NOKIA BELL LABS

L4S – low latency, low loss, and scalable throughput

Enabling large-scale deployments of low-latency services

White paper

Werner Coomans, Koen De Schepper, Olivier Tilmans

Low and consistent end-to-end latency is fundamental to providing a good quality of experience for all interactive services, whether they are mission-critical services (e.g., remote control of straddle carriers at ports), business services (e.g., virtual meetings), or leisure services (e.g., online gaming).

This white paper explains how a next-generation IETF internet-protocol innovation called “L4S”, pioneered by Nokia Bell Labs, can be used to control and reduce the latency that users experience on the internet. L4S significantly improves end-to-end latency by tackling queuing latency, which is the single biggest – and most often overlooked – source of latency. Queuing latency is caused by packets waiting in buffers before being forwarded.

The IETF published the L4S RFCs in January 2023. This paper describes the fundamental components of L4S and outlines a path for end-to-end support in communications service provider (CSP) networks.

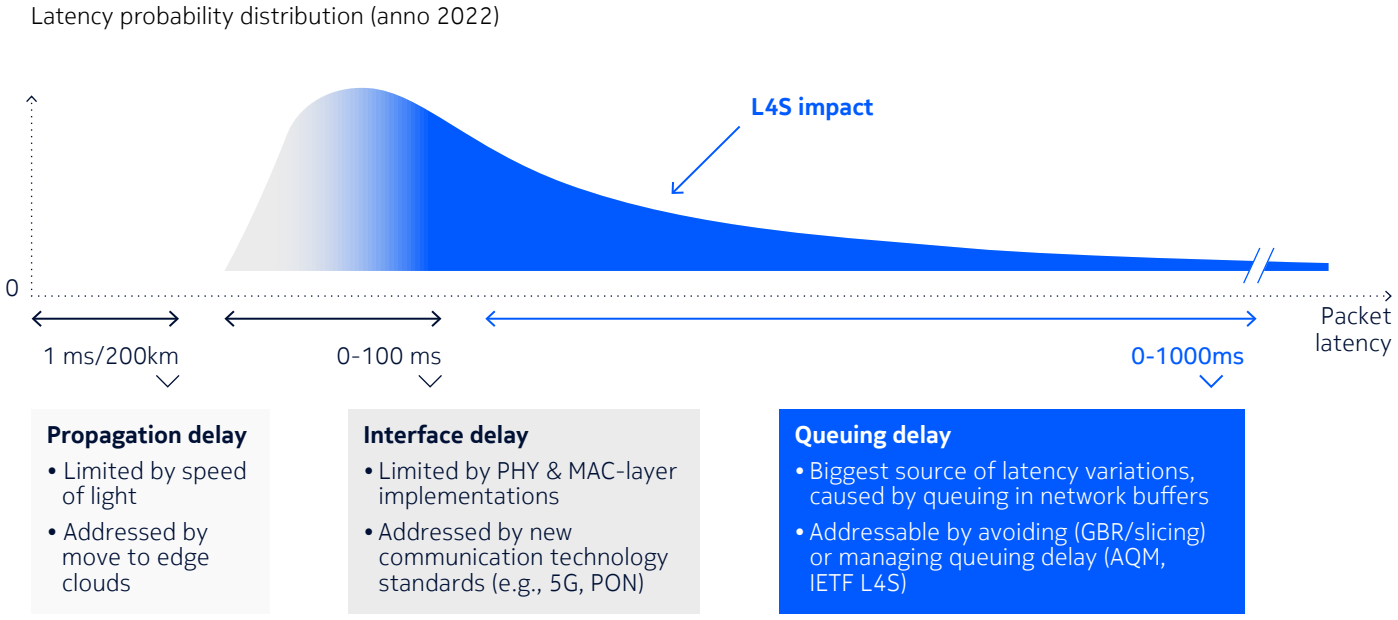
Contents

Causes of network latency

The latency or delay experienced by a data packet when it travels across the internet depends on several factors, which are illustrated in Figure 1. These are:

- The distance it needs to travel (propagation delay).
- The data processing of the underlying communication technologies (interface delay), which takes a finite amount of time to do the following:
  - aggregate data for bulk processing and/or burst transmission (aggregation delay);
  - process data (processing delay);
  - schedule transmit opportunities on a shared medium (media acquisition delay); and finally
  - send the data over the medium (serialization delay).
- The number of routers or switches it passes through, where data packets spend idle time waiting in buffers before being forwarded (queuing delay).

Figure 1: A typical latency probability distribution experienced by packets traveling across today's internet. L4S addresses the long tail of that distribution, arising from queuing delay caused by packets idly waiting in buffers before being forwarded.



Propagation delay can be reduced by bringing the computing resources or data centers closer to the end user in so-called edge clouds. For example, locating data centers within 100 km of the end user ensures a round-trip propagation delay of less than 1 ms. Interface delay is being reduced to sub-millisecond levels by design improvements in the latest technology standards, such as 5G NR and XGS-PON technology. But simply reducing the propagation and interface delays is not sufficient to guarantee a high quality of experience in interactive, low-latency applications. The most dominant source of latency – network queuing delay – needs to be tackled as well.

This is exactly what L4S does. L4S makes it possible to reduce the network queuing delay to negligible near-zero values. It accomplishes this using innovations in how these queues are managed (i.e., active queue management) and how congestion is being handled by traffic sources (i.e., congestion control).

In the network, two primary phenomena cause queuing delay:

- peak traffic concentration
- time-varying link capacity

Peak traffic concentration happens when aggregate traffic from different flows exceeds the outbound link capacity, which causes queueing. Time variations in link capacity especially occur in wireless technologies (both mobile and Wi-Fi). For example, when a mobile user on a city street walks around a corner, the wireless link capacity can suddenly decrease. This leads to a queuing delay in the data traffic destined for that user, because it is held up by the sudden capacity bottleneck on the wireless link.

Rate-adaptivity and congestion control

Popular transport protocols like TCP and QUIC allow rate-adaptive applications to adapt their sending behavior based on observations like packet loss, which occurs when packets are dropped when buffers overflow, or variations in round-trip time. These signals enable applications to discover the capacity limits of the end-to-end connection and adapt their sending rate accordingly. Typical examples are file downloads, whose connections try to reach the highest possible throughput, or adaptive voice or video streaming, which switches between higher and lower target codec bitrates depending on in-band measurements.

TCP and QUIC maintain a so-called congestion window. This congestion window limits how many unacknowledged[1] data packets can be traveling across the network at a given time and hence controls the rate at which an application can send traffic. A congestion-control algorithm is used to increase or decrease this congestion window, usually once every round-trip time (RTT). The most widely used congestion-control algorithms (“Reno” and “CUBIC”) increase the congestion window exponentially fast during the first “link initialization” phase. Once a packet is lost, it transitions into a state called “congestion avoidance.” In this state, the congestion window is gradually increased by adding a small additive increment every RTT, and is multiplicatively decreased by a constant factor whenever a data packet is lost (e.g., x0.5). This is illustrated in Figure 2. The result of this congestion-control method is that the connections can gradually discover the maximum achievable data rate and buffer size.

1 These protocols use an acknowledgement mechanism whereby the receiver notifies the sender when it successfully receives a data packet.

Figure 2: Illustration of the trade-off between link utilization (bottom) and latency (top). Small buffers keep packet latency low at the expense of link underutilization (left part), while large buffers increase link utilization at the expense of latency (right part).

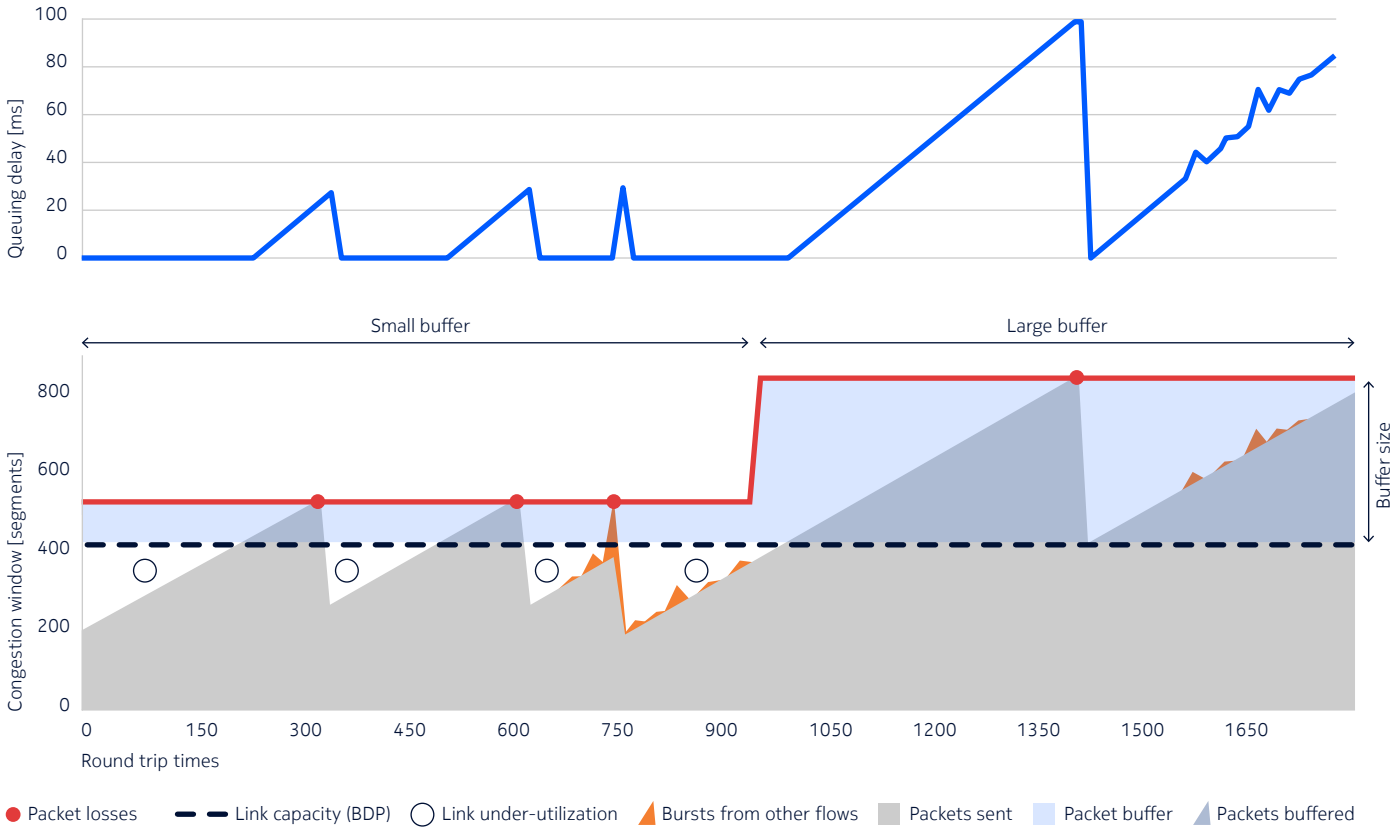


Figure 2 shows how large queues or buffers allow link utilization to be maximized by completely filling the available link capacity with data packets. This imposes a choice between two extremes: either a high link utilization at the cost of high latency, or a low latency at the cost of low link utilization. Any other flows traversing this same link – even short inelastic flows that don't significantly contribute to the overall queueing delay – are forced to experience the large queueing delays caused by the "queue-building" elastic flows. This is depicted in Figure 2 by the small orange traffic bursts. Today's propagation and interface delays towards data centers are often less than 20 ms. Excessive queuing latencies are the main reason why interactive user experiences are degraded whenever there is network congestion.

Core concepts of L4S

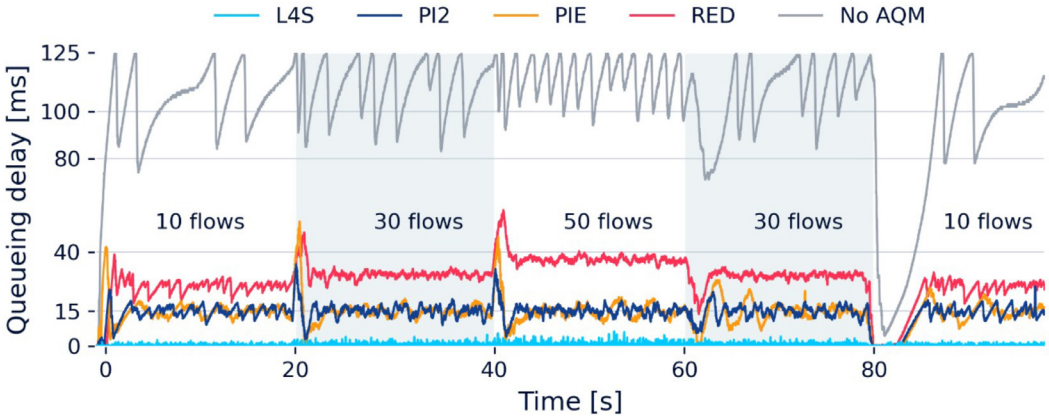
Active Queue Management (AQM)

To prevent buffers from overflowing, the queue size can be manipulated by proactively dropping a packet before the buffer is completely full. This technique is called Active Queue Management, or AQM. AQM algorithms perform early packet drops, driving the queue occupancy to stay close to a target value, based on either queue size or queuing delay (for more background on AQM algorithms, please consult the appendix).

Figure 3 compares the performance of different AQM techniques, showing the queuing delay experienced by packets sent over a fixed-capacity link. The traffic flows are all "rate-adaptive," aiming to obtain a maximum throughput and hence saturate the available link capacity. The number of active flows is changed during the measurement (a common situation on the internet) to show how the queuing delay is impacted by variations in the number of active flows.

Without any AQM on the link, traffic experiences very high queuing delays (gray line). Applying a simple AQM algorithm like Random Early Drop (RED [RFC2309], red line), significantly improves the queuing delay, but still exhibits significant jitter and a strong dependency on variations in the number of active flows. More advanced AQM algorithms like Proportional Integral Enhanced (PIE [RFC8033], orange line) and Nokia Bell Labs' PI squared (PI2 [PI2], [RFC9332], dark blue line) effectively control the queuing delay so that it hovers around a configurable constant value, regardless of the number of flows.

Figure 3: Illustration of the performance (in terms of queuing delay) of different AQM algorithms (no AQM, RED, PIE, PI2) and L4S. L4S flows can operate with sub-ms queues, without sacrificing throughput.



AQMs significantly reduce the queuing latency, but the traffic still exhibits a significant amount of jitter. This variation in packet delay can only be removed by reconsidering the congestion control algorithms at the sender and receiver, as done by L4S. Besides this jitter and the AQM-induced packet loss, current internet congestion controls also still need a decent-sized queue to properly work, for a variety of reasons. In fact the bigger the queue, the better the congestion controls typically work on all fronts except for latency.

By adding new congestion control algorithms (see section 3.2), L4S can achieve sub-ms queueing delays, regardless of the number of flows or their RTTs (shown as the light blue line in Figure 3). L4S congestion control relies on simplified AQMs that instantly respond to a queue build-up with a very low threshold (e.g., 1 ms). L4S AQMs don't need to smooth, delay, or calculate anything, instead they simply mark

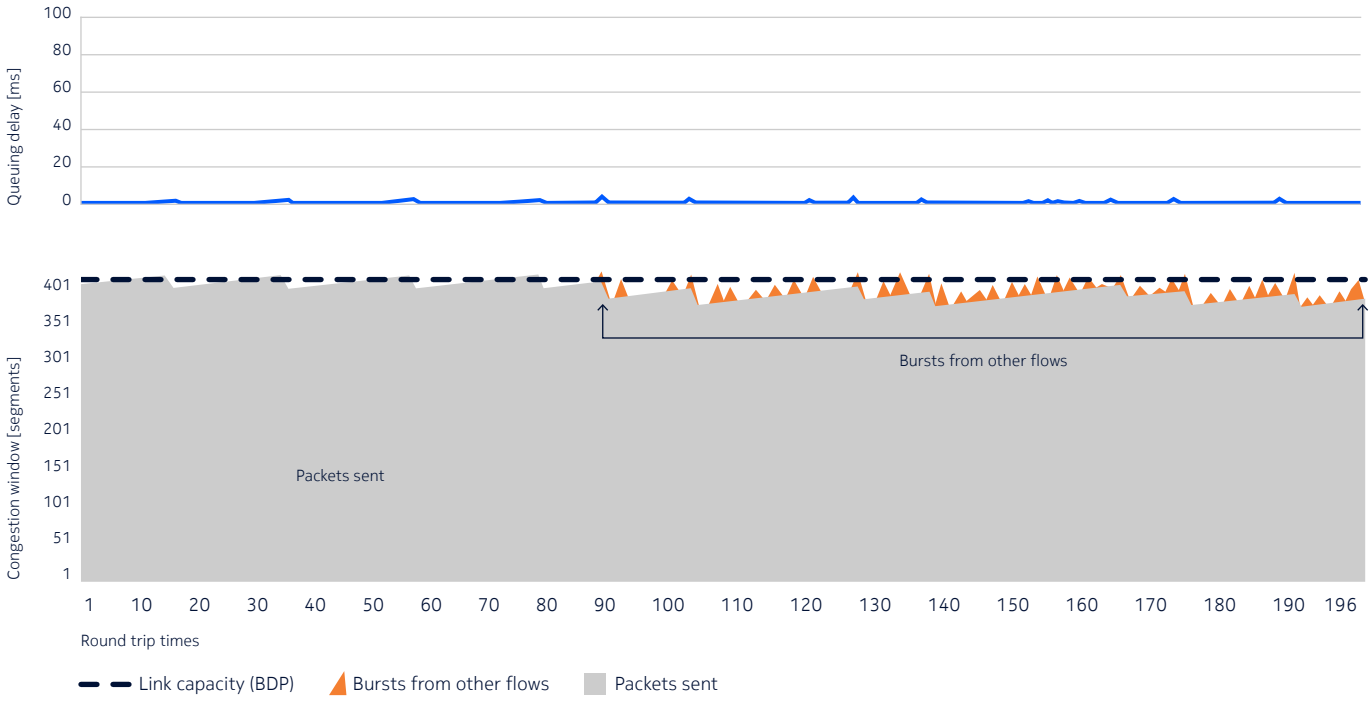
(not drop) packets that cannot be handled in time. The L4S congestion control (i.e., the source of the L4S traffic) handles any trade-offs with a maximum amount of knowledge at its disposal, both on the application as well as the end-to-end link.

Scalable congestion control with Prague congestion control

The new congestion-control algorithm leveraged by L4S is called Prague congestion control, a so-called "scalable" congestion-control algorithm that uses packet marking. When defining this congestion control, it was ensured that the network need not build any queues, delays, or impose loss for it to work well. With this algorithm, packets are "marked" instead of dropped to signal congestion to the sender. This leverages two bits inside the IP header called the Explicit Congestion Notification (ECN) bits. The ECN bits of a packet can be toggled by an AQM algorithm to signal that this packet has experienced congestion somewhere along the end-to-end path. This allows the packet carrying the congestion signal to continue its travel, avoiding the impairment of packet loss.

As opposed to classic congestion-control algorithms, where the time between congestion signals from the network decreases with increasing data rate,[2] scalable congestion-control algorithms exploit a near-constant rate of congestion signals per RTT.[3] This allows the congestion-signaling rate of scalable congestion controllers to be independent of the achievable throughput of the flow, and to ensure that its adjustments to the congestion window are as small as possible. Prague congestion control enables a very smooth throughput, which ignores unavoidable rare latency spikes but avoids frequent latency spikes [Resolving-CC-Tensions]. The requirements for this L4S scalable congestion control algorithm are published in the IETF "Prague" RFC 9331 [RFC9331] and the algorithm in [Prague].

Figure 4: The L4S Prague congestion control allows the use of very shallow buffers without sacrificing link utilization (as opposed to classic congestion control, which is shown in Figure 2).



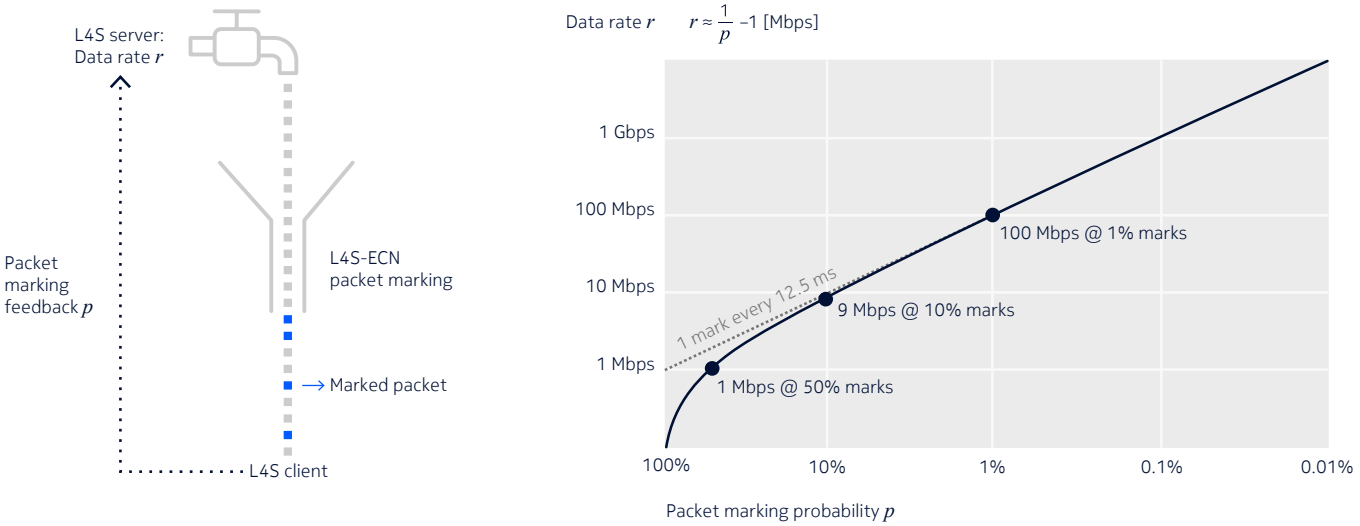
2 This can be compared to driving a car at a very high speed, but only being allowed to briefly open your eyes every 10 seconds.

3 This is enabled by using CE marking instead of packet drops, since this method is not susceptible to introducing retransmission delay and overhead as caused by packet drops.

Figure 4 illustrates how this new type of scalable Prague congestion control allows to tightly control the flow of data, so that no queues are being built up inside the network. When throughput and link capacity are stable with no or very few short, bursty flows sharing the link, the link utilization is always close to maximum without any significant queue (and hence, latency) buildup. In the presence of short bursty flows (orange), the congestion window or data rate (dark blue) is reduced depending on the frequency of their occurrence, avoiding the otherwise frequent latency spikes they would cause for the low-latency L4S flow.

The Prague congestion control also resolves another issue with standard congestion controls. It provides a scalable control signal that can control rates from near-zero all the way up to infinite throughputs, as shown in Figure 5. The amount of ECN packet marks fully control the server sending rate. The number of observed ECN packet marks is continuously echoed by the client to the server, and can occur anywhere on the end-to-end network link without requiring any coordination.

Figure 5: The Prague congestion-control algorithm fully controls the sending rate of the server in response to the fraction of observed ECN marks on received packets, as echoed by the client. The L4S flow data rate is inversely proportional to the fraction of packets that are marked, allowing L4S flow data rates to be easily throttled from sub-Mbps to infinity.



Low latency, low loss, scalable throughput (L4S)

A first implementation for an L4S congestion control originated in the data center world, where it is used for supporting high-throughput, low-latency flows. It is commonly referred to as "Data Center TCP" [DCTCP]. It significantly reduced the amplitude of the sawtooth behavior shown in Figure 2 by leveraging frequent ECN feedback, removing one of the motivations to build a big queue in the network. Under certain conditions, it also exhibited a rate-independent ECN feedback frequency, one of the desired scalable congestion-control properties. But it still required the network to build a queue to work well, for many other reasons. Before the advent of L4S, these useful properties of [DCTCP] were confined to data centers. If it was deployed on the public Internet, it would push away other "classic" TCP data traffic (i.e., reduce their data rate to very low values). This made it impossible to deploy DCTCP-like congestion controls on the public Internet.

The solution to this deployment problem is Nokia Bell Labs' Dual Queue Coupled AQM (DualQ and DualPI2) network architecture defined in RFC 9332 [RFC9332], which makes it possible to port these low-latency-enabling mechanisms from the data center environment to the public Internet domain. L4S also introduces additional requirements for the congestion-control algorithms, standardized as the "Prague" requirements in RFC 9331 [RFC9331]. These eliminated all the remaining reasons for needing queue buildup in the network (like, e.g., RTT independence, rate pacing, limiting traffic burstiness, or the use of fractional congestion windows). Additional requirements in the Prague RFC ensure compatibility with the existing Internet when no L4S support is available in the network.

Another important aspect of the L4S architecture is that L4S flows need to explicitly identify themselves. To achieve this, a repurposed ECN-capable traffic codepoint is used in the ECN bits of the IP header, ECT(1), as shown in Table 1. This ECN-based identification has been shown to be the least susceptible to "bleaching" (overwriting/clearing) on the public Internet [Mandalari18], and is able to survive propagation through tunnel encapsulation/decapsulation (e.g., GTP, L2TP) [RFC6040], [draft-ietf-tsvwg-ecn-encap-guidelines].

Table 1: L4S support is indicated (ECT(1)), and congestion is marked (CE), by leveraging two ECN bit codepoints inside the IP header, highlighted in blue (IPv4: in the "Type of Service" field; IPv6: in the "Traffic Class" field).

ECN Value	Codepoint name	Meaning
00	Non-ECT	Non-ECN-capable transport
10	ECT(0)	ECN-capable transport
01	ECT(1)	L4S-capable transport
11	CE	L4S packet that experienced congestion

The L4S architecture also separates low-latency L4S traffic and classic traffic queues using multi-queue AQMs, with at least one queue for L4S traffic and one queue for classic traffic. This enables coexistence by allowing for different congestion signaling in each queue. In doing so, L4S flows are not able to starve classic TCP flows and the L4S queues are prevented from being overrun by classic traffic, protecting the low queuing latency of L4S.

Link-capacity sharing between L4S and classic traffic can either be automatically and dynamically managed through coupled AQMs [RFC9332], [DualPI2], or statically allocated using weighted round robin or via bandwidth reservation and traffic-management mechanisms that are also used for DiffServ. For example, in coupled dual-queue AQMs, the AQM of the L4S queue increases its congestion signaling proportional to that of the classic queue to force L4S flows to leave room (in the form of link capacity) for classic flows. For more detailed information, please see [RFC9332] and [DualPI2].

Figure 6 illustrates how the coupled dual-queue AQM for L4S and classic traffic can ensure full link utilization, while guaranteeing an extremely low queuing delay or latency for L4S traffic. This is a simulation of a wireless 5G setting, with instantaneously varying channel capacities on each link/flow. Spectral efficiencies are shown in the top plot. Initially, only five classic flows (shown in shades of orange) are active and fully saturate the available spectrum. Due to the classic tail-drop scheme , they experience a queueing latency of ~100 ms. At around t = 4 sec, five L4S flows start up (shown in shades of blue) and quickly acquire their **(flow-fair)** share of bandwidth. They do so without introducing any queueing delay for themselves. At around t = 6 sec, all but one of the classic flows stop, enabling the L4S flows to gradually acquire the vacated link capacity, while the only remaining best-effort flow fills up the vacated link capacity (because of the large queue it had). At around t = 8 sec, the L4S flows again make room for four new classic flows that are started up again, without ever resulting in a significant queueing delay for the L4S flows.

Figure 6: Illustration of the ability of the coupled dual-queue AQM to ensure full link capacity on a mix of L4S and classic (non-L4S) flows (blue: L4S; orange: classic). The dual-queue AQM effectively acts as a permeable membrane achieving latency isolation without bandwidth reservation, and is flow-fair w.r.t. capacity sharing across L4S and non-L4S traffic flows.

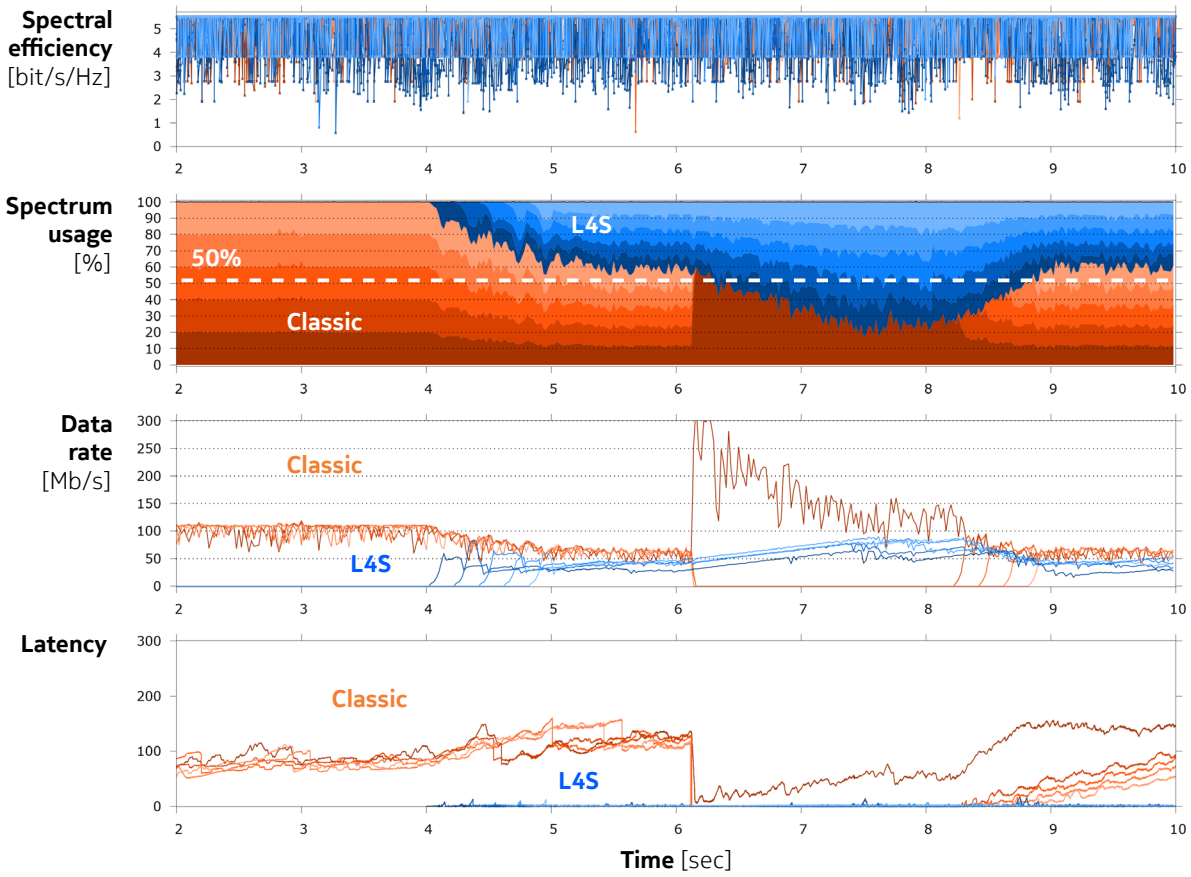
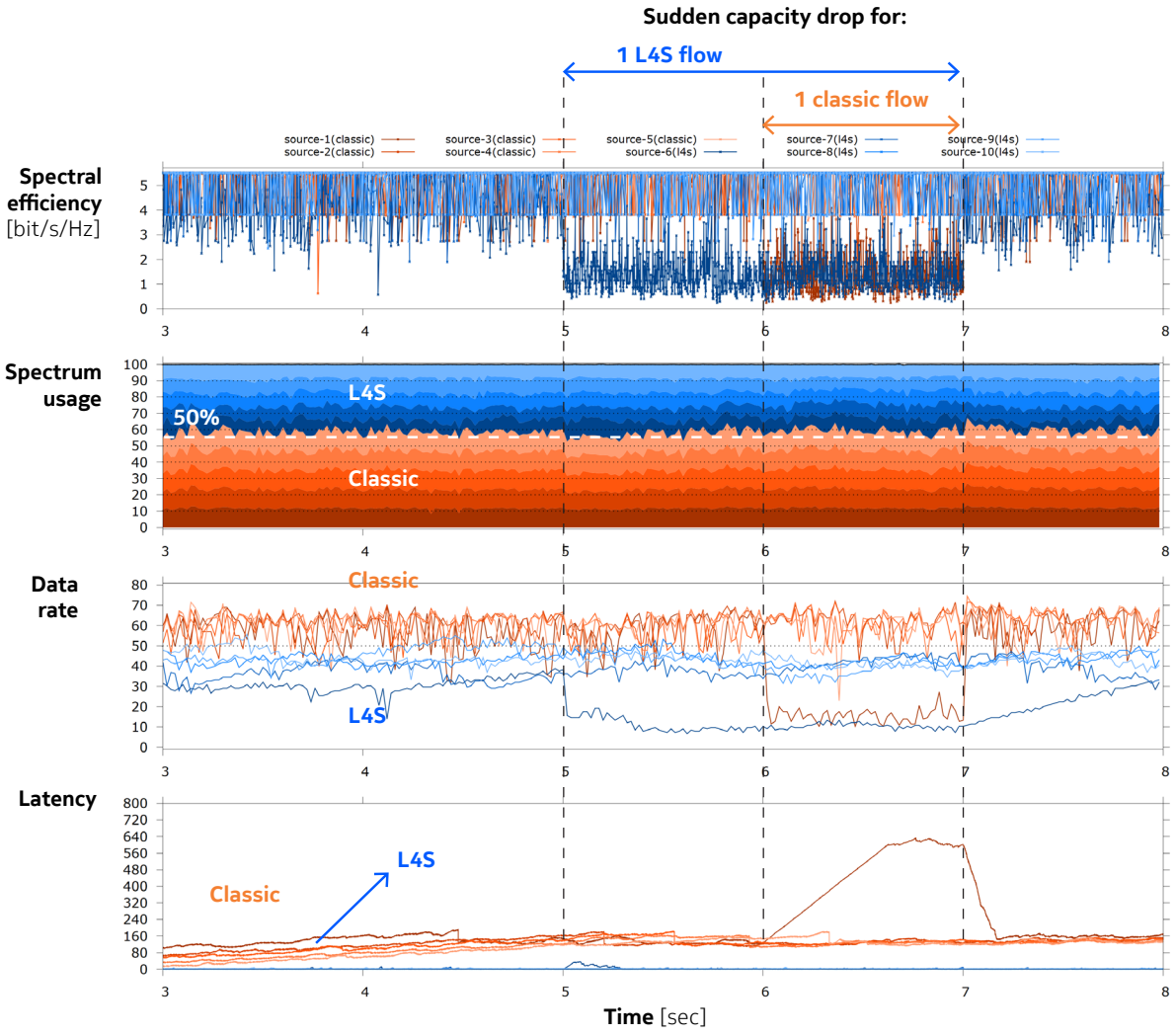


Figure 7 illustrates how L4S can maintain very low latencies, even when faced with sudden drops in link capacity. This is especially relevant for wireless technologies like Wi-Fi and 5G. The figure shows five classic and five L4S flows that are simultaneously active over 10 separate wireless channels. A sudden capacity drop is introduced on one of the L4S flows (between t = 5 sec and t = 7 sec). For the sake of comparison, a sudden capacity drop is also introduced for one of the classic flows (between t = 6 sec and t = 7 sec). The L4S flow can reduce its data rate while maintaining a low latency, while the classic flow builds up a huge queuing latency (on top of its high base latency) in response to the capacity crunch.

Figure 7: Illustration of the ability of L4S to consistently achieve low queuing latencies, even with highly time-varying channel capacities. Same simulation setting as Figure 6, but with a sudden channel capacity drop for one L4S flow (from t = 5 sec to t = 7 sec) and one classic flow (from t = 6 sec to t = 7 sec), creating sudden congestion on those links, with a negligible latency impact for L4S flows and a very significant impact for classic flows.



Deploying L4S services

L4S requires two parts to work: 1) senders and receivers with L4S-capable congestion control in the application; and 2) L4S AQMs and isolation mechanisms deployed in the bottleneck node(s) on the end-to-end path within the network. It is not sufficient to have only one of these two parts. Application providers and network operators, therefore, each hold one half of the key to enabling the performance benefits of L4S.

Supporting L4S on an application

Supporting L4S on an application requires the application to comply with the Prague requirements. Interactive application developers often use proprietary congestion controls over UDP, which can be updated in their own client and server applications. These can be made L4S-compatible by the application provider by implementing a Prague congestion-control algorithm on top of UDP. This does not require any L4S support in the underlying operating system (provided ECN bits can be read and set).

Prague congestion control can also be supported within the Operating System, relieving the application developer of its implementation. Nokia Bell Labs is a main contributor to the open-source TCP Prague implementation, enabling support in Linux OS [TCP Prague]. Apple has also recently announced support for Prague congestion control at their 2023 Worldwide Developer Congress [Apple-WWDC2023]. Other companies, including NVIDIA (Geforce NOW congestion control), Google (BBRv2 congestion control [BBRv2]), and Ericsson (Self-Clocked Rate Adaptation for Multimedia [SCReAM]) have also been trialing implementations of an L4S Prague-compliant congestion-control algorithm at recent IETF interop events.

The application or transport protocol is responsible for echoing the CE marks applied in the NW back to the sender. Specifications for echoing this info in standardized transport layer protocols are available as IETF RFCs or drafts in their final stage ([draft-ietf-tcpm-accurate-ecn] for TCP, [RFC9000] for QUIC, [RFC8888] for RTP).

It's important to note that without L4S support in the network bottleneck, the L4S traffic will simply behave like regular traffic. In other words, there will be no benefit, but also no disadvantage, to activating L4S in the application. At the application clients and servers, L4S can therefore be enabled independently for all users and network devices without requiring a gradual per-user configuration.

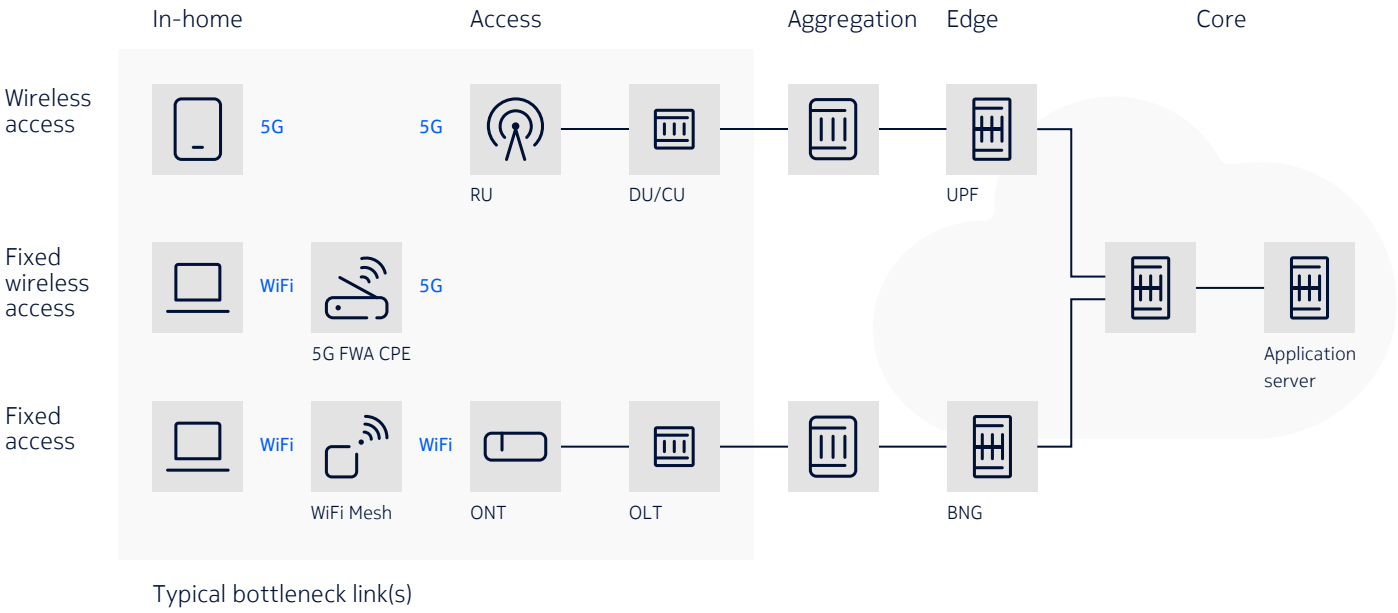
Supporting L4S in a network

L4S network operation relies on the ECN bits present in the IPv4/IPv6 headers, as well as on transport-specific feedback mechanisms. As such, devices forwarding or aggregating IP packets should properly handle the ECN bits.

Devices tunneling L4S packets need to follow the existing guidelines on encapsulation and decapsulation operations. To do so, they need to propagate inner-outer ECN values at tunnel boundaries (see [RFC6040], [draft-ietf-tsvwg-rfc6040update-shim] for details). Devices that process open transport protocol packets also need to handle the transport-layer information related to ECN, such as the TCP flags and options [draft-ietf-tcpm-accurate-ecn] or Real-time Transport Control Protocol (RTCP) feedback messages [RFC8888] and [RFC6679].

While congestion can theoretically happen anywhere in a packet-switched network, it is usually concentrated at a few specific locations identifiable straight from the network design. Support is only required in the "bottleneck" link(s) on the end-to-end path, which is typically the access or the in-home Wi-Fi network (see Figure 8). An L4S-capable network design should ensure that throughput bottlenecks are minimal and are created there where products support L4S.

Figure 8: Bottleneck links in a CSP network are typically located in the access network and/or the in-home Wi-Fi network.



L4S support in IP-transport

IP-transport networks are typically over-dimensioned and not very congested. As jitter in this part of the network would therefore typically be negligible, special measures to support low-latency L4S traffic flows are not required. In case it would be required (e.g., a congested peering or aggregation router), it can be handled through proper prioritization of L4S traffic over classic traffic, which will suppress transient latency spikes. Note that this would make the IP-transport network "L4S-capable," without having to discriminate on a per-user basis or apply ECN marking. Broadband network gateway (BNG) and user plane function (UPF) nodes can identify and treat L4S traffic as a separate service with separate queues, throughput policies and assign it a low-latency service.

L4S support in Mobile Access

In 5G networks, the Radio Access Network (RAN) is the biggest bottleneck requiring L4S support. In 3GPP, L4S has been adopted as a rate-adaptation mechanism in supporting XR traffic in 3GPP Rel-18 TS 23.501. Performance-wise, the best L4S implementation in the 3GPP architecture involves performing ECN marking in the RAN for both upstream and downstream traffic. The RAN has the best awareness of congestion occurring in the Radio bottleneck in both directions and is the optimal location to perform packet marking. Marking in the UPF rather than the RAN would incur an extra delay for downstream traffic and also require extra congestion-reporting and capability negotiation protocols between the RAN and the UPF.

As a scalable mechanism for congestion control and rate-adaptation, L4S should be considered a technology feature that enables large-scale service offerings of real-time applications, such as wide-area video-conferencing, cloud-gaming, extended reality, or remote control. Ultra-Reliable-Low-Latency Communications (URLLC), on the other hand, leverages different approaches like Guaranteed-Bit-Rate and Time-Sensitive Networking as defined in 3GPP Rel-16, Rel-17 and Rel-18. URLLC is well suited to providing strict and guaranteed end-to-end SLAs in well-controlled areas, where proper system dimensioning can ensure appropriate support. L4S could in principle also be used in combination with URLLC. L4S would then provide an extra safeguard in case of drastic link-capacity or service crunches, flash crowds, or the ability to opportunistically use more throughput whenever available.

For fixed wireless access, L4S support would be required both on the mobile RAN as well as the Wi-Fi access or LAN point of the fixed wireless access CPE.

L4S support in Fixed Access

In fixed access deployments including in-home Wi-Fi, the Wi-Fi link would be the primary bottleneck in which L4S support is required. The second bottleneck that would benefit from L4S support is the fixed access network.

A notable difference between wired and wireless technologies (both mobile and Wi-Fi) is that the link capacity in wired technologies hardly varies over time. In a passive optical network, for example, the capacity available on the optical fiber does not vary at all. Therefore, in wired technologies, the ECN marking functionality of L4S is mainly required to handle traffic congestion and avoid throughput starvation of classic (non-L4S) traffic. In wireless technologies, on the other hand, it also serves to handle possible congestion caused by link-capacity variations.

Summary

With the publication of the IETF RFCs in January 2023, L4S, the next-generation IETF internet-protocol innovation pioneered by Nokia Bell Labs, is getting ready for prime time. L4S reduces the single biggest, and most often overlooked, source of latency and latency variation or jitter – queuing delay – to near-zero values. It also minimizes data packet loss by using packet marks (leveraging the ECN bits in the IP header) instead of packet drops to signal network congestion.

L4S promises to eliminate large and varying delays that have a negative impact on the user experience, such as videos that stall, frames that are skipped in cloud-gaming applications, or delays that cause people to start talking over one another in videoconferencing.

With L4S, an operator will be able to tele-operate a crane in a port without any disruption. Kids will be able to play their games smoothly in the cloud over any network. And XR headset designers will have much more design freedom. Nowadays headsets are still bulky as all processing needs to happen on the device itself. But when the network can guarantee a low latency in all conditions, whether at home or on the move, more processing can be offloaded to the cloud. And at this moment new XR headset designs inspired by regular glasses come within reach.

And, as a technology-agnostic solution, L4S will also simplify large-scale deployments of low-latency, real-time services by offering a uniform mechanism across all different network communication technologies. Any “real-time” application, which functions within a time frame that the user senses as immediate, greatly benefits from an end-to-end latency that’s not only low, but consistently low – no matter how busy or congested the network is.

Abbreviations

AQM	Active Queue Management
BNG	Broadband Network Gateway
BBR	Bottleneck Bandwidth and Roundtrip time
BDP	Bandwidth-Delay Product
CE	Congestion Experienced (code point)
CPE	Customer Premises Equipment
CSP	Communication Service Provider
DCTCP	Data Center Transmission Control Protocol
ECN	Explicit Congestion Notification
FWA	Fixed Wireless Access
GTP	GPRS Tunneling Protocol
IP	Internet Protocol
L2TP	Layer 2 Tunneling Protocol
L4S	Low Latency and Low Loss Scalable throughput
PON	Passive Optical Network
QUIC	Quick UDP Internet Connections
RED	Random Early Drop
RTCP	Real-time Transport Control Protocol
RTT	Round Trip Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URLLC	Ultra-Reliable Low Latency Communications

Appendix: additional background information

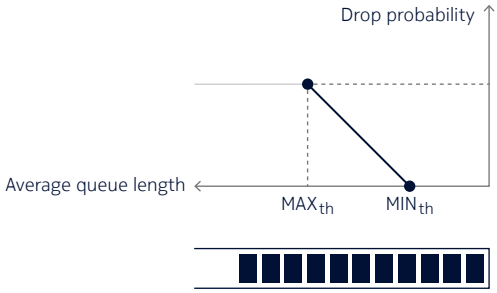
AQM algorithms

There exist many AQM algorithms, the most popular ones being Random Early Drop (RED, [RFC2309]) and Controlled Delay (CoDel, [RFC8289], which are often paired with per-flow queuing (fq_codel [RFC8290]) and Proportional Integral Enhanced (PIE, [RFC8033]). These AQM algorithms are discussed and compared below. We also introduce a more recent AQM scheme created by Nokia Bell Labs that solves many of the drawbacks of the older AQM algorithms, called PI2 (pronounced "pie-square") [PI2]. PI2 is the AQM that is leveraged in the L4S dual-queue AQM called "DualPI2" [RFC9332].

Random Early Drop (RED)

The RED AQM algorithm monitors the average queue size and randomly drops incoming packets according to a certain drop probability (or mark probability, when used in conjunction with ECN). This drop probability increases linearly from 0 to 1 between a preconfigured minimum average queue length threshold value MINth and a maximum average queue length value MAXth, as illustrated in Figure 9.

Figure 9: Illustration of the drop probability used by the RED AQM algorithm as a function of the monitored average queue size.



Since the RED algorithm uses average queue length as the only congestion indicator, its latency performance varies widely depending on operating conditions, such as the number of active flows that traverse the queue (as visible in Figure 3).

RED requires the operator to carefully tune the algorithm slope-of-drop probability vs. queue size (in bytes) and an appropriate averaging/smoothing factor, whose optimal settings depend on several factors, including the underlying link rate, the number of flows, and the flow RTTs. If the slope is too steep (or the smoothing factor is too low) the system will start to oscillate. If the slope is shallow, it will result in large queues, and a high smoothing time will result in large queue overshoots (i.e., latency spikes that are not controlled in a timely manner by a packet drop).

Controlled DELay (CoDel)

CoDel, which is pronounced "coddle," takes as input a target queuing delay (a.k.a. packet sojourn time), which automatically abstracts the link rate. This makes it well suited for links with varying throughput (e.g. radio) and easier to configure. CoDel was designed to improve on the performance of RED by addressing some of its shortcomings. Instead of monitoring the average queuing delay, which varies greatly in bursty traffic scenarios, it monitors the minimum queuing delay that packets experience during a sliding time window.

Packets are dropped when the observed minimum delay crosses a certain threshold value. This threshold is 5 ms by default (5% of a "typical" RTT interval of 100 ms), at a frequency that gradually increases until the delay again drops below this value (i.e., CoDel computes a drop schedule). This method has several advantages, like being parameterless (which means there is nothing to configure), adapting to dynamically changing link rates without having an impact on link utilization, and having a low implementation complexity.

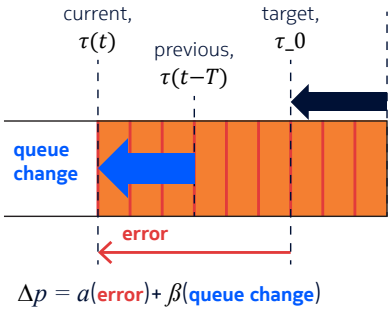
However, as it is a heuristic algorithm specifically designed for a single TCP-Reno flow, CoDel does not work effectively with an aggregate of more than one flow and any congestion controllers other than Reno. This is often obscured by being paired with per-flow queuing, as it compensates for CoDel's poor performance by dynamically prioritizing flows based on their respective share of bandwidth. For example, pings will always be prioritized over downloads as they use less bandwidth.

Therefore, CoDel is only deployed when it is combined with per-flow queuing (FQ_CoDel) [RFC8290]. Like RED, FQ_CoDel's performance becomes unpredictable when subjected to multiple flows per queue or non-elastic flows. Also, when an application flow itself requires a low latency, FQ_CoDel might not always be able to control the latency in the flow's own queue (for instance if the congestion control is non-Reno).

PI, PIE and PI2

PI is a classic Proportional Integral controller, like a thermostat, which aims to control packet sojourn times to be around a given target delay. It has a proportional factor to swiftly react to sudden queue growth, and an integral factor (tracking the queue change over time) enabling it to ramp up faster if needed, or to smooth out transient bursts. It periodically inspects the queue it is controlling and updates an internal random drop probability based on the sojourn time of the oldest packet in the queue (i.e., the head), as illustrated in Figure 10.

Figure 10: PI AQM algorithm, operating like a standard proportional-integral controller (figure from [PI2])



A key drawback of PI is that it works correctly in only a limited range of conditions. The reason for this is that congestion control is not a linear control process. When the congestion level is high, throughput rates need to be kept low and a very high signal needs to be sent. When it's low, a non-proportional lower signal is required. As a first attempt to handle this issue, Cisco developed PIE (PI Enhanced). Although it performed much better than RED and CoDel, a key drawback of PIE is that it uses several heuristics to dynamically configure the proportional and integral factors. This requires the use of large lookup tables, which introduces complexity and hampers performance and stability.

PI2 is at its core a basic proportional-integral controller. But PI2 differs dramatically from PIE in the way the drop probability is used within the AQM algorithm. As a result, PI2 has superior stability and responsiveness compared to PIE, while also being less complex to implement. More specifically, it does this by directly eliminating the nonlinearities in the control equation. PIE attempts to compensate for the nonlinearities using several heuristics, as well as various scaling factors for its α and β parameters. PI2, instead, squares its internal drop probability before using it to decide whether a given packet should be dropped or not, to compensate for the square root non-linearity of classic congestion controllers.

Figure 3 shows an example comparing the relative stability between PI2 and PIE, where the arrival of new flows drives PIE to its unstable region due to the large probability changes required to sustain the added load, which accounts for the delay spikes. In comparison, PI2 can keep this slow-start spike to acceptable levels. Note that 20 new flows translate to a burst of 200 packets before they each enable congestion control.

Immediate vs. smoothing AQMs

Typical legacy AQMs are known as "smoothing" AQMs, which means that they apply their packet drops in a probabilistic manner, which is a function of observations over a time interval spanning many RTTs. This compensates for the expected multiplicative decrease of the sending rate applied by classical TCP senders as a reaction to packet loss(es). Therefore, their operation can be somewhat sluggish depending on its configuration.

On the other hand, immediate AQMs don't use a probabilistic drop/mark function but an all-or-nothing drop function. This means that they will drop/mark all packets that satisfy a certain criterion, such as those that have a packet sojourn time longer than a certain threshold. This "immediate" AQM type is the one leveraged in L4S queues and is typically only used for marking packets, and not for dropping packets. The smoothing is then performed by the end-system itself, empowering it to choose between immediate response (e.g., to a sudden burst of marks indicating a capacity reduction/overshoot) or a delayed/lower one (e.g., to filter out transient events).

Further background on Data Center TCP

Data Center TCP consists of three novel components:

- DCTCP simplifies the in-network AQMs to behave as steps: every packet resulting in a queue size above the set threshold should be marked. This removes the need for smoothing the signal within the network, and vastly simplifies the implementation of AQMs (e.g., no timers, stateless).
- It defines an alternative way for a data receiver to reflect ECN signals to the data sender, such that the sender can accurately count how many "Congestion Encountered" (CE) marks were received during 1 Round Trip Time (RTT). This contrasts with classic TCP [RFC3168], which only reflects that some marks were received.
- Finally, with this more accurate ECN information, the DCTCP congestion controller is able to adjust its congestion window based on the extent of congestion (i.e., proportionally to the number of observed marks), rather than based on the mere binary presence of congestion, as is the case with RFC3168-based congestion controllers. CUBIC, for example, reduces its congestion window by 30% if it receives one or more marks over a period of 1 RTT.

How does L4S react to packet drops?

Using ECN to signal congestion implies that L4S flows only see packet drops in three cases, which simplifies their interpretation when designing a congestion control algorithm:

- The drop is due to network issues (e.g., corruption, a noisy air interface giving up link-local retransmission)
- The drop indicates that the network is overloaded; that is, its queues are so full it can no longer afford to mark packets to signal congestion and is forced to aggressively drop them in a tail-drop fashion.
- The drop indicates that the network device where the congestion is happening does not support L4S.

To minimize the impact of transient losses (case 1), L4S congestion controllers can reuse the same technique as classic congestion controllers, such as filtering out spurious losses (like BBR) or reducing by a smaller fraction of the congestion window (e.g., 30% like CUBIC). To prevent congestion collapse for case 2 and case 3 above, L4S congestion controllers react to packet drops like classic congestion controllers do, (i.e., by performing a multiplicative decrease.

L4S and faster link-capacity discovery congestion control

One of the major shortcomings of both DCTCP and the Reno algorithm is their static congestion window growth function during congestion avoidance, which involves increasing the congestion window by 1 segment per RTT. While it can quickly saturate a low-Bandwidth-Delay-Product (BDP) connection (e.g., a flow traversing a 20 Mbps link with a 10 ms RTT takes up to ~160 ms to saturate it), it becomes a serious scalability issue for higher BDPs (e.g., saturating a 200 Mbps link with 50 ms base RTT takes up to 8.3 seconds). This slow capacity discovery is especially problematic when flows are operating over links with highly variable underlying throughput (e.g., 5G mmWave). Below we introduce two other congestion-control algorithms which aim to address this, and illustrate the performance of both when a flow is subject to a capacity reduction followed by a large increase. Congestion-control improvements enabling faster link-capacity discovery are also applicable to L4S, such as BBRv2 discussed below.

CUBIC

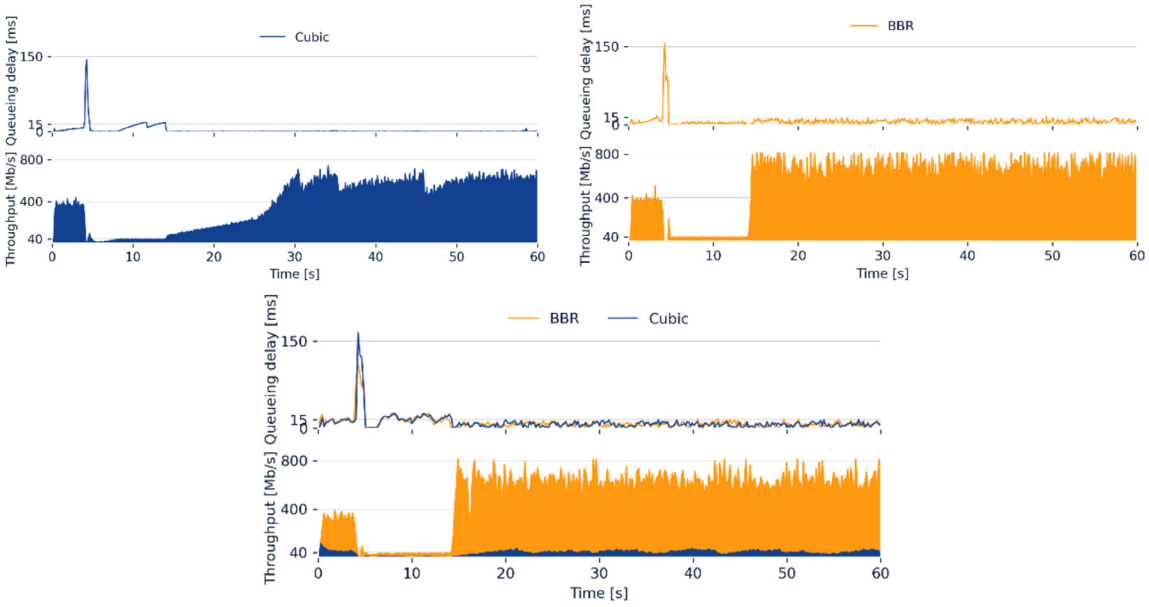
The CUBIC congestion-control algorithm [RFC8312] improves on Reno by combining a new cubic congestion window growth function with additional filtering to ignore spurious random packet losses. The longer it grows its congestion window without receiving a congestion signal from the network, the more aggressive it will become. This has been the default congestion-control algorithm on Linux since 2006.

While comparatively much more effective than Reno, CUBIC still takes tens of seconds to ramp up on link rates commonly available today. For example, a 36 ms RTT flow with Reno takes 57 seconds to saturate an 800 Mbps link after a loss, while CUBIC achieves this in 12 seconds. Although it is a welcome improvement, it is still insufficient for today's and tomorrow's link rates.

Additionally, CUBIC delivers latency performances comparable to Reno, as it still exhibits very large sawtooth behavior (referring to the fluctuation of the data rate over time, like in Figure 2)—in other words, it behaves like a classic congestion controller.

The first graph (left) in Figure 11 shows an example of a CUBIC flow traversing a path equipped with a PI2 AQM, whose underlying rate is first reduced from 400 Mbps to 40 Mbps (t = 5 sec), and then increased to 800 Mbps (t = 15 sec). The CUBIC increase is particularly visible from t = 25 sec to t = 30 sec.

Figure 11: BBR congestion-control can ramp up its data rate much faster than CUBIC congestion control, as can be seen by comparing the throughput evolution after the link-capacity crunch at t = 15 sec in the top-left and top-right figure. However, consequently, the more aggressive BBR pushes away the CUBIC flow when sharing a common bottleneck (as shown in the bottom graph).



Bottleneck Bandwidth and Round-trip time (BBR)

Bottleneck bandwidth and round-trip propagation time (BBR) is a newer congestion control mechanism developed by Google. BBR does not follow a fixed mechanism in the sender, for example by reducing the congestion window by X% when receiving a congestion signal and increasing it by Y every RTT. BBR instead creates a model of the bottleneck link rate by observing the maximum throughput/bandwidth and the minimum RTT over a period of time, and then uses this model to pace outgoing segments. This enables BBR to achieve full link utilization, and to control queue buildup to about 30 ms. BBR filters out transient latency spikes and losses, making it particularly well-suited for wireless and other types of bursty networks and robust against random losses. Finally, BBR continuously probes the network to make use of available capacity as fast as possible (even for very large BDPs).

However, BBR is unable to deliver low latency, as it relies on queue buildup to guarantee link utilization and detect available capacity. Building up a queue gradually inflates the flow RTT, which is a signal that it has saturated the link. Subjected to the same experiment as CUBIC in the second graph (right) of Figure 11, BBR can ramp up from 40 Mbps to 800 Mbps within a second (whereas CUBIC takes 15 sec).

However, as can be seen in the last graph of Figure 11, when CUBIC and BBR share the same bottleneck, BBR will use most of the capacity and CUBIC will only be able to use a small fraction of it. This is due to BBR's aggressive queue filling and loss filtering.

Newer versions of BBR have been developed that promise better compatibility with Classic loss-based congestion controls and AQMs. This will also improve the compatibility with L4S in DualPI2 AQMs.

References

[Apple-WWDC2023] https://developer.apple.com/videos/play/wwdc2023/10004/

[BBRv2] GitHub - Google/BBR, https://github.com/google/bbr

[DCTCP] M. Alizadeh et al, "Data center TCP (DCTCP)," In Proc. SIGCOMM '10, https://doi.org/10.1145/1851182.1851192

[draft-ietf-tcpm-accurate-ecn] B. Briscoe, "More Accurate ECN Feedback in TCP," https://datatracker.ietf.org/doc/draft-ietf-tcpm-accurate-ecn/

[draft-ietf-tsvwg-ecn-encap-guidelines] B Briscoe and J. Kaippallimalil, "Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP," https://datatracker.ietf.org/doc/draft-ietf-tsvwg-ecn-encap-guidelines/

[draft-ietf-tsvwg-rfc6040update-shim] B. Briscoe, "Propagating Explicit Congestion Notification Across IP Tunnel Headers Separated by a Shim," https://datatracker.ietf.org/doc/draft-ietf-tsvwg-rfc6040update-shim/

[DualPI2] K. De Schepper, O. Albisser, O. Tilmans, B. Briscoe, "Dual Queue Coupled AQM: Deployable Very Low Queuing Delay for All," https://arxiv.org/abs/2209.01078

[Mandalari18] A. Mandalari et al., "Measuring ECN++: Good News for ++, Bad News for ECN over Mobile," in Comm. Mag. 56, 3 (March 2018) https://doi.org/10.1109/MCOM.2018.1700739

[PI2] K. De Schepper et al., "PI2 : A Linearized AQM for both Classic and Scalable TCP," In Proc. CoNEXT'16, https://doi.org/10.1145/2999572.2999578

[Prague] K. De Schepper, O. Tilmans and B. Briscoe, "Prague Congestion Control," https://datatracker.ietf.org/doc/draft-briscoe-iccrg-prague-congestion-control/

[Resolving-CC-Tensions] B. Briscoe and K. De Schepper, "Resolving Tensions between Congestion Control Scaling Requirements," Discussion paper, https://arxiv.org/pdf/1904.07605.pdf

[RFC2309] D. Clark et al., "Recommendations on Queue Management and Congestion Avoidance in the Internet," https://datatracker.ietf.org/doc/rfc2309/

[RFC6040] B. Briscoe, "Tunnelling of Explicit Congestion Notification," https://datatracker.ietf.org/doc/rfc6040/

[RFC6679] M. Westerlund, "Explicit Congestion Notification (ECN) for RTP over UDP," https://datatracker.ietf.org/doc/rfc6679/

[RFC8033] R. Pan et al., "Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem," https://datatracker.ietf.org/doc/rfc8033/

[RFC8289] K. Nichols et al., "Controlled Delay Active Queue Management," https://datatracker.ietf.org/doc/rfc8289/

[RFC8290] T. Høiland-Jørgensen et al., " The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm," https://datatracker.ietf.org/doc/rfc8290/

[RFC8312] I. Rhee et al., “CUBIC for Fast Long-Distance Networks,” https://tools.ietf.org/html/rfc8312

[RFC8888] Z. Sarker et al., “RTP Control Protocol (RTCP) Feedback for Congestion Control” https://datatracker.ietf.org/doc/rfc8888/

[RFC9000] J. Iyengar et al., “QUIC: A UDP-Based Multiplexed and Secure Transport” https://datatracker.ietf.org/doc/rfc9000/

[RFC9331] K. De Schepper and B. Briscoe, “RFC 9331 The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S),” https://www.rfc-editor.org/rfc/rfc9331.html

[RFC9332] K. De Schepper, B. Briscoe and G. White, “RFC 9332 Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S),” https://www.rfc-editor.org/rfc/rfc9332.html

[SCReAM] GitHub - EricssonResearch/scream: SCReAM - Mobile optimised congestion control algorithm, https://github.com/EricssonResearch/scream

[TCP Prague] https://github.com/L4STeam/linux/blob/testing/net/ipv4/tcp_prague.c

About Nokia

At Nokia, we create technology that helps the world act together.

As a B2B technology innovation leader, we are pioneering networks that sense, think and act by leveraging our work across mobile, fixed and cloud networks. In addition, we create value with intellectual property and long-term research, led by the award-winning Nokia Bell Labs.

Service providers, enterprises and partners worldwide trust Nokia to deliver secure, reliable and sustainable networks today – and work with us to create the digital services and applications of the future.

Nokia OYJ
Karakaari 7
02610 Espoo
Finland
Tel. +358 (0) 10 44 88 000

721210 (October) CID213410