# Should we care about coexistence in L4S? Assessing round-trip time significance and low-latency for all

**Eduardo Freitas**[1], **Maria Eduarda Veras**[1], **Assis T. de Oliveira Filho**[1], **Judith Kelner**[1], **Djamel Sadok**[1]

[1] Grupo de Pesquisa em Redes e Telecomunicações (GPRT)
Centro de Informática (CIn) – UFPE
Recife, PE – Brazil

{eduardo.freitas, eduarda.martins, assis.tiago}@gprt.ufpe.br

{jk, jamel}@gprt.ufpe.br

***Abstract.*** *The L4S architecture promises to resolve the throughput-latency trade-off, yet concerns regarding its coexistence with classic TCP persist. The literature presents experimentations on how L4S flows can potentially starve non-L4S ones when deployed on networks with incomplete or no support for L4S, an already known fact. However, few works assess the L4S performance considering networks with full support for it. This work evaluates the dynamics of L4S performance on networks that fully support it and assesses that coexistence is no longer a concern and could also be considered architecturally resolved. We demonstrate that classic flows retain fair throughput with L4S flows but suffer self-inflicted queuing delays exceeding 15 ms, deteriorating their RTT.*

## 1. Introduction

The internet service landscape has shifted from a bandwidth constrained paradigm to a latency-sensitive one. Emerging applications such as Cloud Gaming, XR (Extended Reality), and remote autonomous control demand not only high throughput but, consistently low latency and jitter. While network capacity has grown exponentially – often exceeding the requirements of 8K video streams [3GPP 2025, SpeedTest 2025] – the user experience remains bound by the "bufferbloat" phenomenon and the inherent latency-throughput trade-off in classic congestion controls in transport protocols like Transmission Control Protocol (TCP) [Gettys and Nichols 2012].

To address this, the IETF has standardized the Low Latency, Low Loss, and Scalable Throughput (L4S) architecture [Briscoe et al. 2023]. L4S decouples latency from bandwidth by combining *scalable congestion controls*, such as TCP Prague, at the host with *dual-queue coupled* Active Queue Management (AQM) such as DualPI2, at the bottleneck. This architecture promises to maintain ultra-low queuing delay for L4S-compliant flows without starving legacy "classic" traffic, such as TCP Cubic, sharing the same link.

However, the transition to L4S faces the fundamental hurdle of coexistence uncertainty. While the L4S architecture mandates that scalable flows must revert to classic behavior if the network queue management does not support L4S, the reverse scenario, the behavior of mixed traffic within an active L4S bottleneck, remains a subject of debate. Network operators hesitate to deploy DualPI2 fearing that aggressive classic flows

might induce queue coupling effects that degrade the strict latency guarantees required by L4S applications and also the classic flows themselves. In addition, service providers avoid deploying L4S-compatible scalable congestion controls, fearing that competition between other L4S flows might result in "scarce" resources, increasing latency for any L4S flows [Cheshire 2025].

Existing literature focuses on the consequences of L4S flows in legacy queues or fairness issues. There are, to the best of our knowledge, few works providing an empirical assessment of Round-Trip Time (RTT) stability under aggressive multi-flow competition in real-world Linux implementations.

In this work, we bridge this gap by experimentally assessing the coexistence dynamics of L4S. We deploy a replicated testbed using the reference Linux Kernel implementations of DualPI2 and TCP Prague to evaluate performance in a constrained bottleneck scenario. We challenge the premise that L4S flows require ideal, dedicated environments, demonstrating instead that they robustly handle hostile competition.

Our main contributions are as follows:

- Isolation validation: we demonstrate that DualPI2 effectively isolates latency domains. Even under higher competition with TCP Cubic flows, L4S flows maintain a mean queue delay of $\approx$100 $\mu$s, unaffected by the 15 ms+ queue buildup of the classic flows.
- RTT scalability: we show that RTT degradation in L4S is strictly linear and dominated by physical propagation delay rather than queuing delay. Furthermore, we demonstrate that under L4S networks, there are no resource competition issues regarding latency for L4S flows. We consider RTT as a crucial metric since it measures the time for packets to traverse the network, in other words, the packet latency. And since providing low latency for transport transmissions is one of the main goals of L4S, we consider measuring RTT from the server perspective an essential way to assess latency performance.
- Throughput fairness: We confirm that switching to L4S does not penalize bandwidth usage. L4S flows achieve equitable link utilization compared to classic flows, debunking starvation concerns in coupled-queue environments.

Indeed, we provide a provocative title for our work. We understand that coexistence is a crucial concern on L4S, but we also understand that, given the appropriate network deployment, coexistence should be a supplemental concern, and attention should be directed towards congestion controls and enabling better performance for any service.

The remainder of this paper is structured as follows. Section 2 provides the technical background on the L4S architecture, specifically the interplay between DualPI2 and Scalable Congestion Controls, and reviews related literature on coexistence challenges. Section 3 details our experimental methodology, describing the replicated Linux-based testbed and the specific bottleneck constraints designed to emulate edge network conditions. Section 4 presents a comprehensive analysis of our empirical results, dissecting the impact of heterogeneous flow competition on throughput fairness, RTT stability, and queue latency. Finally, Section 5 synthesizes our findings and offers concluding remarks on the operational viability of L4S deployment.

## 2. Background and Related Works

L4S is a two-part architecture, addressing changes on the host and on the network to allow low latency, low loss, and scalable throughput [Briscoe et al. 2023]. The basic foundation of L4S is Explicit Congestion Notification (ECN). L4S uses ECN by default with an improvement on the way it works. ECN, defined in Requests for Comments (RFC) 3168, specifies that a packet marked with Congestion Experienced (CE) should be treated by the TCP sender as if it was a dropped packet, meaning a drastic congestion window decrease. This simple approach is the standard ECN definition, and it is often referred to as *classic ECN*, and the Congestion Control Algorithm (CCA) that use it as *classic TCP*.

L4S specifies that the network and the hosts treat ECN marks as a fine-grained meaning of congestion. So, in a generalized way, a single ECN mark demonstrates discrete congestion, and the more ECN marks the network generates, the heavier congestion it is experiencing. This means that the marks in L4S can be more frequent and also that the congestion control needs to treat ECN marks differently, without a drastic decrease, as defined in RFC 9331 [Schepper and Briscoe 2023].

This frequent marking behavior is possible through the Dual Queue Coupled AQM (DualQ). It is a generic specification defined in RFC 9332 [Schepper et al. 2023] which specifies that to enable L4S deployment on the Internet, the network path must have:

- Two queues, one for low latency, scalable congestion control traffic, called the L4S queue, and another queue for standard queue-building flows, called the classic queue.
- A coupled marking/dropping probability that ensures that classic and L4S flows receive a fairness-promoting congestion signaling, to enable that no CCA type starve another.
- A scheduler to prioritize the L4S packets dequeuing in favor of classic packets, without starving the classic ones.

The dual queue requirement is mandatory to provide latency isolation. Since classic flows require queue buildup to allow high link utilization, and L4S flows require low queue occupation to enable low queue delay, it is necessary to have separate queues. Having one queue could not satisfy this requirement, since the L4S flows would have to sit in a standing queue built by the classic packets.

The *coupled* part of the DualQ regards the coupled mark/drop probability across the two queues. Since the main goal of the DualQ is to allow safe and fair coexistence between classic and L4S flows, the marking rate for each queue must be compatible with the congestion experienced by the DualQ. Most importantly, it must also correspond to how each congestion control reacts to congestion signals. This correspondence is key because if any CCA receives more or less congestion signals in an uncoupled way, it will be able to scale better or worse its rate, creating an imbalance between the flows.

The final step of the DualQ is the conditional priority scheduler. It serves the purpose of indeed prioritizing dequeuing L4S packets in favor of classic ones, since they have higher priority and must not stay in the queue for too long. The proposed AQM that complies with these standards is the DualPI2, developed by the L4S official group and available as a Linux AQM qdisc.

In addition, for a L4S flow to receive frequent marks from the network, it must

react appropriately. L4S addresses this by defining the *scalable congestion control* for L4S flows. Such CCAs do not need to build a queue on the AQM, relying on the fact that the AQM will be constantly marking packets to avoid the high sawtooth of bandwidth probing. Additionally, the congestion window of scalable CCAs will provide a "small" sawtooth that remains on near-full link utilization but never overshoots. And since it does not induce queue building, it will not induce queue delay.

To enable any congestion control to become scalable and compatible with L4S, L4S defined the *Prague Requirements*, a set of directives that any CCA must comply with to be considered an L4S capable flow, that would take advantage of the low latency network while maintaining coexistence between any other flow type. Based on these requirements, *TCP Prague* is the main scalable CCA compatible with L4S, provided by the official L4S group and also available as a Linux CCA option.

In essence, L4S it is a type of "agreement" between the senders and the network: the senders guarantee they will not fill the network queues and will not overshoot with bandwidth probing. And the network guarantees that it will prioritize those flows and constantly mark these packets, delivering them with low queue delay.

The works in [De Schepper et al. 2025], [Shirmarz et al. 2025], [Fejes et al. 2020], [Schepper et al. 2022] and [Graff et al. 2024] present extensive evaluation on L4S flows considering different scenarios such as network access, congestion controls, and AQMs. [De Schepper et al. 2025] focus on military tactical defense applications on a 5G network. [Shirmarz et al. 2025] focuses on cloud gaming traffic, with approaches for automatic classification of L4S flows. [Fejes et al. 2020] presents comparison of different scalable congestion control flavors, considering classic and L4S AQMs. And [Graff et al. 2024] presents experimentation on how background classic traffic can interfere with cloud gaming L4S-enabled traffic. [Sarpkaya et al. 2025] focuses specifically on L4S flows competing with classic ones under classic AQMs. Lastly, [Schepper et al. 2022] presents the in-depth L4S evaluation from the official research group, assessing coexistence and overall L4S performance.

What all these works have in common is that they perform evaluation considering different scalable congestion controls under L4S and non-L4S queues. The expected results, as also reported in the L4S RFC draft [White 2025], is that a L4S flow will starve any classic flows under a single-queue classic AQM such as PIE, RED or CoDel. In addition, using the state-of-the-art FQ-CoDel AQM will provide a better coexistence but not as low queue delay as DualPI2 provides. Furthermore, scalable congestion controls such as BBR or DCTCP are also not expected to properly work under DualPI2, because of known under-utilization issues [Team 2025].

Because of this, further extending experiments on the same coexistence issue cannot yield different results, since the performances are expected and known. However, assessing round-trip time importance under an L4S network for the classic flows is still significant. After all, assessing adoption consequences for servers is also important, given that they are the main purpose of such architectures.

## 3. Experimental setup

We create a basic dumbbell topology with a server hypervisor connected to a router that will deploy the DualPI2 AQM. Then, the AQM router connects to another router that

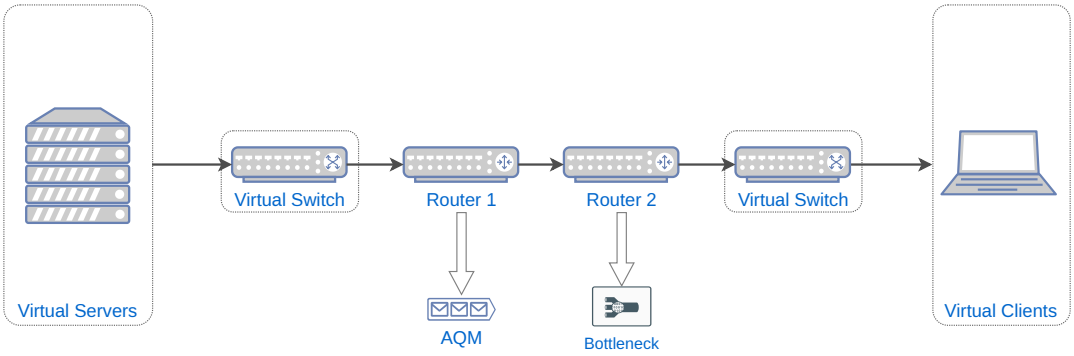


**Figure 1. Generic logical topology for all of our testbed experiments**

emulates the bottleneck condition using netem and tc applications. This router is connected to the client hypervisor. We use hypervisors to enable multiple client and server deployments without the complexity of having multiple hardware, and we have a separate network for experiment controlling. This scenario emulates a common home telecom scenario, where the Internet connection usually with high capacity connects to a residential bottleneck network such as Wi-Fi.

All network cards are gigabit ethernet, which gives enough room for bottleneck experimenting. Figure 1 illustrates the generic topology used for our experiments. We say generic because we will deploy different numbers of clients along the experiments, as explained during the discussion.

In addition, one of the main functions of queue management, and, consequently, L4S, is managing network resources for a network bottleneck. Because of this, we emulate a bottleneck in our scenario to further assess that queuing would be necessary and also that the evaluation would be appropriate with the literature. For our results, we use a bandwidth of 10 Mbps with a base RTT of 30 ms, aligning with common network conditions on home Wi-Fi connections or rural mobile access networks, for example [Schepper et al. 2022].

**Table 1. Hardware and software testbed specification**

| | **Routers** | **Servers** |
|---|---|---|
| **Processor** | Intel(R) Core(TM) i5 3.33GHz 4 Cores | 1 vCPU of Intel(R) Xeon(R) E-2434 |
| **RAM** | 8GiB | 4GiB |
| **NIC** | 8x Intel Corporation 82574L Gigabit | 2x VirtIO gigabit network card |
| **OS** | Debian GNU/Linux 13 | Ubuntu 22.04.5 LTS |
| **Kernel** | 6.12.54-0b264c55b-l4steam-117 | 6.6.114-a76b708b2-l4steam-116 |
| **iproute2** | 5.15, libbpf 0.3.0 | 5.15, libbpf 0.5.0 |

Our testbed's hardware and software specification is defined in Table 1. The operating systems are different simply due to hardware compatibility on the routers, which

were more stable and simpler to configure with Debian GNU/Linux. Our server Virtual Machines (VMs) were already used in other experiments, and to avoid reimplementation and reconfiguration to match the operating system on the routers, we chose to keep the distinct systems. This explains the different kernel versions as well. However, we must indicate that between versions 6.6 and 6.12 of L4S's kernel codes for Prague or DualPI2, there are no differences in their inner functionalities. We use DualPI2 and TCP Prague as our L4S software, both from the official L4S repository [Team 2025]. The configuration of the server in Table 1 regards the VM on the server, so every VM on the server will have this configuration. We run each experiment for 30 seconds using iperf as a flow sender, and we repeat every experiment 30 times for increased statistical validity. We activate ECN marks for the TCP Cubic flows, whenever an experiment contains such protocol.

## 4. Results and discussion

We start the results with a baseline discussion. For this experiment, we run only two flows, one TCP Prague and one Cubic for 30 s on a bottleneck of 10 Mbps bandwidth and 30 ms base RTT. This baseline serves to address the correct functionality of our testbed and also to assess the basic metrics. The results are in Figure 2.

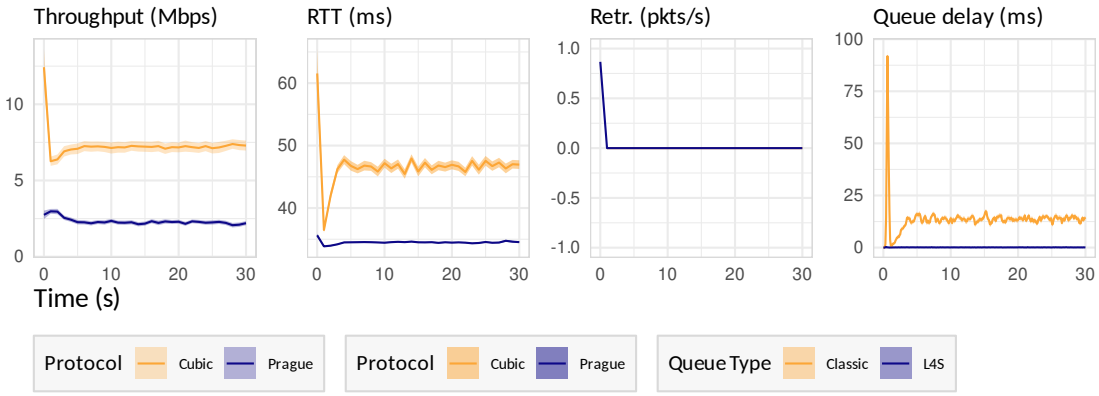


**Figure 2. Line plot for the mean with a region of confidence interval of 95%. Server and AQM results for baseline scenario with bottleneck of 10 Mbps bandwidth and 30 ms RTT.**

Given the low bandwidth of this scenario, it is common for an initial overshoot on throughput, given that the initial slow-start tends to increase the congestion window exponentially. This happens independently of the CCA or AQM, being a common behavior for TCP in low bandwidth scenarios [Partridge et al. 2002].

DualPI2 provides fairness to an extent for Prague flows; it does not starve any TCP but ends up giving more bandwidth utilization to Cubic. This behavior is expected since DualPI2's default marking threshold is 1 ms. Considering that the serialization time for a single packet on a 10 Mbps link is 1.2 ms, it is expected that the AQM will mark more packets because it will more likely exceed this threshold. This will induce TCP Prague to further reduce its congestion window. We could increase the threshold to accommodate this time, but we chose to keep the default parameters to better emulate a realistic scenario where a variable bandwidth would prevent predicting the available bandwidth and parameter tuning.

Nonetheless, it is important to notice that such difference is not massive, not characterizing Prague starvation. Prague receives a stable ≈2.5 Mbps throughput and Cubic ≈7.5 Mbps. Ultimately, retransmission – caused mainly by packet loss – is close to zero, since Cubic uses ECN as well.

The L4S mean queue delay is around 100 $\mu$s (0.1 ms), indeed the close to zero queue delay as expected. Because of this, the line in the graph appears floored at zero. The classic queue delay on DualPI2 has a good result as well, considering a classic queue, with around 15 ms – the threshold for classic queues – throughout the transmission with low variability.

### 4.1. Classic competition

We continue our experiments by reverting a bit to a classic scenario with competition. In our baseline, we have established that Prague and Cubic can coexist in a steady-state transmission. To further assess coexistence, we want to understand how a classic scenario behaves with multiple flows competing for the same bandwidth. To achieve this, we first change the AQM to FQ-CoDel, since it is the current state-of-the-art AQM [Albisser et al. 2019] and the default in Linux kernel and widely deployed over current network equipment. Furthermore, we use TCP Cubic as the transmission protocol for the same reasons.

We implement four servers and four clients, and each of them initiates a flow at a different time, 10 seconds after the other. So, the first server initiates its transmission, and 10 seconds later the second server initiates its transmission, and so on. Our fourth server initiates its transmission at 30 s, and it lasts for 5 seconds so we can assess if it is possible to converge rate at this given time.

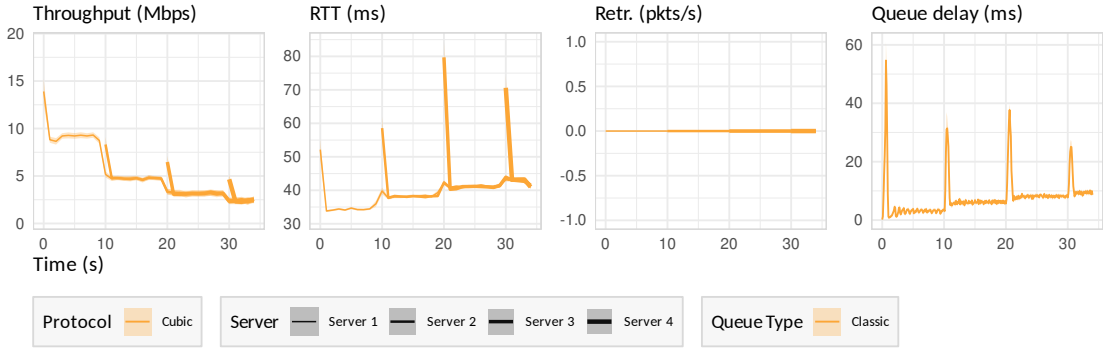


**Figure 3. Line plot for the mean with a region of confidence interval of 95%. Server and AQM results for classic competition scenario with bottleneck of 10 Mbps bandwidth and 30 ms RTT.**

The results for these experiments are available in Figure 3. Throughput performance is satisfactory. The first server achieves near-full link utilization of ≈9.8 Mbps with low variability. When the second server starts, both flows converge at around 4.9 Mbps. This pattern continues as long as the other flows start, with all flows having its fair share of bandwidth utilization. This is what the classic FQ-CoDel provides for its flows.

However, a problem is visible when looking at RTT performance. Remember that the base link RTT is 30 ms. The first server, when running alone achieves considerable

low RTT of ≈34 ms, only 4 ms above link RTT. However, the simple addition of another flow at 20 s increases the RTT for both servers to around 38 ms, another 4 ms addition to the base RTT. This pattern continues until all four flows are running with 43 ms RTT each. We conclude that the constant addition of more flows can result in the constant addition of more RTT to all flows present in the network path, which indeed is not considered scalable and is unable to achieve the low latency connection status.

Furthermore, queue latency is another symptom of such behavior. Queue delay has high oscillation, with values ranging from 4 to 2 ms throughout the transmission of the first server. As long as other flows keep initiating on the network, the behavior observed in RTT remains, with an increasing step behavior on queue delay that gradually increases until ≈10 ms with four flows.

Retransmission has a good performance, since we use ECN on all Cubic flows. This ensures that the queue marks packets before dropping them, preventing further needs for retransmissions.

From this classic experiment, we can assess that in a classic bottleneck, is expected that classic TCPs such as Cubic performs with a fair throughput and low loss, but high RTT that increases proportionally with the amount of flows present in the network path.

### 4.2. L4S competition

After understanding how classic TCPs performs in classic AQMs, we can now compare it with the L4S flows. To achieve this, we repeat the same approach of the experiments in the last section but using the L4S components. We maintain four servers that initiate their flows sequentially 10 seconds after one another, but we add support for TCP Prague for each server. In addition, we change the AQM on the bottleneck for the DualPI2. To further assess competition in a L4S scenario, we gradually swap CCA for each server.

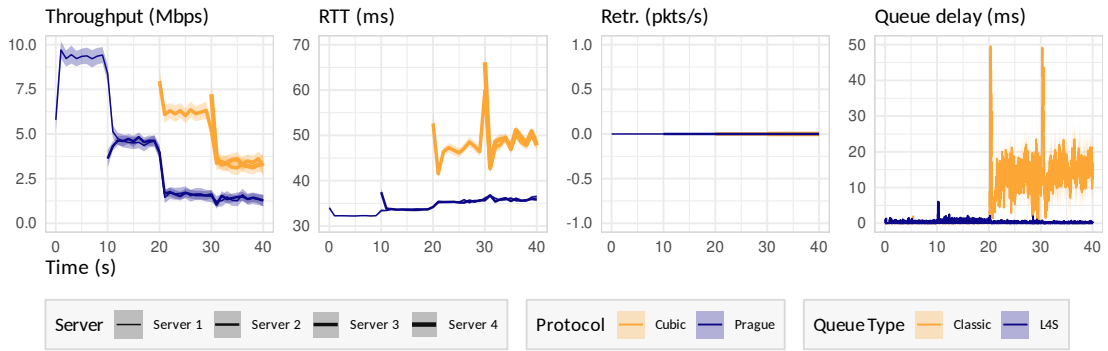


**Figure 4. Line plot for the mean with a region of confidence interval of 95%. Serverand AQM results for L4S competition scenario with 2 Prague servers and 2 cubic servers, with bottleneck of 10 Mbps bandwidth and 30 ms RTT.**

In the first scenario in these experiments, we deploy the first two servers with TCP Prague and the last two with Cubic. Figure 4 shows the results. Server 1, that runs TCP Prague, initiates alone for 10 s, where it achieves a near-full bandwidth utilization of around 9.5 Mbps. After the second server initiates, both flows smoothly converge at around 4.7 Mbps, with both performing at their fair share of throughput. After the third

server initiates its Cubic flow, the same behavior from our baseline happens. Cubic takes ≈6 Mbps and the other two Prague flows take ≈2 Mbps each. When the last Cubic flow starts, the first Cubic flow ends up yielding more bandwidth, since it is now sharing the DualPI2's classic queue with another flow, and both take around 3.2 Mbps. The other two Prague flows have a slight decrease and remain in 1.5 Mbps. These results show that under L4S, Cubic has no coexistence issues with other Prague flows, even being able to achieve higher bandwidth than it would on a classic AQM. It is important to mention again that this behavior is due to the high serialization time in our bottleneck scenario. Other bottleneck configurations could result in more equal throughput performance, but L4S ensures fairness is maintained [Schepper et al. 2022, Albisser et al. 2019].

The RTT behavior shows an interesting path. Both Prague flows maintain low queue delay throughout the entire transmission. The first flow, when running alone, stays around 32 ms, which is only 2 ms above the link latency. With the addition of another Prague flow, both RTT stabilizes at ≈33 ms, only adding ≈1 ms. On the first milliseconds of the transmission, the starting flow has a rapid spike of around 2 ms above its stable converged RTT, mainly due to the initial burst in the slow-start window mentioned in the baseline section.

When the Cubic flows initiates, the impact on Prague's RTT is small when compared to the impact on Cubic flows seen on the previous experiments. The higher RTT experienced in the Prague flows is 35.5 ms, which is when the second Cubic flow starts. The worst Prague RTT is roughly the same as the best Cubic RTT from the last experiments. The Cubic RTT results, following the same trend from the last experiments, are high and susceptible to addition of other Cubic flows. They show a higher variation, oscillating from 46 to 50 ms, with high spikes of around 60 ms when new Cubic flows initiates.

Queue delay has a similar result shown in baseline. The L4S queue delay remains below 1 ms with some spikes above this value. On the other hand, the classic queue remains roughly around 15 ms with high oscillation between 10 and 20 ms.

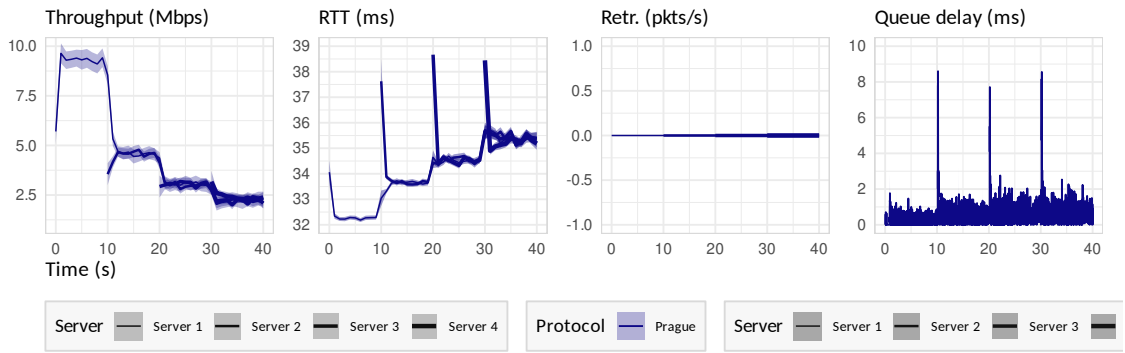


**Figure 5. Line plot for the mean with a region of confidence interval of 95%. Server and AQM results for L4S competition scenario with only Prague servers and no Cubic servers, with bottleneck of 10 Mbps bandwidth and 30 ms RTT.**

Moving forward on our experiments, we remove all Cubic flows from the scenario, and use only TCP Prague on all servers. Figure 5 demonstrates these results. Regarding

throughput, all flows share the bandwidth with fairness and smooth convergence. During the entire transmission, flows can achieve a near-full link utilization, starting with the first flow running alone at 9.5 Mbps until the end, where all four flows have ≈2.3 Mbps each.

RTT performance follows the same trend as in Figure 4. The first flow starts at around 32.5 ms and steps up roughly 1 ms as new flows initiates, rounding at 35.5 ms when all four flows are running, which is the same RTT for the Prague flows when we had 2 Prague and 2 Cubic flows.

This demonstrates that the addition of more flows to a L4S connection will only impact the Prague flows regarding the bare minimum physical propagation delay of splitting the link capacity between the flows. And this impact for a given TCP Prague is indifferent if it is classic or L4S flow. We can further conclude this when looking at the queue delay, where we can see that the 1 ms RTT does not originate there.

The queue delay at the beginning of the experiment, when the first flow is running alone, has a mean of around 0.8 ms. After the second flow starts, the queue delay increases to around 1.3 ms; however, the greater concentration is still below 1 ms. Furthermore, after the third flow starts, the queue delay increases to 1.5 ms while also keeping the greater concentration of results around 1.2 ms. After the fourth flow starts, the queue delay remains the same, showing a possible stability where more flows would not necessarily yield increased queue delay. To improve analysis of queue delay, we zoomed the graph in Figure 6. It is possible to see in the figure how the concentration of value remains roughly below 1.2 ms, with constant spikes above that. We highlight that every 10 seconds, the queue delay has a rapid spike of 8 ms, which is a direct result of the slow-start from the newly initiated Prague flows. The queue rapidly manages this spike with increased marking, which makes the server properly react to it by reducing the window.

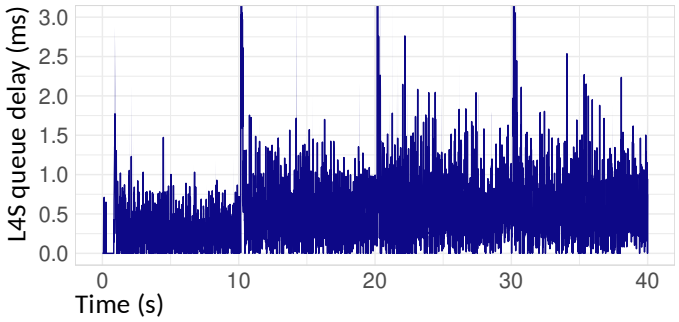


**Figure 6. Zoomed line plot for the mean with a region of confidence interval of 95%. AQM queue delay results for L4S queue for the scenario with only Prague servers and no Cubic servers, with bottleneck of 10 Mbps bandwidth and 30 ms RTT.**

To further assess L4S's performance, we repeat the same experiments using a different bottleneck configuration with higher capacity to emulate scenarios with better configurations. For this scenario, we use a bandwidth of 100 Mbps and a base RTT of 5 ms. Figure 7 shows the results for this scenario.

The same pattern is observed in this higher-capacity scenario. TCP Prague is able to converge throughput with fairness across any number of flows, with a final throughput share of 25 Mbps for all four flows. In addition, since the bandwidth is higher, the ad-

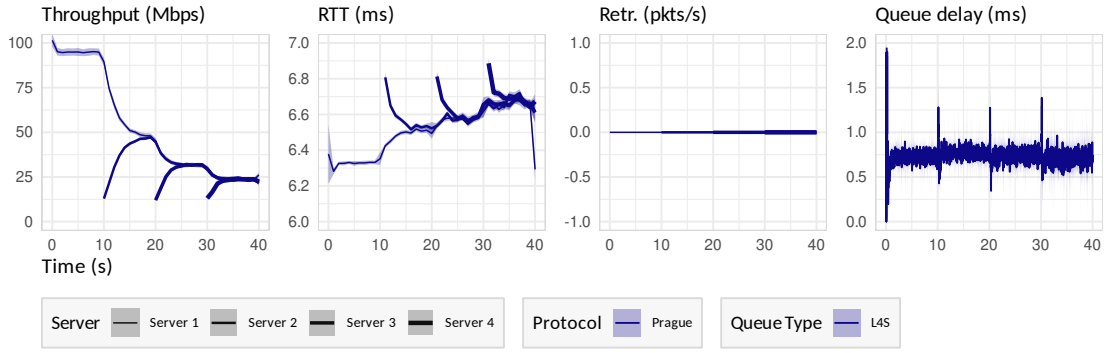


**Figure 7. Line plot for the mean with a region of confidence interval of 95%. Server and AQM results for L4S competition scenario with only Prague servers and no Cubic servers, with bottleneck of 100 Mbps bandwidth and 5 ms RTT.**

ditional delay on RTT is much lower, since the queue delay can remain drastically low with the higher bandwidth capacity. The queue delay in this scenario remains unchanged by the presence of more Prague flows, where the addition of more flows does not induce higher queue delay or higher variation. The mean remains constantly below 1 ms, with small variation.

This reflects on the RTT, which increases only from 6.3 ms for one flow to 6.7 ms for all four flows. We understand that the increase is again the bare minimum from propagation/processing delay, since queue delay remains unchanged and the amount of mean RTT increase is close to the serialization time of 0.12 ms.

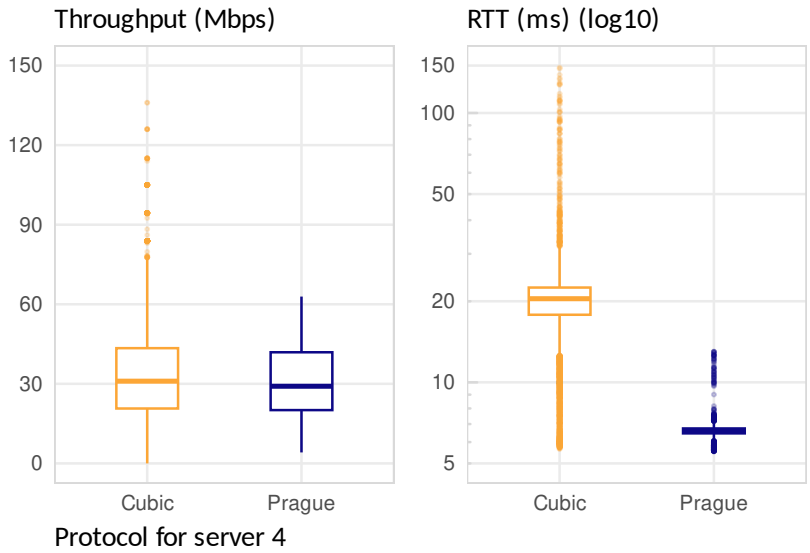


**Figure 8. Box plot for all samples of all executions for the scenario with bottleneck of 100 Mbps bandwidth and 5 ms RTT. The samples are from server 4, from the scenario with 2 Prague and 2 Cubic flows and with 4 Prague and no Cubic flows, so server 4 uses either Cubic of Prague depending on the scenario.**

Figure 8 shows a box plot for all samples of all experiment executions of this same scenario. We plotted the throughput and RTT of server 4, which is executed with both TCP Prague and Cubic, and it enables us to directly assess data distribution when a certain server uses each CCA under the same network scenario. The plots show that there is no statistical difference for the throughput performance, which presents similar medians and quartiles. Even though Cubic presents outliers for higher throughput, it does not necessarily qualify as higher performance since it is inconsistent for the entire transmission.

On the other hand, when analyzing RTT, the difference in performance is visible. The median for TCP Cubic stays around 20 ms, while for Prague, the median and quartiles are close enough to make it difficult to see the distinction on the graph, staying around 6.5 ms. The outliers in TCP Prague are no higher than 15 ms, while in Cubic it is as high as 150 ms, which is 10x greater.

From these results, we understand that L4S can coexist with classic flows, where the adoption of an L4S AQM on a network bottleneck would not deteriorate existing classic flows. However, the option to remain using a classic transport may be unjustified, since the results show that classic flows would only harm themselves with the queue buildup behavior. Switching to an L4S flow would guarantee the same bandwidth utilization with the benefit of low latency.

Furthermore, results show that adding more Prague flows does not translate to worse performance for the Prague flows themselves. The throughput is shared evenly for all flows, and the RTT only increases roughly 1 ms for each new flow, which we understand is the minimum of the link propagation. In addition, queue delay remains low and stable, under 1 ms. This contrasts with Cubic, however, which suffers higher RTT degradation from each newly initiated Cubic flow, which may lead to higher latency considering a more "polluted" scenario.

The results indicate that L4S does not deal with "scarce" resources, where more flows using it would worsen performance. In fact, more services adopting it would only benefit from it and provide better services for their users.

## 5. Conclusion

This work empirically addressed the coexistence issue of L4S flows under an L4S-capable bottleneck. Our findings lead to three conclusions. First, DualPI2 successfully enforces latency isolation. Even under saturation by multiple Cubic flows inducing queue delays exceeding 15 ms, L4S flows consistently maintained negligible queuing delay ($\approx$ 100 $\mu$s), proving that low-latency guarantees can be sustained in "hostile" shared bottlenecks. Second, coexistence does not come at the cost of legacy starvation. Classic flows retained their fair share of bandwidth, demonstrating that the coupled probability mechanism effectively balances throughput across disparate congestion control logic.

Most importantly, we evidenced that, given a network with a proper L4S AQM implementation, L4S flows pose no danger to classic flows. Classic transports can only benefit from switching to L4S-compatible congestion controls, with no degradation of bandwidth utilization or fairness and still achieving low latency. In addition, this switch would not result in performance degradation for any other L4S flows sharing the same network.

Therefore, we argue that network operators can deploy L4S AQMs immediately to support emerging real-time applications, and also service providers can activate L4S-compatible CCAs to take advantage of such networks.

Future work will expand this analysis to variable capacity wireless links (e.g., Wi-Fi 6/7) to assess how DualPI2 adapts to rapid physical layer rate fluctuations and also how well TCP Prague can adapt to such fluctuations.

## Acknowledgment

This research was supported by *Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP)*, *Fundação de Amparo a Ciência e Tecnologia de Pernambuco (FACEPE)*, and *Ericsson Telecomunicações S.A. – Brasil*.

## References

3GPP (2025). *Typical traffic characteristics of media services on 3GPP networks (Release 19) – TR 26.925*. Technical Specification Group Services and System Aspects.

Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and Steen, H. (2019). Dualpi2-low latency, low loss and scalable throughput (l4s) aqm. *Proc. of Netdev*.

Briscoe, B., Schepper, K. D., Bagnulo, M., and White, G. (2023). Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture. RFC 9330.

Cheshire, S. (2025). Ten things you need to know about l4s.

De Schepper, K., Chang, C.-Y., Hamilton, A., and Spruyt, K. (2025). Low Latency Low Loss Scalable Throughput (L4S) for Time-Critical Defense Applications. In *2025 International Conference on Military Communication and Information Systems (ICMCIS)*, pages 1–6.

Fejes, F., Gombos, G., Laki, S., and Nádasy, S. (2020). On the incompatibility of scalable congestion controls over the internet. In *2020 IFIP Networking Conference (Networking)*, pages 749–754.

Gettys, J. and Nichols, K. (2012). Bufferbloat: dark buffers in the internet. *Commun. ACM*, 55(1):57–65.

Graff, P., Marchal, X., Cholez, T., Mathieu, B., Tuffin, S., and Festor, O. (2024). Improving cloud gaming traffic qos: A comparison between class-based queuing policy and l4s. In *2024 8th Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–10.

Partridge, D. C., Allman, M., and Floyd, S. (2002). Increasing TCP's Initial Window. RFC 3390.

Sarpkaya, F. B., Fund, F., and Panwar, S. (2025). To adopt or not to adopt l4s-compatible congestion control? understanding performance in a partial l4s deployment. In Testart, C., van Rijswijk-Deij, R., and Stiller, B., editors, *Passive and Active Measurement*, pages 217–246, Cham. Springer Nature Switzerland.

Schepper, K. D., Albisser, O., Tilmans, O., and Briscoe, B. (2022). Dual queue coupled aqm: Deployable very low queuing delay for all. Accessed in december 2025.

Schepper, K. D. and Briscoe, B. (2023). The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S). RFC 9331.

Schepper, K. D., Briscoe, B., and White, G. (2023). Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S). RFC 9332.

Shirmarz, A., Bragatto, M. N., Verdi, F. L., Singh, S. K., Rothenberg, C. E., Patra, G., and Pongrácz, G. (2025). In-network ar/cg traffic classification entirely deployed in the programmable data plane: Unlocking rtp features and l4s integration. In *2025 IEEE 11th International Conference on Network Softwarization (NetSoft)*, pages 477–485.

SpeedTest (2025). Speedtest global index. Internet. Accessed: November 2025.

Team, L. (2025). Kernel tree containing patches for tcp prague and the dualpi2 qdisc. GitHub Repository.

White, G. (2025). Operational Guidance on Coexistence with Classic ECN during L4S Deployment. Internet-Draft draft-ietf-tsvwg-l4sops-08, Internet Engineering Task Force. Work in Progress.