinguish L4S and Classic packets with an identifier so that the network can classify them into their separate treatments. The L4S identifier spec. [I-D.ietf-tsvwg-ecn-l4s-id] concludes that all alternatives involve compromises, but the ECT(1) and CE codepoints of the ECN field represent a workable solution. As already explained, the network also uses ECN to immediately signal the very start of queue growth to the transport.

## 3. Terminology

[Note to the RFC Editor (to be removed before publication as an RFC): The following definitions are copied from the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] for the reader's convenience. Except, here, Classic CC and Scalable CC are condensed because they refer to Section 5.1 later. Also the definition of Traffic Policing is not needed in [I-D.ietf-tsvwg-ecn-l4s-id].]

**Classic Congestion Control:** A congestion control behaviour that can co-exist with standard Reno [RFC5681] without causing significantly negative impact on its flow rate [RFC5033]. The scaling problem with Classic congestion control is explained, with examples, in Section 5.1 and in [RFC3649].

**Scalable Congestion Control:** A congestion control where the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. For instance, DCTCP averages 2 congestion signals per round-trip whatever the flow rate, as do other recently developed scalable congestion controls, e.g. Relentless TCP [Mathis09], TCP Prague [I-D.briscoe-iccrg-prague-congestion-control], [PragueLinux], BBRv2 [BBRv2], [I-D.cardwell-iccrg-bbr-congestion-control] and the L4S variant of SCReAM for real-time media [SCReAM], [RFC8298]). See Section 4.3 of [I-D.ietf-tsvwg-ecn-l4s-id] for more explanation.

**Classic service:** The Classic service is intended for all the congestion control behaviours that co-exist with Reno [RFC5681] (e.g. Reno itself, Cubic [RFC8312], Compound [I-D.sridharan-tcpm-ctcp], TFRC [RFC5348]). The term 'Classic queue' means a queue providing the Classic service.

**Low-Latency, Low-Loss Scalable throughput (L4S) service:** The 'L4S' service is intended for traffic from scalable congestion control algorithms, such as the Prague congestion control [I-D.briscoe-iccrg-prague-congestion-control], which was derived from DCTCP [RFC8257]. The L4S service is for more general traffic than just Prague -- it allows the set of congestion controls with similar scaling properties to Prague to evolve, such as the examples listed above (Relentless, SCReAM). The term 'L4S queue' means a queue providing the L4S service.

The terms Classic or L4S can also qualify other nouns, such as 'queue', 'codepoint', 'identifier', 'classification', 'packet', 'flow'. For example: an L4S packet means a packet with an L4S identifier sent from an L4S congestion control.

Both Classic and L4S services can cope with a proportion of unresponsive or less-responsive traffic as well, but in the L4S case its rate has to be smooth enough or low enough to not build a queue (e.g. DNS, VoIP, game sync datagrams, etc.).

**Reno-friendly:** The subset of Classic traffic that is friendly to the standard Reno congestion control defined for TCP in [RFC5681]. The TFRC spec. [RFC5348] indirectly implies that 'friendly' is defined as "generally within a factor of two of the sending rate of a TCP flow under the same conditions". Reno-friendly is used here in place of 'TCP-friendly', given the latter has become imprecise, because the TCP protocol is now used with so many different congestion control behaviours, and Reno is used in non-TCP transports such as QUIC [RFC9000].

**Classic ECN:** The original Explicit Congestion Notification (ECN) protocol [RFC3168], which requires ECN signals to be treated as equivalent to drops, both when generated in the network and when responded to by the sender.

L4S uses the ECN field as an identifier [I-D.ietf-tsvwg-ecn-l4s-id] with the names for the four codepoints of the 2-bit IP-ECN field unchanged from those defined in the ECN spec [RFC3168]: Not ECT, ECT(0), ECT(1) and CE, where ECT stands for ECN-Capable Transport and CE stands for Congestion Experienced. A packet marked with the CE codepoint is termed 'ECN-marked' or sometimes just 'marked' where the context makes ECN obvious.

**Site:**
A home, mobile device, small enterprise or campus, where the network bottleneck is typically the access link to the site. Not all network arrangements fit this model but it is a useful, widely applicable generalization.

**Traffic policing:** Limiting traffic by dropping packets or shifting them to lower service class (as opposed to introducing delay, which is termed traffic shaping). Policing can involve limiting average rate and/or burst size. Policing focused on limiting queuing but not average flow rate is termed congestion policing, latency policing, burst policing or queue protection in this document. Otherwise, the term rate policing is used.

## 4. L4S Architecture Components

The L4S architecture is composed of the elements in the following three subsections.

### 4.1. Protocol Mechanisms

The L4S architecture involves: a) unassignment of the previous use of the identifier; b) reassignment of the same identifier; and c) optional further identifiers:

a. An essential aspect of a scalable congestion control is the use of explicit congestion signals. 'Classic' ECN [RFC3168] requires an ECN signal to be treated as equivalent to drop, both when it is generated in the network and when it is responded to by hosts. L4S needs networks and hosts to support a more fine-grained meaning for each ECN signal that is less severe than a drop, so that the L4S signals:

   *can be much more frequent;

   *can be signalled immediately, without the significant delay required to smooth out fluctuations in the queue.

   To enable L4S, the standards track Classic ECN spec. [RFC3168] has had to be updated to allow L4S packets to depart from the 'equivalent to drop' constraint. [RFC8311] is a standards track update to relax specific requirements in RFC 3168 (and certain other standards track RFCs), which clears the way for the experimental changes proposed for L4S. Also, the ECT(1) codepoint was previously assigned as the experimental ECN nonce [RFC3540], which RFC 8311 recategorizes as historic to make the codepoint available again.

b. [I-D.ietf-tsvwg-ecn-l4s-id] specifies that ECT(1) is used as the identifier to classify L4S packets into a separate treatment from Classic packets. This satisfies the requirement for identifying an alternative ECN treatment in [RFC4774].

The CE codepoint is used to indicate Congestion Experienced by both L4S and Classic treatments. This raises the concern that a Classic AQM earlier on the path might have marked some ECT(0) packets as CE. Then these packets will be erroneously classified into the L4S queue. Appendix B of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] explains why five unlikely eventualities all have to coincide for this to have any detrimental effect, which even then would only involve a vanishingly small likelihood of a spurious retransmission.

c. A network operator might wish to include certain unresponsive, non-L4S traffic in the L4S queue if it is deemed to be smoothly enough paced and low enough rate not to build a queue. For instance, VoIP, low rate datagrams to sync online games, relatively low rate application-limited traffic, DNS, LDAP, etc. This traffic would need to be tagged with specific identifiers, e.g. a low latency Diffserv Codepoint such as Expedited Forwarding (EF [RFC3246]), Non-Queue-Building (NQB [I-D.ietf-tsvwg-nqb]), or operator-specific identifiers.

### 4.2. Network Components

The L4S architecture aims to provide low latency without the *need* for per-flow operations in network components. Nonetheless, the architecture does not preclude per-flow solutions. The following bullets describe the known arrangements: a) the DualQ Coupled AQM with an L4S AQM in one queue coupled from a Classic AQM in the other; b) Per-Flow Queues with an instance of a Classic and an L4S AQM in each queue; c) Dual queues with per-flow AQMs, but no per-flow queues:

a. The Dual Queue Coupled AQM (illustrated in Figure 1) achieves the 'semi-permeable' membrane property mentioned earlier as follows:

   * Latency isolation: Two separate queues are used to isolate L4S queuing delay from the larger queue that Classic traffic needs to maintain full utilization.

   * Bandwidth pooling: The two queues act as if they are a single pool of bandwidth in which flows of either type get roughly equal throughput without the scheduler needing to identify any flows. This is achieved by having an AQM in each queue, but the Classic AQM provides a congestion signal to both queues in a manner that ensures a consistent response from the two classes of congestion control.

Specifically, the Classic AQM generates a drop/mark probability based on congestion in its own queue, which it uses both to drop/mark packets in its own queue and to affect the marking probability in the L4S queue. The strength of the coupling of the congestion signalling between the two queues is enough to make the L4S flows slow down to leave the right amount of capacity for the Classic flows (as they would if they were the same type of traffic sharing the same queue).

Then the scheduler can serve the L4S queue with priority (denoted by the '1' on the higher priority input), because the L4S traffic isn't offering up enough traffic to use all the priority that it is given. Therefore:

* for latency isolation on short time-scales (sub-round-trip) the prioritization of the L4S queue protects its low latency by allowing bursts to dissipate quickly;

* but for bandwidth pooling on longer time-scales (round-trip and longer) the Classic queue creates an equal and opposite pressure against the L4S traffic to ensure that neither has priority when it comes to bandwidth - the tension between prioritizing L4S and coupling the marking from the Classic AQM results in approximate per-flow fairness.

To protect against unresponsive traffic taking advantage of the prioritization of the L4S queue and starving the Classic queue, it is advisable for the priority to be conditional, not strict (see Appendix A of the DualQ spec [I-D.ietf-tsvwg-aqm-dualq-coupled]).

When there is no Classic traffic, the L4S queue's own AQM comes into play. It starts congestion marking with a very shallow queue, so L4S traffic maintains very low queuing delay.

If either queue becomes persistently overloaded, drop of ECN-capable packets is introduced, as recommended in Section 7 of the ECN spec [RFC3168] and Section 4.2.1 of the AQM recommendations [RFC7567]. Then both queues introduce the same level of drop (not shown in the figure).

The Dual Queue Coupled AQM has been specified as generically as possible [I-D.ietf-tsvwg-aqm-dualq-coupled] without specifying the particular AQMs to use in the two queues so that designers are free to implement diverse ideas. Informational appendices in that draft give pseudocode examples of two different specific AQM approaches: one called DualPI2 (pronounced Dual PI Squared) [DualPI2Linux] that uses the PI2 variant of PIE, and a zero-config variant of RED called Curvy RED. A DualQ Coupled AQM based on PIE has also been specified and implemented for Low Latency DOCSIS [DOCSIS3.1].

```
                     (3)                  (2)
              .-------^------..------------^------------------.
 ,-(1)-----.                               _____
; ________  :            L4S  -------.    |     |
:|Scalable| :               _\      ||__\_|mark |
:| sender | :  __________  / /      ||  / |_____|\   _________
:|________|\; |          |/   -------'       ^    \1|condit'nl|
 `---------'\_|  IP-ECN  |          Coupling :     \|priority |_\
  ________  / |Classifier|                   :     /|scheduler| /
 |Classic |/  |__________|\   -------.     __:__  / |_________|
 | sender |                \_\ || | ||__\_|mark/|/
 |________|                  / || | ||  / |drop |
                      Classic -------'    |_____|
```

Figure 1: Components of an L4S DualQ Coupled AQM Solution: 1) Scalable Sending Host; 2) Isolation in separate network queues; and 3) Packet Identification Protocol

b. Per-Flow Queues and AQMs: A scheduler with per-flow queues such as FQ-CoDel or FQ-PIE can be used for L4S. For instance within each queue of an FQ-CoDel system, as well as a CoDel AQM, there is typically also the option of ECN marking at an immediate (unsmoothed) shallow threshold to support use in data centres (see Sec.5.2.7 of the FQ-CoDel spec [RFC8290]). In Linux, this has been modified so that the shallow threshold can be solely applied to ECT(1) packets [FQ_CoDel_Thresh]. Then, if there is a flow of non-ECN or ECT(0) packets in the per-flow-queue, the Classic AQM (e.g. CoDel) is applied; while if there is a flow of ECT(1) packets in the queue, the shallower (typically sub-millisecond) threshold is applied. In addition, ECT(0) and not-ECT packets could potentially be classified into a separate flow-queue from ECT(1) and CE packets to avoid them mixing if they share a common flow-identifier (e.g. in a VPN).

c. Dual-queues, but per-flow AQMs: It should also be possible to use dual queues for isolation, but with per-flow marking to control flow-rates (instead of the coupled per-queue marking of the Dual Queue Coupled AQM). One of the two queues would be for isolating L4S packets, which would be classified by the ECN codepoint. Flow rates could be controlled by flow-specific marking. The policy goal of the marking could be to differentiate flow rates (e.g. [Nadas20], which requires additional signalling of a per-flow 'value'), or to equalize flow-rates (perhaps in a similar way to Approx Fair

CoDel [AFCD], [I-D.morton-tsvwg-codel-approx-fair], but with two queues not one).

Note that whenever the term 'DualQ' is used loosely without saying whether marking is per-queue or per-flow, it means a dual queue AQM with per-queue marking.

### 4.3. Host Mechanisms

The L4S architecture includes two main mechanisms in the end host that we enumerate next:

a. Scalable Congestion Control at the sender: Section 2 defines a scalable congestion control as one where the average time from one congestion signal to the next (the recovery time) remains invariant as the flow rate scales, all other factors being equal. Data Center TCP is the most widely used example. It has been documented as an informational record of the protocol currently in use in controlled environments [RFC8257]. A draft list of safety and performance improvements for a scalable congestion control to be usable on the public Internet has been drawn up (the so-called 'Prague L4S requirements' in Appendix A of [I-D.ietf-tsvwg-ecn-l4s-id]). The subset that involve risk of harm to others have been captured as normative requirements in Section 4 of [I-D.ietf-tsvwg-ecn-l4s-id]. TCP Prague [I-D.briscoe-iccrg-prague-congestion-control] has been implemented in Linux as a reference implementation to address these requirements [PragueLinux].

   Transport protocols other than TCP use various congestion controls that are designed to be friendly with Reno. Before they can use the L4S service, they will need to be updated to implement a scalable congestion response, which they will have to indicate by using the ECT(1) codepoint. Scalable variants are under consideration for more recent transport protocols, e.g. QUIC, and the L4S ECN part of BBRv2 [BBRv2], [I-D.cardwell-iccrg-bbr-congestion-control] is a scalable congestion control intended for the TCP and QUIC transports, amongst others. Also, an L4S variant of the RMCAT SCReAM controller [RFC8298] has been implemented [SCReAM] for media transported over RTP.

   Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] defines scalable congestion control in more detail, and specifies the requirements that an L4S scalable congestion control has to comply with.

b. The ECN feedback in some transport protocols is already sufficiently fine-grained for L4S (specifically DCCP [RFC4340] and QUIC [RFC9000]). But others either require update or are in the process of being updated:

* For the case of TCP, the feedback protocol for ECN embeds the assumption from Classic ECN [RFC3168] that an ECN mark is equivalent to a drop, making it unusable for a scalable TCP. Therefore, the implementation of TCP receivers will have to be upgraded [RFC7560]. Work to standardize and implement more accurate ECN feedback for TCP (AccECN) is in progress [I-D.ietf-tcpm-accurate-ecn], [PragueLinux].

* ECN feedback was only roughly sketched in an appendix of the now obsoleted second specification of SCTP [RFC4960], while a fuller specification was proposed in a long-expired draft [I-D.stewart-tsvwg-sctpecn]. A new design would need to be implemented and deployed before SCTP could support L4S.

* For RTP, sufficient ECN feedback was defined in [RFC6679], but [RFC8888] defines the latest standards track improvements.

## 5. Rationale

### 5.1. Why These Primary Components?

**Explicit congestion signalling (protocol):** Explicit congestion signalling is a key part of the L4S approach. In contrast, use of drop as a congestion signal creates a tension because drop is both an impairment (less would be better) and a useful signal (more would be better):

* Explicit congestion signals can be used many times per round trip, to keep tight control, without any impairment. Under heavy load, even more explicit signals can be applied, so that the queue can be kept short whatever the load. In contrast, Classic AQMs have to introduce very high packet drop at high load to keep the queue short. By using ECN, an L4S congestion control's sawtooth reduction can be smaller and therefore return to the operating point more often, without worrying that more sawteeth will cause more signals. The consequent smaller amplitude sawteeth fit between an empty queue and a very shallow marking threshold (~1 ms in the public Internet), so queue delay variation can be very low, without risk of under-utilization.

* Explicit congestion signals can be emitted immediately to track fluctuations of the queue. L4S shifts smoothing from the network to the host. The network doesn't know the round trip times of any of the flows. So if the network is

responsible for smoothing (as in the Classic approach), it has to assume a worst case RTT, otherwise long RTT flows would become unstable. This delays Classic congestion signals by 100-200 ms. In contrast, each host knows its own round trip time. So, in the L4S approach, the host can smooth each flow over its own RTT, introducing no more smoothing delay than strictly necessary (usually only a few milliseconds). A host can also choose not to introduce any smoothing delay if appropriate, e.g. during flow start-up.

Neither of the above are feasible if explicit congestion signalling has to be considered 'equivalent to drop' (as was required with Classic ECN [RFC3168]), because drop is an impairment as well as a signal. So drop cannot be excessively frequent, and drop cannot be immediate, otherwise too many drops would turn out to have been due to only a transient fluctuation in the queue that would not have warranted dropping a packet in hindsight. Therefore, in an L4S AQM, the L4S queue uses a new L4S variant of ECN that is not equivalent to drop (see section 5.2 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]), while the Classic queue uses either Classic ECN [RFC3168] or drop, which are equivalent to each other.

Before Classic ECN was standardized, there were various proposals to give an ECN mark a different meaning from drop. However, there was no particular reason to agree on any one of the alternative meanings, so 'equivalent to drop' was the only compromise that could be reached. RFC 3168 contains a statement that:

> "An environment where all end nodes were ECN-Capable could allow new criteria to be developed for setting the CE codepoint, and new congestion control mechanisms for end-node reaction to CE packets. However, this is a research issue, and as such is not addressed in this document."

**Latency isolation (network):** L4S congestion controls keep queue delay low whereas Classic congestion controls need a queue of the order of the RTT to avoid under-utilization. One queue cannot have two lengths, therefore L4S traffic needs to be isolated in a separate queue (e.g. DualQ) or queues (e.g. FQ).

**Coupled congestion notification:** Coupling the congestion notification between two queues as in the DualQ Coupled AQM is not necessarily essential, but it is a simple way to allow senders to determine their rate, packet by packet, rather than be overridden by a network scheduler. An alternative is for a network scheduler to control the rate of each application flow (see discussion in Section 5.2).

**L4S packet identifier (protocol):** Once there are at least two treatments in the network, hosts need an identifier at the IP layer to distinguish which treatment they intend to use.

**Scalable congestion notification:** A scalable congestion control in the host keeps the signalling frequency from the network high whatever the flow rate, so that queue delay variations can be small when conditions are stable, and rate can track variations in available capacity as rapidly as possible otherwise.

**Low loss:** Latency is not the only concern of L4S. The 'Low Loss' part of the name denotes that L4S generally achieves zero congestion loss due to its use of ECN. Otherwise, loss would itself cause delay, particularly for short flows, due to retransmission delay [RFC2884].

**Scalable throughput:** The "Scalable throughput" part of the name denotes that the per-flow throughput of scalable congestion controls should scale indefinitely, avoiding the imminent scaling problems with Reno-friendly congestion control algorithms [RFC3649]. It was known when TCP congestion avoidance was first developed in 1988 that it would not scale to high bandwidth-delay products (see footnote 6 in [TCP-CA]). Today, regular broadband flow rates over WAN distances are already beyond the scaling range of Classic Reno congestion control. So `less unscalable' Cubic [RFC8312] and Compound [I-D.sridharan-tcpm-ctcp] variants of TCP have been successfully deployed. However, these are now approaching their scaling limits.

For instance, we will consider a scenario with a maximum RTT of 30 ms at the peak of each sawtooth. As Reno packet rate scales 8x from 1,250 to 10,000 packet/s (from 15 to 120 Mb/s with 1500 B packets), the time to recover from a congestion event rises proportionately by 8x as well, from 422 ms to 3.38 s. It is clearly problematic for a congestion control to take multiple seconds to recover from each congestion event. Cubic [RFC8312] was developed to be less unscalable, but it is approaching its scaling limit; with the same max RTT of 30 ms, at 120 Mb/s Cubic is still fully in its Reno-friendly mode, so it takes about 4.3 s to recover. However, once the flow rate scales by 8x again to 960 Mb/s it enters true Cubic mode, with a recovery time of 12.2 s. From then on, each further scaling by 8x doubles Cubic's recovery time (because the cube root of 8 is 2), e.g. at 7.68 Gb/s the recovery time is 24.3 s. In contrast, a scalable congestion control like DCTCP or TCP Prague induces 2 congestion signals per round trip on average, which remains invariant for any flow rate, keeping dynamic control very tight.

For a feel of where the global average lone-flow download sits on this scale at the time of writing (2021), according to [BDPdata] globally averaged fixed access capacity was 103 Mb/s in 2020 and averaged base RTT to a CDN was 25-34ms in 2019. Averaging of per-country data was weighted by Internet user population (data collected globally is necessarily of variable quality, but the paper does double-check that the outcome compares well against a second source). So a lone CUBIC flow would at best take about 200 round trips (5 s) to recover from each of its sawtooth reductions, if the flow even lasted that long. This is described as 'at best' because it assumes everyone uses an AQM, whereas in reality most users still have a (probably bloated) tail-drop buffer. In the tail-drop case, likely average recovery time would be at least 4x 5 s, if not more, because RTT under load would be at least double that of an AQM, and recovery time depends on the square of RTT.

Although work on scaling congestion controls tends to start with TCP as the transport, the above is not intended to exclude other transports (e.g. SCTP, QUIC) or less elastic algorithms (e.g. RMCAT), which all tend to adopt the same or similar developments.

### 5.2. What L4S adds to Existing Approaches

All the following approaches address some part of the same problem space as L4S. In each case, it is shown that L4S complements them or improves on them, rather than being a mutually exclusive alternative:

**Diffserv:** Diffserv addresses the problem of bandwidth apportionment for important traffic as well as queuing latency for delay-sensitive traffic. Of these, L4S solely addresses the problem of queuing latency. Diffserv will still be necessary where important traffic requires priority (e.g. for commercial reasons, or for protection of critical infrastructure traffic) - see [I-D.briscoe-tsvwg-l4s-diffserv]. Nonetheless, the L4S approach can provide low latency for all traffic within each Diffserv class (including the case where there is only the one default Diffserv class).

Also, Diffserv can only provide a latency benefit if a small subset of the traffic on a bottleneck link requests low latency. As already explained, it has no effect when all the applications in use at one time at a single site (home, small business or mobile device) require low latency. In contrast, because L4S works for all traffic, it needs none of the management baggage (traffic policing, traffic contracts) associated with favouring some packets over others. This lack of management baggage ought to give L4S a better chance of end-to-end deployment.

In particular, because networks tend not to trust end systems to identify which packets should be favoured over others, where networks assign packets to Diffserv classes they tend to use packet inspection of application flow identifiers or deeper inspection of application signatures. Thus, nowadays, Diffserv doesn't always sit well with encryption of the layers above IP [RFC8404]. So users have to choose between privacy and QoS.

As with Diffserv, the L4S identifier is in the IP header. But, in contrast to Diffserv, the L4S identifier does not convey a want or a need for a certain level of quality. Rather, it promises a certain behaviour (scalable congestion response), which networks can objectively verify if they need to. This is because low delay depends on collective host behaviour, whereas bandwidth priority depends on network behaviour.

**State-of-the-art AQMs:** AQMs such as PIE and FQ-CoDel give a significant reduction in queuing delay relative to no AQM at all. L4S is intended to complement these AQMs, and should not distract from the need to deploy them as widely as possible. Nonetheless, AQMs alone cannot reduce queuing delay too far without significantly reducing link utilization, because the root cause of the problem is on the host - where Classic congestion controls use large saw-toothing rate variations. The L4S approach resolves this tension between delay and utilization by enabling hosts to minimize the amplitude of their sawteeth. A single-queue Classic AQM is not sufficient to allow hosts to use small sawteeth for two reasons: i) smaller sawteeth would not get lower delay in an AQM designed for larger amplitude Classic sawteeth, because a queue can only have one length at a time; and ii) much smaller sawteeth implies much more frequent sawteeth, so L4S flows would drive a Classic AQM into a high level of ECN-marking, which would appear as heavy congestion to Classic flows, which in turn would greatly reduce their rate as a result (see Section 6.4.4).

**Per-flow queuing or marking:** Similarly, per-flow approaches such as FQ-CoDel or Approx Fair CoDel [AFCD] are not incompatible with the L4S approach. However, per-flow queuing alone is not enough - it only isolates the queuing of one flow from others; not from itself. Per-flow implementations need to have support for scalable congestion control added, which has already been done for FQ-CoDel in Linux (see Sec.5.2.7 of [RFC8290] and [FQ_CoDel_Thresh]). Without this simple modification, per-flow AQMs like FQ-CoDel would still not be able to support applications that need both very low delay and high bandwidth, e.g. video-based control of remote procedures, or interactive cloud-based video (see Note 1 below).

Although per-flow techniques are not incompatible with L4S, it is important to have the DualQ alternative. This is because handling end-to-end (layer 4) flows in the network (layer 3 or 2) precludes some important end-to-end functions. For instance:

a. Per-flow forms of L4S like FQ-CoDel are incompatible with full end-to-end encryption of transport layer identifiers for privacy and confidentiality (e.g. IPSec or encrypted VPN tunnels, as opposed to DTLS over UDP), because they require packet inspection to access the end-to-end transport flow identifiers.

   In contrast, the DualQ form of L4S requires no deeper inspection than the IP layer. So, as long as operators take the DualQ approach, their users can have both very low queuing delay and full end-to-end encryption [RFC8404].

b. With per-flow forms of L4S, the network takes over control of the relative rates of each application flow. Some see it as an advantage that the network will prevent some flows running faster than others. Others consider it an inherent part of the Internet's appeal that applications can control their rate while taking account of the needs of others via congestion signals. They maintain that this has allowed applications with interesting rate behaviours to evolve, for instance, variable bit-rate video that varies around an equal share rather than being forced to remain equal at every instant, or e2e scavenger behaviours [RFC6817] that use less than an equal share of capacity [LEDBAT_AQM].

   The L4S architecture does not require the IETF to commit to one approach over the other, because it supports both, so that the 'market' can decide. Nonetheless, in the spirit of 'Do one thing and do it well' [McIlroy78], the DualQ option provides low delay without prejudging the issue of flow-rate control. Then, flow rate policing can be added separately if desired. This allows application control up to a point, but the network can still choose to set the point at which it intervenes to prevent one flow completely starving another.

Note:

1. It might seem that self-inflicted queuing delay within a per-flow queue should not be counted, because if the delay wasn't in the network it would just shift to the sender. However, modern adaptive applications, e.g. HTTP/2 [RFC9113] or some interactive media applications (see Section 6.1), can keep low latency objects at the front of their local send queue by shuffling priorities of other objects dependent on the progress of other transfers (for example see [lowat]). They cannot shuffle objects once they have released them into the network.

**Alternative Back-off ECN (ABE):** Here again, L4S is not an alternative to ABE but a complement that introduces much lower queuing delay. ABE [RFC8511] alters the host behaviour in response to ECN marking to utilize a link better and give ECN flows faster throughput. It uses ECT(0) and assumes the network still treats ECN and drop the same. Therefore, ABE exploits any lower queuing delay that AQMs can provide. But, as explained above, AQMs still cannot reduce queuing delay too far without losing link utilization (to allow for other, non-ABE, flows).

**BBR:** Bottleneck Bandwidth and Round-trip propagation time (BBR [I-D.cardwell-iccrg-bbr-congestion-control]) controls queuing delay end-to-end without needing any special logic in the network, such as an AQM. So it works pretty-much on any path. BBR keeps queuing delay reasonably low, but perhaps not quite as low as with state-of-the-art AQMs such as PIE or FQ-CoDel, and certainly nowhere near as low as with L4S. Queuing delay is also not consistently low, due to BBR's regular bandwidth probing spikes and its aggressive flow start-up phase.

L4S complements BBR. Indeed, BBRv2 can use L4S ECN where available and a scalable L4S congestion control behaviour in response to any ECN signalling from the path [BBRv2]. The L4S ECN signal complements the delay based congestion control aspects of BBR with an explicit indication that hosts can use, both to converge on a fair rate and to keep below a shallow queue target set by the network. Without L4S ECN, both these aspects need to be assumed or estimated.

## 6. Applicability

### 6.1. Applications

A transport layer that solves the current latency issues will provide new service, product and application opportunities.

With the L4S approach, the following existing applications also experience significantly better quality of experience under load:

* Gaming, including cloud based gaming;

* VoIP;

* Video conferencing;

* Web browsing;

* (Adaptive) video streaming;

* Instant messaging.

The significantly lower queuing latency also enables some interactive application functions to be offloaded to the cloud that would hardly even be usable today:

* Cloud based interactive video;

* Cloud based virtual and augmented reality.

The above two applications have been successfully demonstrated with L4S, both running together over a 40 Mb/s broadband access link loaded up with the numerous other latency sensitive applications in the previous list as well as numerous downloads - all sharing the same bottleneck queue simultaneously [L4Sdemo16]. For the former, a panoramic video of a football stadium could be swiped and pinched so that, on the fly, a proxy in the cloud could generate a sub-window of the match video under the finger-gesture control of each user. For the latter, a virtual reality headset displayed a viewport taken from a 360-degree camera in a racing car. The user's head movements controlled the viewport extracted by a cloud-based proxy. In both cases, with 7 ms end-to-end base delay, the additional queuing delay of roughly 1 ms was so low that it seemed the video was generated locally.

Using a swiping finger gesture or head movement to pan a video are extremely latency-demanding actions -- far more demanding than VoIP. Because human vision can detect extremely low delays of the order of single milliseconds when delay is translated into a visual lag between a video and a reference point (the finger or the orientation of the head sensed by the balance system in the inner ear -- the vestibular system). With an alternative AQM, the video noticeably lagged behind the finger gestures and head movements.

Without the low queuing delay of L4S, cloud-based applications like these would not be credible without significantly more access bandwidth (to deliver all possible video that might be viewed) and more local processing, which would increase the weight and power consumption of head-mounted displays. When all interactive processing can be done in the cloud, only the data to be rendered for the end user needs to be sent.

Other low latency high bandwidth applications such as:

* Interactive remote presence;

* Video-assisted remote control of machinery or industrial processes.

are not credible at all without very low queuing delay. No amount of extra access bandwidth or local processing can make up for lost time.

## 6.2. Use Cases

The following use-cases for L4S are being considered by various interested parties:

* Where the bottleneck is one of various types of access network: e.g. DSL, Passive Optical Networks (PON), DOCSIS cable, mobile, satellite (see Section 6.3 for some technology-specific details)

* Private networks of heterogeneous data centres, where there is no single administrator that can arrange for all the simultaneous changes to senders, receivers and network needed to deploy DCTCP:

  - a set of private data centres interconnected over a wide area with separate administrations, but within the same company

  - a set of data centres operated by separate companies interconnected by a community of interest network (e.g. for the finance sector)

  - multi-tenant (cloud) data centres where tenants choose their operating system stack (Infrastructure as a Service - IaaS)

* Different types of transport (or application) congestion control:

  - elastic (TCP/SCTP);

  - real-time (RTP, RMCAT);

  - query (DNS/LDAP).

* Where low delay quality of service is required, but without inspecting or intervening above the IP layer [RFC8404]:

  - mobile and other networks have tended to inspect higher layers in order to guess application QoS requirements. However, with growing demand for support of privacy and encryption, L4S offers an alternative. There is no need to select which traffic to favour for queuing, when L4S can give favourable queuing to all traffic.

* If queuing delay is minimized, applications with a fixed delay budget can communicate over longer distances, or via a longer chain of service functions [RFC7665] or onion routers.

* If delay jitter is minimized, it is possible to reduce the dejitter buffers on the receive end of video streaming, which should improve the interactive experience

### 6.3. Applicability with Specific Link Technologies

Certain link technologies aggregate data from multiple packets into bursts, and buffer incoming packets while building each burst. Wi-Fi, PON and cable all involve such packet aggregation, whereas fixed Ethernet and DSL do not. No sender, whether L4S or not, can do anything to reduce the buffering needed for packet aggregation. So an AQM should not count this buffering as part of the queue that it controls, given no amount of congestion signals will reduce it.

Certain link technologies also add buffering for other reasons, specifically:

* Radio links (cellular, Wi-Fi, satellite) that are distant from the source are particularly challenging. The radio link capacity can vary rapidly by orders of magnitude, so it is considered desirable to hold a standing queue that can utilize sudden increases of capacity;

* Cellular networks are further complicated by a perceived need to buffer in order to make hand-overs imperceptible;

L4S cannot remove the need for all these different forms of buffering. However, by removing 'the longest pole in the tent' (buffering for the large sawteeth of Classic congestion controls), L4S exposes all these 'shorter poles' to greater scrutiny.

Until now, the buffering needed for these additional reasons tended to be over-specified - with the excuse that none were 'the longest pole in the tent'. But having removed the 'longest pole', it becomes worthwhile to minimize them, for instance reducing packet aggregation burst sizes and MAC scheduling intervals.

Also certain link types, particularly radio-based links, are far more prone to transmission losses. Section 6.4.3 explains how an L4S response to loss has to be as drastic as a Classic response. Nonetheless, research referred to in the same section has demonstrated potential for considerably more effective loss repair at the link layer, due to the relaxed ordering constraints of L4S packets.

## 6.4. Deployment Considerations

L4S AQMs, whether DualQ [I-D.ietf-tsvwg-aqm-dualq-coupled] or FQ, e.g. [RFC8290] are, in themselves, an incremental deployment mechanism for L4S - so that L4S traffic can coexist with existing Classic (Reno-friendly) traffic. Section 6.4.1 explains why only deploying an L4S AQM in one node at each end of the access link will realize nearly all the benefit of L4S.

L4S involves both end systems and the network, so Section 6.4.2 suggests some typical sequences to deploy each part, and why there will be an immediate and significant benefit after deploying just one part.

Section 6.4.3 and Section 6.4.4 describe the converse incremental deployment case where there is no L4S AQM at the network bottleneck, so any L4S flow traversing this bottleneck has to take care in case it is competing with Classic traffic.

### 6.4.1. Deployment Topology

L4S AQMs will not have to be deployed throughout the Internet before L4S can benefit anyone. Operators of public Internet access networks typically design their networks so that the bottleneck will nearly always occur at one known (logical) link. This confines the cost of queue management technology to one place.

The case of mesh networks is different and will be discussed later in this section. But the known bottleneck case is generally true for Internet access to all sorts of different 'sites', where the word 'site' includes home networks, small- to medium-sized campus or enterprise networks and even cellular devices (Figure 2). Also, this known-bottleneck case tends to be applicable whatever the access link technology; whether xDSL, cable, PON, cellular, line of sight wireless or satellite.

Therefore, the full benefit of the L4S service should be available in the downstream direction when an L4S AQM is deployed at the ingress to this bottleneck link. And similarly, the full upstream service will be available once an L4S AQM is deployed at the ingress into the upstream link. (Of course, multi-homed sites would only see the full benefit once all their access links were covered.)

```
                                         ______
                                        (      )
                      __          __  (          )
                     |DQ\________/DQ|( enterprise )
                 ___ |__/        \__| ( /campus  )
                (   )                   (______)
              (      )                            ___||_
+----+      (          )  __                  __ /      \
| DC |-----(    Core    )|DQ\________________/DQ|| home |
+----+      (          ) |__/                \__||______|
               (_____) __
                      |DQ\__/\        __ ,===.
                      |__/    \  ____/DQ||| ||mobile
                               \/    \__|||_||device
                                         | o |
                                         `---'
```

Figure 2: Likely location of DualQ (DQ) Deployments in common access topologies

Deployment in mesh topologies depends on how overbooked the core is. If the core is non-blocking, or at least generously provisioned so that the edges are nearly always the bottlenecks, it would only be necessary to deploy an L4S AQM at the edge bottlenecks. For example, some data-centre networks are designed with the bottleneck in the hypervisor or host NICs, while others bottleneck at the top-of-rack switch (both the output ports facing hosts and those facing the core).

An L4S AQM would often next be needed where the Wi-Fi links in a home sometimes become the bottleneck. And an L4S AQM would eventually also need to be deployed at any other persistent bottlenecks such as network interconnections, e.g. some public Internet exchange points and the ingress and egress to WAN links interconnecting data-centres.

### 6.4.2. Deployment Sequences

For any one L4S flow to provide benefit, it requires three (or sometimes two) parts to have been deployed: i) the congestion control at the sender; ii) the AQM at the bottleneck; and iii) older transports (namely TCP) need upgraded receiver feedback too. This was the same deployment problem that ECN faced [RFC8170] so we have learned from that experience.

Firstly, L4S deployment exploits the fact that DCTCP already exists on many Internet hosts (Windows, FreeBSD and Linux); both servers and clients. Therefore, an L4S AQM can be deployed at a network bottleneck to immediately give a working deployment of all the L4S parts for testing, as long as the ECT(0) codepoint is switched to ECT(1). DCTCP needs some safety concerns to be fixed for general use over the public Internet (see Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]), but DCTCP is not on by default, so these issues can be managed within controlled deployments or controlled trials.

Secondly, the performance improvement with L4S is so significant that it enables new interactive services and products that were not previously possible. It is much easier for companies to initiate new work on deployment if there is budget for a new product trial. If, in contrast, there were only an incremental performance improvement (as with Classic ECN), spending on deployment tends to be much harder to justify.

Thirdly, the L4S identifier is defined so that initially network operators can enable L4S exclusively for certain customers or certain applications. But this is carefully defined so that it does not compromise future evolution towards L4S as an Internet-wide service. This is because the L4S identifier is defined not only as the end-to-end ECN field, but it can also optionally be combined with any other packet header or some status of a customer or their access link (see section 5.4 of [I-D.ietf-tsvwg-ecn-l4s-id]). Operators could do this anyway, even if it were not blessed by the IETF. However, it is best for the IETF to specify that, if they use their own local identifier, it must be in combination with the IETF's identifier. Then, if an operator has opted for an exclusive local-use approach, later they only have to remove this extra rule to make the service work Internet-wide - it will already traverse middleboxes, peerings, etc.

```
+-+--------------------+--------------------+---------------------+
| | Servers or proxies |      Access link    |             Clients |
+-+--------------------+--------------------+---------------------+
|0| DCTCP (existing)   |                     |    DCTCP (existing) |
+-+--------------------+--------------------+---------------------+
|1|                    |Add L4S AQM downstream|                     |
| |       WORKS DOWNSTREAM FOR CONTROLLED DEPLOYMENTS/TRIALS        |
+-+--------------------+--------------------+---------------------+
|2| Upgrade DCTCP to   |                     |Replace DCTCP feedb'k|
| | TCP Prague         |                     |         with AccECN |
| |                FULLY     WORKS     DOWNSTREAM                    |
+-+--------------------+--------------------+---------------------+
| |                    |                     |    Upgrade DCTCP to |
|3|                    | Add L4S AQM upstream |          TCP Prague |
| |                    |                     |                     |
| |             FULLY WORKS UPSTREAM AND DOWNSTREAM                  |
+-+--------------------+--------------------+---------------------+
```

Figure 3: Example L4S Deployment Sequence

Figure 3 illustrates some example sequences in which the parts of L4S might be deployed. It consists of the following stages, preceded by a presumption that DCTCP is already installed at both ends:

1. DCTCP is not applicable for use over the public Internet, so it is emphasized here that any DCTCP flow has to be completely contained within a controlled trial environment.

   Within this trial environment, once an L4S AQM has been deployed, the trial DCTCP flow will experience immediate benefit, without any other deployment being needed. In this example downstream deployment is first, but in other scenarios the upstream might be deployed first. If no AQM at all was previously deployed for the downstream access, an L4S AQM greatly improves the Classic service (as well as adding the L4S service). If an AQM was already deployed, the Classic service will be unchanged (and L4S will add an improvement on top).

2. In this stage, the name 'TCP Prague' [I-D.briscoe-iccrg-prague-congestion-control] is used to represent a variant of DCTCP that is designed to be used in a production Internet environment (that is, it has to comply with all the requirements in Section 4 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id], which then means it can be used over the public Internet). If the application is primarily unidirectional, 'TCP Prague' at one end will provide all the benefit needed.

For TCP transports, Accurate ECN feedback (AccECN) [I-D.ietf-tcpm-accurate-ecn] is needed at the other end, but it is a generic ECN feedback facility that is already planned to be deployed for other purposes, e.g. DCTCP, BBR. The two ends can be deployed in either order, because, in TCP, an L4S congestion control only enables itself if it has negotiated the use of AccECN feedback with the other end during the connection handshake. Thus, deployment of TCP Prague on a server enables L4S trials to move to a production service in one direction, wherever AccECN is deployed at the other end. This stage might be further motivated by the performance improvements of TCP Prague relative to DCTCP (see Appendix A.2 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]).

Unlike TCP, from the outset, QUIC ECN feedback [RFC9000] has supported L4S. Therefore, if the transport is QUIC, one-ended deployment of a Prague congestion control at this stage is simple and sufficient.

For QUIC, if a proxy sits in the path between multiple origin servers and the access bottlenecks to multiple clients, then upgrading the proxy with a Scalable congestion control would provide the benefits of L4S over all the clients' downstream bottlenecks in one go --- whether or not all the origin servers were upgraded. Conversely, where a proxy has not been upgraded, the clients served by it will not benefit from L4S at all in the downstream, even when any origin server behind the proxy has been upgraded to support L4S.

For TCP, a proxy upgraded to support 'TCP Prague' would provide the benefits of L4S downstream to all clients that support AccECN (whether or not they support L4S as well). And in the upstream, the proxy would also support AccECN as a receiver, so that any client deploying its own L4S support would benefit in the upstream direction, irrespective of whether any origin server beyond the proxy supported AccECN.

3. This is a two-move stage to enable L4S upstream. An L4S AQM or TCP Prague can be deployed in either order as already explained. To motivate the first of two independent moves, the deferred benefit of enabling new services after the second move has to be worth it to cover the first mover's investment risk. As explained already, the potential for new interactive services provides this motivation. An L4S AQM also improves the upstream Classic service - significantly if no other AQM has already been deployed.

Note that other deployment sequences might occur. For instance: the upstream might be deployed first; a non-TCP protocol might be used end-to-end, e.g. QUIC, RTP; a body such as the 3GPP might require L4S to be implemented in 5G user equipment, or other random acts of kindness.

#### 6.4.3. L4S Flow but Non-ECN Bottleneck

If L4S is enabled between two hosts, the L4S sender is required to coexist safely with Reno in response to any drop (see Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]).

Unfortunately, as well as protecting Classic traffic, this rule degrades the L4S service whenever there is any loss, even if the cause is not persistent congestion at a bottleneck, e.g.:

* congestion loss at other transient bottlenecks, e.g. due to bursts in shallower queues;

* transmission errors, e.g. due to electrical interference;

* rate policing.

Three complementary approaches are in progress to address this issue, but they are all currently research:

* In Prague congestion control, ignore certain losses deemed unlikely to be due to congestion (using some ideas from BBR [I-D.cardwell-iccrg-bbr-congestion-control] regarding isolated losses). This could mask any of the above types of loss while still coexisting with drop-based congestion controls.

* A combination of RACK, L4S and link retransmission without resequencing could repair transmission errors without the head of line blocking delay usually associated with link-layer retransmission [UnorderedLTE], [I-D.ietf-tsvwg-ecn-l4s-id];

* Hybrid ECN/drop rate policers (see Section 8.3).

L4S deployment scenarios that minimize these issues (e.g. over wireline networks) can proceed in parallel to this research, in the expectation that research success could continually widen L4S applicability.

#### 6.4.4. L4S Flow but Classic ECN Bottleneck

Classic ECN support is starting to materialize on the Internet as an increased level of CE marking. It is hard to detect whether this is all due to the addition of support for ECN in implementations of FQ-CoDel and/or FQ-COBALT, which is not generally problematic, because flow-queue (FQ) scheduling inherently prevents a flow from exceeding the 'fair' rate irrespective of its aggressiveness. However, some of this Classic ECN marking might be due to single-queue ECN deployment. This case is discussed in Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id].

#### 6.4.5. L4S AQM Deployment within Tunnels

An L4S AQM uses the ECN field to signal congestion. So, in common with Classic ECN, if the AQM is within a tunnel or at a lower layer, correct functioning of ECN signalling requires correct propagation of the ECN field up the layers [RFC6040], [I-D.ietf-tsvwg-rfc6040update-shim], [I-D.ietf-tsvwg-ecn-encap-guidelines].

## 7. IANA Considerations (to be removed by RFC Editor)

This specification contains no IANA considerations.

## 8. Security Considerations

### 8.1. Traffic Rate (Non-)Policing

#### 8.1.1. (Non-)Policing Rate per Flow

In the current Internet, ISPs usually enforce separation between the capacity of shared links assigned to different 'sites' (e.g. households, businesses or mobile users - see terminology in Section 3) using some form of scheduler [RFC0970]. And they use various techniques like redirection to traffic scrubbing facilities to deal with flooding attacks. However, there has never been a universal need to police the rate of individual application flows - the Internet has generally always relied on self-restraint of congestion controls at senders for sharing intra-'site' capacity.

L4S has been designed not to upset this status quo. If a DualQ is used to provide L4S service, section 4.2 of [I-D.ietf-tsvwg-aqm-dualq-coupled] explains how it is designed to give no more rate advantage to unresponsive flows than a single-queue AQM would, whether or not there is traffic overload.

Also, in case per-flow rate policing is ever required, it can be added because it is orthogonal to the distinction between L4S and Classic. As explained in Section 5.2, the DualQ variant of L4S provides low delay without prejudging the issue of flow-rate control. So, if flow-rate control is needed, per-flow-queuing (FQ) with L4S support can be used instead, or flow rate policing can be added as a modular addition to a DualQ. However, per-flow rate control is not usually deployed as a security mechanism, because an active attacker can just shard its traffic over more flow IDs if the rate of each is restricted.

#### 8.1.2. (Non-)Policing L4S Service Rate

Section 5.2 explains how Diffserv only makes a difference if some packets get less favourable treatment than others, which typically requires traffic rate policing for a low latency class. In contrast, it should not be necessary to rate-police access to the L4S service to protect the Classic service, because L4S is designed to reduce delay without harming the delay or rate of any Classic traffic.

During early deployment (and perhaps always), some networks will not offer the L4S service. In general, these networks should not need to police L4S traffic. They are required (by both the ECN spec [RFC3168] and the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]) not to change the L4S identifier, which would interfere with end-to-end congestion control. If they already treat ECN traffic as Not-ECT, they can merely treat L4S traffic as Not-ECT too. At a bottleneck, such networks will introduce some queuing and dropping. When a scalable congestion control detects a drop it will have to respond safely with respect to Classic congestion controls (as required in Section 4.3 of [I-D.ietf-tsvwg-ecn-l4s-id]). This will degrade the L4S service to be no better (but never worse) than Classic best efforts, whenever a non-ECN bottleneck is encountered on a path (see Section 6.4.3).

In cases that are expected to be rare, networks that solely support Classic ECN [RFC3168] in a single queue bottleneck might opt to police L4S traffic so as to protect competing Classic ECN traffic (for instance, see Section 6.1.3 of the L4S operational guidance [I-D.ietf-tsvwg-l4sops]). However, Section 4.3 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id] recommends that the sender adapts its congestion response to properly coexist with Classic ECN flows, i.e. reverting to the self-restraint approach.

Certain network operators might choose to restrict access to the L4S service, perhaps only to selected premium customers as a value-added service. Their packet classifier (item 2 in Figure 1) could identify such customers against some other field (e.g. source address range) as well as classifying on the ECN field. If only the ECN L4S identifier matched, but not the source address (say), the classifier could direct these packets (from non-premium customers) into the Classic queue. Explaining clearly how operators can use additional local classifiers (see section 5.4 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id]) is intended to remove any motivation to clear the L4S identifier. Then at least the L4S ECN identifier will be more likely to survive end-to-end even though the service may not be supported at every hop. Such local arrangements would only require simple registered/not-registered packet classification, rather than the managed, application-specific traffic policing against customer-specific traffic contracts that Diffserv uses.

### 8.2. 'Latency Friendliness'

Like the Classic service, the L4S service relies on self-restraint - limiting rate in response to congestion. In addition, the L4S service requires self-restraint in terms of limiting latency (burstiness). It is hoped that self-interest and guidance on dynamic behaviour (especially flow start-up, which might need to be standardized) will be sufficient to prevent transports from sending excessive bursts of L4S traffic, given the application's own latency will suffer most from such behaviour.

Because the L4S service can reduce delay without discernibly increasing the delay of any Classic traffic, it should not be necessary to police L4S traffic to protect the delay of Classic. However, whether burst policing becomes necessary to protect other L4S traffic remains to be seen. Without it, there will be potential for attacks on the low latency of the L4S service.

If needed, various arrangements could be used to address this concern:

**Local bottleneck queue protection:** A per-flow (5-tuple) queue protection function [I-D.briscoe-docsis-q-protection] has been developed for the low latency queue in DOCSIS, which has adopted the DualQ L4S architecture. It protects the low latency service from any queue-building flows that accidentally or maliciously classify themselves into the low latency queue. It is designed to score flows based solely on their contribution to queuing (not flow rate in itself). Then, if the shared low latency queue is at risk of exceeding a threshold, the function redirects enough packets of the highest scoring flow(s) into the Classic queue to preserve low latency.

**Distributed traffic scrubbing:** Rather than policing locally at each bottleneck, it may only be necessary to address problems reactively, e.g. punitively target any deployments of new bursty malware, in a similar way to how traffic from flooding attack sources is rerouted via scrubbing facilities.

**Local bottleneck per-flow scheduling:** Per-flow scheduling should inherently isolate non-bursty flows from bursty (see Section 5.2 for discussion of the merits of per-flow scheduling relative to per-flow policing).

**Distributed access subnet queue protection:** Per-flow queue protection could be arranged for a queue structure distributed across a subnet intercommunicating using lower layer control messages (see Section 2.1.4 of [QDyn]). For instance, in a radio access network, user equipment already sends regular buffer status reports to a radio network controller, which could use this information to remotely police individual flows.

**Distributed Congestion Exposure to Ingress Policers:** The Congestion Exposure (ConEx) architecture [RFC7713] uses egress audit to motivate senders to truthfully signal path congestion in-band where it can be used by ingress policers. An edge-to-edge variant of this architecture is also possible.

**Distributed Domain-edge traffic conditioning:** An architecture similar to Diffserv [RFC2475] may be preferred, where traffic is proactively conditioned on entry to a domain, rather than reactively policed only if it leads to queuing once combined with other traffic at a bottleneck.

**Distributed core network queue protection:** The policing function could be divided between per-flow mechanisms at the network ingress that characterize the burstiness of each flow into a signal carried with the traffic, and per-class mechanisms at bottlenecks that act on these signals if queuing actually occurs once the traffic converges. This would be somewhat similar to [Nadas20], which is in turn similar to the idea behind core stateless fair queuing.

No single one of these possible queue protection capabilities is considered an essential part of the L4S architecture, which works without any of them under non-attack conditions (much as the Internet normally works without per-flow rate policing). Indeed, even where latency policers are deployed, under normal circumstances they would not intervene, and if operators found they were not necessary they could disable them. Part of the L4S experiment will be to see whether such a function is necessary, and which arrangements are most appropriate to the size of the problem.

### 8.3. Interaction between Rate Policing and L4S

As mentioned in Section 5.2, L4S should remove the need for low latency Diffserv classes. However, those Diffserv classes that give certain applications or users priority over capacity, would still be applicable in certain scenarios (e.g. corporate networks). Then, within such Diffserv classes, L4S would often be applicable to give traffic low latency and low loss as well. Within such a Diffserv class, the bandwidth available to a user or application is often limited by a rate policer. Similarly, in the default Diffserv class, rate policers are sometimes used to partition shared capacity.

A classic rate policer drops any packets exceeding a set rate, usually also giving a burst allowance (variants exist where the policer re-marks non-compliant traffic to a discard-eligible

Diffserv codepoint, so they can be dropped elsewhere during contention). Whenever L4S traffic encounters one of these rate policers, it will experience drops and the source will have to fall back to a Classic congestion control, thus losing the benefits of L4S (Section 6.4.3). So, in networks that already use rate policers and plan to deploy L4S, it will be preferable to redesign these rate policers to be more friendly to the L4S service.

L4S-friendly rate policing is currently a research area (note that this is not the same as latency policing). It might be achieved by setting a threshold where ECN marking is introduced, such that it is just under the policed rate or just under the burst allowance where drop is introduced. For instance the two-rate three-colour marker [RFC2698] or a PCN threshold and excess-rate marker [RFC5670] could mark ECN at the lower rate and drop at the higher. Or an existing rate policer could have congestion-rate policing added, e.g. using the 'local' (non-ConEx) variant of the ConEx aggregate congestion policer [I-D.briscoe-conex-policing]. It might also be possible to design scalable congestion controls to respond less catastrophically to loss that has not been preceded by a period of increasing delay.

The design of L4S-friendly rate policers will require a separate dedicated document. For further discussion of the interaction between L4S and Diffserv, see [I-D.briscoe-tsvwg-l4s-diffserv].

### 8.4. ECN Integrity

Various ways have been developed to protect the integrity of the congestion feedback loop (whether signalled by loss, Classic ECN or L4S ECN) against misbehaviour by the receiver, sender or network (or all three). Brief details of each including applicability, pros and cons is given in Appendix C.1 of the L4S ECN spec [I-D.ietf-tsvwg-ecn-l4s-id].

### 8.5. Privacy Considerations

As discussed in Section 5.2, the L4S architecture does not preclude approaches that inspect end-to-end transport layer identifiers. For instance, L4S support has been added to FQ-CoDel, which classifies by application flow ID in the network. However, the main innovation of L4S is the DualQ AQM framework that does not need to inspect any deeper than the outermost IP header, because the L4S identifier is in the IP-ECN field.

Thus, the L4S architecture enables very low queuing delay without *requiring* inspection of information above the IP layer. This means that users who want to encrypt application flow identifiers, e.g. in

IPSec or other encrypted VPN tunnels, don't have to sacrifice low delay [RFC8404].

Because L4S can provide low delay for a broad set of applications that choose to use it, there is no need for individual applications or classes within that broad set to be distinguishable in any way while traversing networks. This removes much of the ability to correlate between the delay requirements of traffic and other identifying features [RFC6973]. There may be some types of traffic that prefer not to use L4S, but the coarse binary categorization of traffic reveals very little that could be exploited to compromise privacy.

## 9. Informative References


**[AFCD]** Xue, L., Kumar, S., Cui, C., Kondikoppa, P., Chiu, C-H., and S-J. Park, "Towards fair and low latency next generation high speed networks: AFCD queuing", Journal of Network and Computer Applications 70:183--193, July 2016, <https://doi.org/10.1016/j.jnca.2016.03.021>.

**[BBRv2]** Cardwell, N., "TCP BBR v2 Alpha/Preview Release", GitHub repository; Linux congestion control module, <https://github.com/google/bbr/blob/v2alpha/README.md>.

**[BDPdata]** Briscoe, B., "PI2 Parameters", Technical Report TR-BB-2021-001 arXiv:2107.01003 [cs.NI], July 2021, <https://arxiv.org/abs/2107.01003>.

**[BufferSize]** Appenzeller, G., Keslassy, I., and N. McKeown, "Sizing Router Buffers", In Proc. SIGCOMM'04 34(4):281--292, September 2004, <https://doi.org/10.1145/1015467.1015499>.

**[COBALT]** Palmei, J., Gupta, S., Imputato, P., Morton, J., Tahiliani, M. P., Avallone, S., and D. Täht, "Design and Evaluation of COBALT Queue Discipline", In Proc. IEEE Int'l Symp. Local and Metropolitan Area Networks (LANMAN'19) 2019:1-6, July 2019, <https://ieeexplore.ieee.org/abstract/document/8847054>.

**[DCttH19]** De Schepper, K., Bondarenko, O., Tilmans, O., and B. Briscoe, "`Data Centre to the Home': Ultra-Low Latency for All", Updated RITE project Technical Report , July 2019, <https://bobbriscoe.net/pubs.html#DCttH_TR>.

**[DOCSIS3.1]** CableLabs, "MAC and Upper Layer Protocols Interface (MULPI) Specification, CM-SP-MULPIv3.1", Data-Over-Cable Service Interface Specifications DOCSIS® 3.1 Version i17 or later, 21 January 2019, <https://specification-search.cablelabs.com/CM-SP-MULPIv3.1>.

**[DOCSIS3AQM]** White, G., "Active Queue Management Algorithms for DOCSIS 3.0; A Simulation Study of CoDel, SFQ-CoDel and PIE in DOCSIS 3.0 Networks", CableLabs Technical Report , April 2013, <{https://www.cablelabs.com/wp-content/uploads/2013/11/Active_Queue_Management_Algorithms_DOCSIS_3_0.pdf>.

**[DualPI2Linux]** Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and H. Steen, "DUALPI2 - Low Latency, Low Loss and Scalable (L4S) AQM", Proc. Linux Netdev 0x13 , March 2019, <https://www.netdevconf.org/0x13/session.html?talk-DUALPI2-AQM>.

**[Dukkipati06]** Dukkipati, N. and N. McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control", ACM CCR 36(1):59--62, January 2006, <https://dl.acm.org/doi/10.1145/1111322.1111336>.

**[FQ_CoDel_Thresh]** Høiland-Jørgensen, T., "fq_codel: generalise ce_threshold marking for subset of traffic", Linux Patch Commit ID: dfcb63ce1de6b10b, 20 October 2021, <https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.git/commit/?id=dfcb63ce1de6b10b>.

**[Hohlfeld14]** Hohlfeld, O., Pujol, E., Ciucu, F., Feldmann, A., and P. Barford, "A QoE Perspective on Sizing Network Buffers", Proc. ACM Internet Measurement Conf (IMC'14) hmm, November 2014, <https://doi.acm.org/10.1145/2663716.2663730>.

**[I-D.briscoe-conex-policing]**
Briscoe, B., "Network Performance Isolation using Congestion Policing", Work in Progress, Internet-Draft, draft-briscoe-conex-policing-01, 14 February 2014, <https://www.ietf.org/archive/id/draft-briscoe-conex-policing-01.txt>.

**[I-D.briscoe-docsis-q-protection]** Briscoe, B. and G. White, "The DOCSIS(r) Queue Protection Algorithm to Preserve Low Latency", Work in Progress, Internet-Draft, draft-briscoe-docsis-q-protection-06, 13 May 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-briscoe-docsis-q-protection/>.

**[I-D.briscoe-iccrg-prague-congestion-control]** Schepper, K. D., Tilmans, O., and B. Briscoe, "Prague Congestion Control", Work in Progress, Internet-Draft, draft-briscoe-iccrg-prague-congestion-control-01, 11 July 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-briscoe-iccrg-prague-congestion-control/>.

**[I-D.briscoe-tsvwg-l4s-diffserv]**
Briscoe, B., "Interactions between Low Latency, Low Loss, Scalable Throughput (L4S) and Differentiated Services", Work in Progress, Internet-Draft, draft-briscoe-tsvwg-l4s-diffserv-02, 2 July 2018, <https://datatracker.ietf.org/api/v1/doc/document/draft-briscoe-tsvwg-l4s-diffserv/>.

**[I-D.cardwell-iccrg-bbr-congestion-control]**
Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrg-bbr-congestion-control-02, 7 March 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-cardwell-iccrg-bbr-congestion-control/>.

**[I-D.ietf-tcpm-accurate-ecn]** Briscoe, B., Kühlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", Work in Progress, Internet-Draft, draft-ietf-tcpm-accurate-ecn-20, 25 July 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-tcpm-accurate-ecn/>.

**[I-D.ietf-tsvwg-aqm-dualq-coupled]** Schepper, K. D., Briscoe, B., and G. White, "DualQ Coupled AQMs for Low Latency, Low Loss and Scalable Throughput (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-aqm-dualq-coupled-24, 7 July 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-tsvwg-aqm-dualq-coupled/>.

**[I-D.ietf-tsvwg-ecn-encap-guidelines]** Briscoe, B. and J. Kaippallimalil, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-encap-guidelines-17, 11 July 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-tsvwg-ecn-encap-guidelines/>.

**[I-D.ietf-tsvwg-ecn-l4s-id]** Schepper, K. D. and B. Briscoe, "Explicit Congestion Notification (ECN) Protocol for Very Low Queuing Delay (L4S)", Work in Progress, Internet-Draft, draft-ietf-tsvwg-ecn-l4s-id-28, 8 August 2022,

<https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-tsvwg-ecn-l4s-id/>.

**[I-D.ietf-tsvwg-l4sops]** White, G., "Operational Guidance for Deployment of L4S in the Internet", Work in Progress, Internet-Draft, draft-ietf-tsvwg-l4sops-03, 28 April 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-tsvwg-l4sops/>.

**[I-D.ietf-tsvwg-nqb]** White, G. and T. Fossati, "A Non-Queue-Building Per-Hop Behavior (NQB PHB) for Differentiated Services", Work in Progress, Internet-Draft, draft-ietf-tsvwg-nqb-10, 4 March 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-tsvwg-nqb/>.

**[I-D.ietf-tsvwg-rfc6040update-shim]** Briscoe, B., "Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim", Work in Progress, Internet-Draft, draft-ietf-tsvwg-rfc6040update-shim-15, 11 July 2022, <https://datatracker.ietf.org/api/v1/doc/document/draft-ietf-tsvwg-rfc6040update-shim/>.

**[I-D.morton-tsvwg-codel-approx-fair]** Morton, J. and P. G. Heist, "Controlled Delay Approximate Fairness AQM", Work in Progress, Internet-Draft, draft-morton-tsvwg-codel-approx-fair-01, 9 March 2020, <https://www.ietf.org/archive/id/draft-morton-tsvwg-codel-approx-fair-01.txt>.

**[I-D.sridharan-tcpm-ctcp]** Sridharan, M., Tan, K., Bansal, D., and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", Work in Progress, Internet-Draft, draft-sridharan-tcpm-ctcp-02, 29 October 2007, <https://datatracker.ietf.org/api/v1/doc/document/draft-sridharan-tcpm-ctcp/>.

**[I-D.stewart-tsvwg-sctpecn]** Stewart, R. R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", Work in Progress, Internet-Draft, draft-stewart-tsvwg-sctpecn-05, 15 January 2014, <https://www.ietf.org/archive/id/draft-stewart-tsvwg-sctpecn-05.txt>.

**[L4Sdemo16]** Bondarenko, O., De Schepper, K., Tsang, I., and B. Briscoe, "Ultra-Low Delay for All: Live Experience, Live Analysis", Proc. MMSYS'16 pp33:1--33:4, May 2016, <https://dl.acm.org/citation.cfm?doid=2910017.2910633 (videos of demos: https://riteproject.eu/dctth/#1511dispatchwg )>.

**[LEDBAT_AQM]**
Al-Saadi, R., Armitage, G., and J. But, "Characterising LEDBAT Performance Through Bottlenecks Using PIE, FQ-CoDel and FQ-PIE Active Queue Management", Proc. IEEE 42nd Conference on Local Computer Networks (LCN) 278--285, 2017, <https://ieeexplore.ieee.org/document/8109367>.

**[lowat]** Meenan, P., "Optimizing HTTP/2 prioritization with BBR and tcp_notsent_lowat", Cloudflare Blog , 12 October 2018, <https://blog.cloudflare.com/http-2-prioritization-with-nginx/>.

**[Mathis09]** Mathis, M., "Relentless Congestion Control", PFLDNeT'09 , May 2009, <https://www.gdt.id.au/~gdt/presentations/2010-07-06-questnet-tcp/reference-materials/papers/mathis-relentless-congestion-control.pdf>.

**[McIlroy78]** McIlroy, M.D., Pinson, E. N., and B. A. Tague, "UNIX Time-Sharing System: Foreword", The Bell System Technical Journal 57:6(1902--1903), July 1978, <https://archive.org/details/bstj57-6-1899>.

**[Nadas20]** Nádas, S., Gombos, G., Fejes, F., and S. Laki, "A Congestion Control Independent L4S Scheduler", Proc. Applied Networking Research Workshop (ANRW '20) 45--51, July 2020, <https://doi.org/10.1145/3404868.3406669>.

**[NASA04]** Bailey, R.R., Trey Arthur III, J.J., and S.P. Williams, "Latency Requirements for Head-Worn Display S/EVS Applications", SPIE Defense and Security Symposium LF99-1955, April 2004, <https://ntrs.nasa.gov/api/citations/20120009198/downloads/20120009198.pdf?attachment=true>.

**[PragueLinux]** Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kühlewind, M., and A.S. Ahmed, "Implementing the `TCP Prague' Requirements for Low Latency Low Loss Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 , March 2019, <https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>.

**[QDyn]** Briscoe, B., "Rapid Signalling of Queue Dynamics", bobbriscoe.net Technical Report TR-BB-2017-001; arXiv:1904.07044 [cs.NI], September 2017, <https://arxiv.org/abs/1904.07044>.

**[Raaen14]** Raaen, K. and T-M. Grønli, "Latency thresholds for usability in games: A survey", Norsk IKT-konferanse for forskning og utdanning , 2014, <http://ojs.bibsys.no/index.php/NIK/article/view/9/6>.

**[Rajiullah15]** Rajiullah, M., "Towards a Low Latency Internet: Understanding and Solutions", Master's Thesis; Karlstad Uni, Dept of Maths & CS 2015:41, 2015, <https://www.diva-portal.org/smash/get/diva2:846109/FULLTEXT01.pdf>.

**[RFC0970]** Nagle, J., "On Packet Switches With Infinite Storage", RFC 970, DOI 10.17487/RFC0970, December 1985, <https://www.rfc-editor.org/info/rfc970>.

**[RFC2475]** Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <https://www.rfc-editor.org/info/rfc2475>.

**[RFC2698]** Heinanen, J. and R. Guerin, "A Two Rate Three Color Marker", RFC 2698, DOI 10.17487/RFC2698, September 1999, <https://www.rfc-editor.org/info/rfc2698>.

**[RFC2884]** Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, DOI 10.17487/RFC2884, July 2000, <https://www.rfc-editor.org/info/rfc2884>.

**[RFC3168]** Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

**[RFC3246]**
Davie, B., Charny, A., Bennet, J.C.R., Benson, K., Le Boudec, J.Y., Courtney, W., Davari, S., Firoiu, V., and D. Stiliadis, "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, DOI 10.17487/RFC3246, March 2002, <https://www.rfc-editor.org/info/rfc3246>.

**[RFC3540]** Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <https://www.rfc-editor.org/info/rfc3540>.

**[RFC3649]** Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, DOI 10.17487/RFC3649, December 2003, <https://www.rfc-editor.org/info/rfc3649>.

**[RFC4340]** Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, DOI

10.17487/RFC4340, March 2006, <https://www.rfc-editor.org/info/rfc4340>.

**[RFC4774]** Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, DOI 10.17487/RFC4774, November 2006, <https://www.rfc-editor.org/info/rfc4774>.

**[RFC4960]** Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

**[RFC5033]** Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <https://www.rfc-editor.org/info/rfc5033>.

**[RFC5348]** Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, DOI 10.17487/RFC5348, September 2008, <https://www.rfc-editor.org/info/rfc5348>.

**[RFC5670]** Eardley, P., Ed., "Metering and Marking Behaviour of PCN-Nodes", RFC 5670, DOI 10.17487/RFC5670, November 2009, <https://www.rfc-editor.org/info/rfc5670>.

**[RFC5681]** Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

**[RFC6040]** Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <https://www.rfc-editor.org/info/rfc6040>.

**[RFC6679]** Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, DOI 10.17487/RFC6679, August 2012, <https://www.rfc-editor.org/info/rfc6679>.

**[RFC6817]** Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, DOI 10.17487/RFC6817, December 2012, <https://www.rfc-editor.org/info/rfc6817>.

**[RFC6973]** Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

**[RFC7560]**
Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <https://www.rfc-editor.org/info/rfc7560>.

**[RFC7567]** Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <https://www.rfc-editor.org/info/rfc7567>.

**[RFC7665]** Halpern, J., Ed. and C. Pignataro, Ed., "Service Function Chaining (SFC) Architecture", RFC 7665, DOI 10.17487/RFC7665, October 2015, <https://www.rfc-editor.org/info/rfc7665>.

**[RFC7713]** Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <https://www.rfc-editor.org/info/rfc7713>.

**[RFC8033]** Pan, R., Natarajan, P., Baker, F., and G. White, "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem", RFC 8033, DOI 10.17487/RFC8033, February 2017, <https://www.rfc-editor.org/info/rfc8033>.

**[RFC8034]** White, G. and R. Pan, "Active Queue Management (AQM) Based on Proportional Integral Controller Enhanced PIE) for Data-Over-Cable Service Interface Specifications (DOCSIS) Cable Modems", RFC 8034, DOI 10.17487/RFC8034, February 2017, <https://www.rfc-editor.org/info/rfc8034>.

**[RFC8170]** Thaler, D., Ed., "Planning for Protocol Adoption and Subsequent Transitions", RFC 8170, DOI 10.17487/RFC8170, May 2017, <https://www.rfc-editor.org/info/rfc8170>.

**[RFC8257]** Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <https://www.rfc-editor.org/info/rfc8257>.

**[RFC8290]** Hoeiland-Joergensen, T., McKenney, P., Taht, D., Gettys, J., and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm", RFC 8290, DOI 10.17487/RFC8290, January 2018, <https://www.rfc-editor.org/info/rfc8290>.

**[RFC8298]**
Johansson, I. and Z. Sarker, "Self-Clocked Rate Adaptation for Multimedia", RFC 8298, DOI 10.17487/RFC8298, December 2017, <https://www.rfc-editor.org/info/rfc8298>.

**[RFC8311]** Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <https://www.rfc-editor.org/info/rfc8311>.

**[RFC8312]** Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks", RFC 8312, DOI 10.17487/RFC8312, February 2018, <https://www.rfc-editor.org/info/rfc8312>.

**[RFC8404]** Moriarty, K., Ed. and A. Morton, Ed., "Effects of Pervasive Encryption on Operators", RFC 8404, DOI 10.17487/RFC8404, July 2018, <https://www.rfc-editor.org/info/rfc8404>.

**[RFC8511]** Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <https://www.rfc-editor.org/info/rfc8511>.

**[RFC8888]** Sarker, Z., Perkins, C., Singh, V., and M. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", RFC 8888, DOI 10.17487/RFC8888, January 2021, <https://www.rfc-editor.org/info/rfc8888>.

**[RFC9000]** Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI

10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

**[RFC9113]** Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

**[SCReAM]** Johansson, I., "SCReAM", GitHub repository; , <https://github.com/EricssonResearch/scream/blob/master/README.md>.

**[TCP-CA]** Jacobson, V. and M.J. Karels, "Congestion Avoidance and Control", Laurence Berkeley Labs Technical Report , November 1988, <https://ee.lbl.gov/papers/congavoid.pdf>.

**[UnorderedLTE]** Austrheim, M.V., "Implementing immediate forwarding for 4G in a network simulator", Master's Thesis, Uni Oslo , June 2019.

## Acknowledgements

Thanks to Richard Scheffenegger, Wes Eddy, Karen Nielsen, David Black, Jake Holland, Vidhi Goel, Ermin Sakic, Praveen Balasubramanian, Gorry Fairhurst, Mirja Kuehlewind, Philip Eardley, Neal Cardwell, Pete Heist and Martin Duke for their useful review comments. Thanks also to the area reviewers: Marco Tiloca, Lars Eggert, Roman Danyliw and Eric Vyncke.

Bob Briscoe and Koen De Schepper were part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The contribution of Koen De Schepper was also part-funded by the 5Growth and DAEMON EU H2020 projects. Bob Briscoe was also part-funded by the Research Council of Norway through the TimeIn project, partly by CableLabs and partly by the Comcast Innovation Fund. The views expressed here are solely those of the authors.

## Authors' Addresses

Bob Briscoe (editor)
Independent
United Kingdom

Email: ietf@bobbriscoe.net
URI: https://bobbriscoe.net/

Koen De Schepper
Nokia Bell Labs
Antwerp
Belgium

Email: koen.de_schepper@nokia.com
URI: https://www.bell-labs.com/about/researcher-profiles/koende_schepper/

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
Spain

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: https://www.it.uc3m.es

Greg White
CableLabs
United States of America

Email: G.White@CableLabs.com