Simulating and Analyzing the Performance of TCP Under Extreme Conditions

Evaluating the Impact of L4S on TCP Performance

Alexandru-Ioan Tăbăcaru

Supervisor(s): Fernando Kuipers, Adrian Zapletal

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 13, 2025

Name of the student: Alexandru-Ioan Tabacaru
Final project course: CSE3000 Research Project
Thesis committee: Fernando Kuipers, Adrian Zapletal, Asterios Katsifodimos

Abstract

The Low-Latency, Low-Loss, Scalable-Throughput (L4S) service aims to support real-time applications by enabling high throughput with sub-millisecond queueing delay. It combines scalable ECN-based congestion control (e.g., TCP Prague) with a Dual-Queue AQM such as DualPI2 to separate low-latency and classic traffic.

This paper evaluates how L4S behaves under stressful network conditions using an extended `ns-3` testbed with DualPI2, TCP Prague, and ECN-enabled BBRv3. We test five scenarios: RTT jitter, bandwidth shifts, mixed traffic, wireless loss, and scalable-to-scalable coexistence.

Our results show that TCP Prague consistently delivers low delay and stable throughput, whereas legacy TCP Cubic shows elevated and more variable delay—especially under jitter and in shared queues. ECN-BBRv3 coexists cleanly with Prague, but when Cubic and Prague share a queue, Prague dominates bandwidth. L4S thus meets its latency goals, but fairness with classic TCP remains an open issue.

1 Introduction

The Transmission Control Protocol (TCP) is the primary transport protocol for reliable end-to-end data delivery on the Internet, adapting its congestion window to observed network conditions to balance throughput and delay [2]. In 2025, emerging applications such as cloud-based extended reality (XR), real-time gaming, and interactive remote collaboration demand sub-millisecond latency and proactive loss avoidance to support real-time interactivity.

Traditional loss-based congestion control algorithms such as Cubic and model-based algorithms like BBR can deliver high throughput under steady conditions [8; 10; 5; 17]. However, Cubic tends to induce persistent queueing delays due to its aggressive probing and buffer-filling behavior, leading to the well-known bufferbloat problem [16; 7]. BBR generally avoids large queues when operating in isolation, but may still cause increased queueing delay in the presence of loss-based cross-traffic or under rapidly changing network conditions [16].

The Low-Latency, Low-Loss, Scalable-Throughput (L4S) architecture promises to address these challenges by combining dual-queue Active Queue Management (AQM) with Explicit Congestion Notification (ECN)-driven congestion controllers to maintain high throughput while bounding queueing delays below one millisecond [4]. A key design feature of L4S is the separation of classic (loss-based) and L4S (ECN-based) traffic into distinct AQM queues, minimizing interference between legacy and scalable flows. Early evaluations demonstrate significant latency reductions and throughput improvements under moderate conditions [12], yet L4S's behavior under extreme RTT variance, sudden bandwidth shifts, mixed legacy traffic, and competition between multiple scalable controllers remains unexplored. To address this, we develop a simulation-based framework in `ns-3` to analyze the performance of L4S under challenging conditions.

Research Questions. We address:

1. Under realistic **RTT variance**, how does L4S compare to legacy TCP (Cubic, BBR) in throughput and queueing delay?
2. What is the effect of **rapid bandwidth fluctuations** on throughput stability and queueing delay of L4S versus legacy TCP?
3. How does L4S handle **coexistence fairness** when TCP Prague shares a DualPI2 queue with Cubic on a wired path?
4. How do **competing scalable controllers** (TCP Prague and ECN-enabled BBRv3) interact within the L4S framework?
5. How robust is Prague to **random Wi-Fi losses** compared with Cubic?

Key Findings. Our results show that TCP Prague maintains sub-millisecond queueing delay across all tested scenarios and generally sustains near link-rate throughput. Compared to Cubic, it offers better delay control and greater resilience to jitter and capacity changes, though it dominates in shared queues due to DualPI2's asymmetric marking thresholds. ECN-enabled BBRv3 coexists fairly with Prague in scalable-only settings, achieving balanced throughput and low delay. Finally, Prague remains robust under loss, staying within 5–8% of Cubic's throughput.

Contributions. We **implemented and integrated** missing components in `ns-3`—including a dual-queue AQM, **TCP Prague**, and **ECN-enabled BBRv3**—to support full-path L4S evaluation. We **designed stress-test scenarios** involving delay variability, bandwidth collapse, traffic competition, and wireless loss, and used these to **benchmark L4S performance** against classic TCP and modern scalable controllers.

Paper Structure. The remainder of this paper is organized as follows. Section 2 covers background. Section 3 reviews related work. Section 4 details our simulation methodology (Simulation Setup and Implementation Details). Section 5 presents evaluation scenarios, results, and analysis. Section 6 discusses responsible research considerations. Section 7 offers future work and concludes the paper.

2 Background

This section introduces the building blocks needed for the methodology and results. We first discuss the evolution of TCP congestion control, then explain how modern queue-management and signalling mechanisms enable very low latency, and conclude with the scalable controllers evaluated in this study.

2.1 Congestion-Control Evolution

TCP Reno introduced the now-standard congestion control principles of *additive-increase and multiplicative-decrease (AIMD)*, along with mechanisms like *slow start*, *fast retransmit*, and *fast recovery*. These features prevent congestion collapse by reducing the sending rate upon detecting packet loss, which is interpreted as a congestion signal [2].

TCP Cubic builds upon Reno's approach but replaces its linear window growth with a *cubic function* to better utilize high-bandwidth and long-delay paths [8]. The cubic function enables faster probing of available capacity after a congestion event, making Cubic more aggressive than Reno in growing its congestion window.

In contrast, **BBR** (Bottleneck Bandwidth and Round-trip time) rethinks congestion control by avoiding both loss and delay as signals. Instead, it models the network path by periodically estimating the *bottleneck bandwidth* and *minimum RTT* to compute a pacing rate [5]. While BBR often achieves low queueing delay and high throughput in stable networks, its model can mis-estimate available capacity or RTT in dynamic environments, leading to unfairness or elevated delay, especially when competing with legacy congestion control algorithms [17].

2.2 Active Queue Management (AQM)

Routers running *Active Queue Management* act before buffers overflow, keeping delay low and providing early congestion signals. For instance, CoDel monitors per-packet queueing delay in the router and drops or marks when delay rises beyond a bound, with no manual tuning required [10]. PIE uses a proportional–integral controller to hold queueing delay near a target via probabilistic drops or marks [11]. FQ-CoDel adds flow-queuing to CoDel, isolating contending flows while maintaining similar delay bounds [9].

2.3 Explicit Congestion Notification (ECN)

Explicit Congestion Notification lets an AQM mark packets, not drop them, when queues begin to build [6]. The receiver echoes the mark, allowing the sender to slow down without loss or retransmission. ECN therefore relies on an AQM; without one, packets never get marked. Vanilla ECN cuts the congestion window by a fixed fraction for any round trip that contains one or more marks, giving only coarse feedback.

2.4 Data Center TCP (DCTCP)

DCTCP improves ECN accuracy by reducing the window in proportion to the fraction of marked packets within a round trip, maintaining sub-millisecond queues while preserving throughput in data-centre fabrics[1].

2.5 L4S Architecture

The *Low-Latency, Low-Loss, Scalable-Throughput (L4S)* service couples two queues under a dual-queue AQM: a scalable queue that marks packets using a DCTCP-style algorithm, and a classic queue for legacy Reno/Cubic traffic [4]. Coupling logic balances the rates of the two queues so scalable flows keep their delay near one millisecond without starving classic flows. **DualPI2** implements this by running two coupled proportional–integral controllers (one per queue) that compute mark/drop probabilities. The scalable queue uses frequent, low-intensity DCTCP-style ECN marks to enforce the low-latency target, while the classic queue operates under a higher delay bound and more conservative marking or dropping behavior [14].

2.6 Scalable Congestion Controllers

TCP Prague generalises DCTCP for wide-area paths, applying proportional ECN back-offs when marks are present and reverting to AIMD on non-ECN paths [15]. **ECN-BBRv3** grafts ECN responsiveness onto BBR's model, improving fairness with other scalable flows while retaining model-based pacing [5; 17].

3 Related Work

Research on the Low-Latency, Low-Loss, Scalable-Throughput (L4S) architecture spans full deployments, incremental roll-outs, fairness concerns, and sensitivity to path dynamics. We summarise the key findings thematically.

Full deployment. De Schepper *et al.* evaluated the DualQ Coupled AQM in a residential–broadband testbed and reported *median* queueing delays of 100-300µs with the 99th percentile below 2 ms while sustaining near line-rate utilisation and zero congestion loss [13]. Oljira *et al.* benchmarked the DualPI2 implementation on a packet-level testbed, validating that L4S delivers sub-millisecond latency isolation for scalable flows and co-exists with classic flows under the DualQ AQM [3].

Partial deployment. Sarpkaya *et al.* modelled incremental roll-out with `ns-3` and hardware switches; when only half of the bottlenecks were L4S-capable, scalable flows still cut their median delay but classic Cubic flows occasionally lost throughput, illustrating non-linear trade-offs between latency and capacity in mixed deployments [12].

Fairness with mixed traffic. P4AIR, proposed by Turkovic and Kuipers, classifies flows by congestion-control behaviour inside a P4 switch and redistributes queueing resources; their experiments show substantial fairness gains compared with FQ-CoDel when Cubic and BBR share a link, though without L4S dual-queue isolation [16].

RTT variability and bandwidth shifts. Zeynali *et al.* investigated BBRv3 across paths whose RTT varied by up to ±20 ms and different buffer sizes; BBRv3 improved fairness relative to earlier BBR versions, but did not consider dual-queue AQMs or more extreme dynamics [17]. No prior work stress-tests L4S itself under rapid RTT swings or abrupt capacity drops.

Gap. Collectively, existing studies validate L4S under ideal or mildly variable conditions, highlight coexistence issues in partial deployments, and explore fairness mechanisms outside the DualQ framework. They leave unanswered how L4S behaves under *extreme* RTT variance, sudden bandwidth collapses, mixed scalable controllers (TCP Prague versus ECN-BBRv3), and Wi-Fi loss, questions addressed by the comprehensive `ns-3` evaluation presented in this paper.

4 Methodology

Our simulation experiments build upon the CableLabs ns-3 repository[1], specifically utilizing the `l4s-wifi` branch. The scenarios `l4s-wired` and `l4s-wifi` serve as the bases for

[1] https://github.com/cablelabs/l4s-wifi-ns3

our experiments, onto which we implemented our specific scenarios.

4.1 Base topologies.

All scenarios use one of two 4-node chains, with S=sender, C=receiver, and $R_{1/2}$ the edge/core routers.

Wired dumbbell: S—R_1—R_2—C. Congestion only on the R_1–R_2 hop; host–router links are set to 2Gbps to ensure the bottleneck is isolated to the router–router hop.

Wi-Fi chain: S—AP—STA. Congestion only on the wireless AP–STA hop; the wired leg (S–AP) is 10Gbps, 10ms one-way delay. The receiver runs directly on the STA.

4.2 Common Configuration

All experiments utilize packet-level simulations in ns-3[2]. Unless stated otherwise, runs last **60 s**, with the first and last 5 s discarded. Two exceptions apply: the bandwidth-step tests in **RQ2** run for 40 s, and the scalable-controller mixes in **RQ4** run for 30 s (discarding only the first 8 s). Flows are long-lived BulkSend→PacketSink connections paced at the sender, using a Maximum Segment Size (MSS) of 1448 bytes. Socket buffers are sized to approximately 1.5–2 times the bandwidth-delay product (BDP) to ensure that flows are not window-limited. To ensure identical randomness across configurations, all runs use the same rngRun=1 seed when launching simulations.

We evaluate four congestion-control and AQM combinations: **TCP Prague with DualPI2** (L4S-compatible), **TCP Cubic with Fq-CoDel** (classic TCP), **TCP Prague and TCP Cubic sharing a DualPI2 queue**, and **TCP Prague and ECN-enabled BBRv3 sharing a DualPI2 queue**.

Prague uses **DualPI2** and Cubic uses **Fq-CoDel** in RQ1 and in the *wired* bandwidth-step runs of RQ2; in every other experiment (Wi-Fi bandwidth-step in RQ2, all mixed-traffic tests in RQ3, and the Wi-Fi loss-sensitivity runs of RQ4) they share the same *DualPI2* queue. Default queue parameters from ns-3 are used (DualPI2: 1000 packets, Fq-CoDel: 10,240 packets).

The performance metrics collected throughout our simulations include throughput, mean and 95th percentile queueing delay (Q_{95}), and fairness quantified by Jain's Fairness Index (JFI).

5 Evaluation

This section evaluates L4S performance across the five research questions. For each RQ, we outline the setup, present results, and summarise key takeaways.

5.1 RQ1 – Sensitivity to Baseline Delay and RTT Jitter

Setup - Wired

To isolate the impact of delay dynamics on a single transport flow we run packet–level simulations in NS-3. Each experiment consists of *one* long-lived TCP connection that traverses a single bottleneck link (100Mbps), fed by a paced sender at host S and drained by a PacketSink at host C.

[2]https://www.nsnam.org/

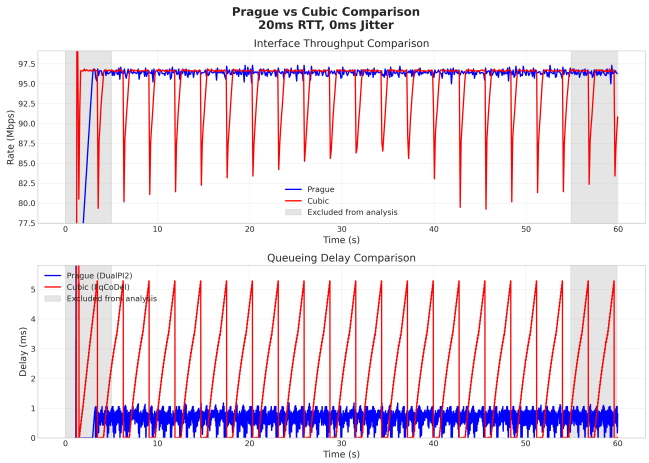

Figure 1: RTT 20 ms, no jitter. Prague keeps the queue below 1 ms; Cubic's saw-tooth climbs to about 5 ms. Both saturate the 100 Mb/s link.

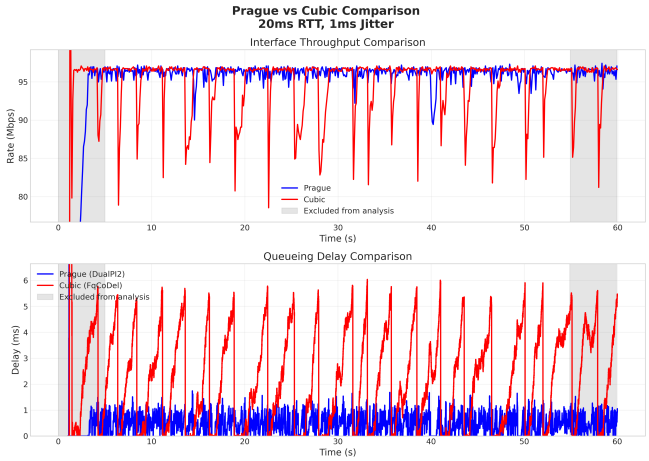

Figure 2: RTT 20 ms with ±1 ms jitter. Cubic shows spikes around 6 ms every few RTTs, whereas Prague's trace remains nearly flat.

Delay model. The one-way propagation delay on the S → R_1 leg is set to $d_0 \in \{10, 20, 40\}$ms (corresponding RTTs 20;40;80ms) and then perturbed by a bounded random walk of amplitude $J \in \{0, 1, 5\}$ms.[3] Only that single link is jittered so that the resulting delay fluctuations fully dominate the end-to-end RTT.

Results

For space reasons we reproduce the 20ms RTT runs only (Figs. 1–3). This is the *most demanding* case for an AQM: the bandwidth–delay product (BDP) is smallest, so any queue growth is immediately visible in the end-to-end delay. The 40 and 80ms traces, qualitatively similar, are archived in Appendix A.

Table 1 condenses the same runs into mean throughput and 95-percentile queueing delay.

[3]Implemented with a helper that reschedules the point–to–point channel delay every 10ms; step size $= 0.1\,J$.

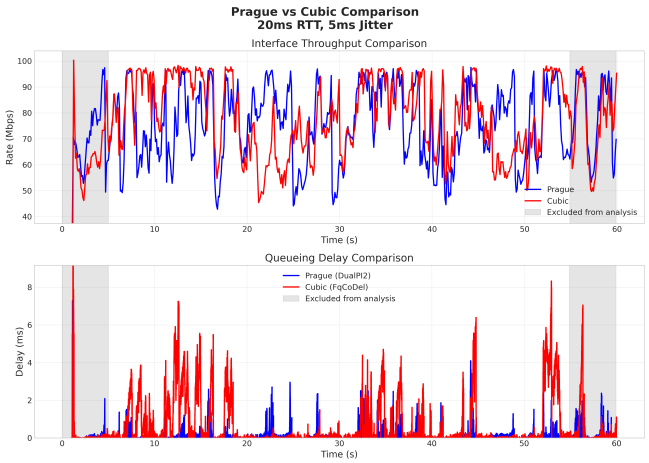

Figure 3: RTT 20 ms with ±5 ms jitter. Cubic peaks at about 8 ms and averages 78.9 Mb/s; Prague stays below 1 ms but falls to 75.7 Mb/s due to more frequent ECN marks.

Table 1: Mean throughput and Q_{95} at 20ms RTT.

	Throughput [Mb/s]		Q_{95} [ms]	
J	Prague	Cubic	Prague	Cubic
0	96.4	95.1	0.93	4.92
±1	96.2	95.0	1.01	5.05
±5	75.7	79.0	0.84	3.90

Throughput response

At 20ms RTT, Cubic reaches 95.1Mb/s with no jitter (Fig. 1), and remains nearly unchanged with ±1ms jitter (95.0Mb/s, Fig. 2). However, under ±5ms jitter (Fig. 3), Cubic's throughput drops to 79.0Mb/s. This reduction occurs as frequent drops shrink the congestion window during jitter spikes.

Prague sustains 96.4Mb/s with no jitter (Fig. 1), 96.2Mb/s with ±1ms (Fig. 2), and falls to 75.7Mb/s under ±5ms (Fig. 3). Although the average remains comparable to Cubic, the throughput trace shows visible oscillations even for Prague at high jitter. These reflect repeated ECN-triggered reductions and recoveries, suggesting that Prague too becomes unstable in terms of rate, however without the high queueing delay.

Queueing delay

In terms of delay, Cubic exhibits significant queueing even without jitter, with a Q_{95} of 4.92ms. With ±1 ms jitter this grows slightly to 5.05ms, and under ±5 ms it drops to 3.90ms (Table 1). These bursts arise as the Fq-CoDel sub-queue quickly fills during jitter-induced variation, leading to packet drops and saw-tooth behavior.

Prague, by contrast, consistently keeps queueing delay under 1.05ms, with $Q_{95} = 0.93ms$ at zero jitter, 1.01ms at ±1 ms, and 0.84ms at ±5 ms (Table 1). DualPI2 marks packets well before large buffers form, allowing Prague to react early and maintain low delay even as jitter increases.

AQM delay bounding. Neither DualPI2 nor Fq-CoDel approach their configured queue capacity in our single-flow tests. Instead, both AQMs bound queueing through delay: Fq-CoDel begins dropping once sojourn time exceeds 5ms, while DualPI2 reacts earlier via ECN marking. These delay-based responses explain the differing queueing behaviors observed.

Higher RTT overview. At higher RTTs (40 and 80ms), we observe the same trend: Prague maintains sub-millisecond queueing and suffers less severe throughput degradation under jitter than Cubic. For full statistics and figures, see Appendix A.

Take-away. Prague+DualPI2 consistently enforces sub-millisecond queuing even under jitter, while sustaining near-link throughput. Cubic + Fq-CoDel behaves acceptably on clean or long-RTT paths, but on short RTTs jitter produces multi-millisecond bursts and the flow loses more throughput as jitter increases.

5.2 RQ2 – Response to Rapid Bandwidth Fluctuations

In §5.1 we kept the link *capacity* constant and varied only the baseline delay and its jitter. Wireless networks, however, often suffer the opposite disturbance: a sudden change of bandwidth caused by link re-provisioning, routing fail-over, or a Wi-Fi rate switch. RQ 2 therefore holds the RTT fixed and introduces a single, instantaneous capacity step at the mid-point of each run. We measure how quickly TCP PRAGUE and TCP CUBIC re-establish the right amount of data in flight, and how much queueing delay they create in the process.

Setup - Wired & Wi-Fi

Wired scenario. A point-to-point bottleneck of fixed RTT 40ms runs at $R_0 \in \{25, 100\}$ Mb/s for the first half of the run and R_1 (the other rate in the set—100 or 25 Mb/s) for the second half. Two step directions are tested for each algorithm:

- **25 → 100 Mb/s** — Prague (P-B1) and Cubic (C-B1);
- **100 → 25 Mb/s** — Prague (P-B2) and Cubic (C-B2).

The rate change is applied to *both* NetDevices so the queue is not flushed.

Wi-Fi scenario. The middle hop is replaced by an 802.11ax AP/STA link (20MHz, 1 spatial stream), connected to the sender via a wired 10ms leg. The rest of the 20ms RTT comes from the AP–STA wireless hop. At $t = 20s$, we call `ChangeMcs()` on both peers to model a rate switch. Six HE-MCS transitions are exercised, each once with Prague and once with Cubic:

- 2 → 7 (P-W1 / C-W1), 7 → 2 (P-W2 / C-W2),
- 4 → 9 (P-W3 / C-W3), 9 → 4 (P-W4 / C-W4),
- 4 → 7 (P-W5 / C-W5), 7 → 4 (P-W6 / C-W6).

What the MCS numbers mean. MCS stands for *Modulation and Coding Scheme*, which defines the modulation format and code rate used by Wi-Fi radios. Higher MCS indices correspond to faster PHY-layer bitrates, assuming sufficient signal quality. In our setup (20 MHz channel, one antenna) the four indices we use map to these approximate net throughputs (after MAC/preamble/ACK overhead):

Table 2: Approximate PHY rates per MCS (20 MHz, 1×1 ax).

MCS 2	MCS 4	MCS 7	MCS 9
19 Mb/s	39 Mb/s	78 Mb/s	103 Mb/s

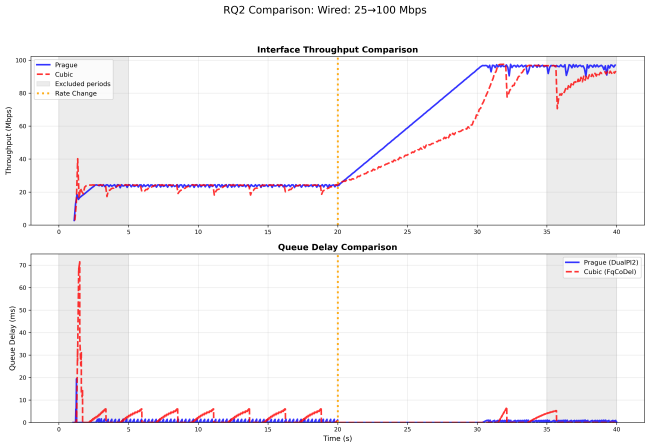

Figure 4: Throughput and queue delay for a wired 25→100 Mbps step. Prague converges faster and avoids queue oscillations; Cubic shows characteristic probe-induced ripples.

Results

Figures 4, 5, 6, and 7 show the main results for the 25→100, 100→25, MCS 2→7 and MCS 7→2 transitions. The remaining rate switches follow the same trend and are reported in Appendix B.

Throughput response

Wired step-up (25→100 Mb/s) — Figure 4: Prague hits full rate in roughly 10 s. Cubic is fully up by 12 s.

Wired step-down (100→25 Mb/s) — Figure 5: When the pipe shrinks, both senders briefly overshoot the new BDP. Prague's backlog drains in 0.3 s. Cubic lands on the right rate almost instantly.

Wi-Fi step-up (MCS 2→7) — Figure 6: Prague settles in 4 s. Cubic needs 6 s. The same "faster fill" pattern repeats for all three upward steps.

Wi-Fi step-down (MCS 7→2) — Figure 7: Prague shows a short transient and reaches the target rate. Cubic also reaches the new rate quickly. Other step-downs (MCS 7→4, 9→4) follow the same behaviour.

Queueing delay

Wired step-up — In Figure 4, Prague keeps $Q_{95} < 2ms$. Cubic shows repeating 5–8 ms saw-teeth during each probe.

Wired step-down — In Figure 5, Prague's queue stays below 1 ms after the rate cut. Cubic brings back the familiar 0–5 ms saw-teeth.

Wi-Fi step-up — In Figure 6, Prague maintains sub-2ms delay. Cubic's queue oscillates up to 40ms.

Wi-Fi step-down — Figure 7 shows Prague with a single spike (<70 ms) right at the switch, then near-zero delay. Cubic continues raising the queue in 10–40 ms cycles for the rest of the run.

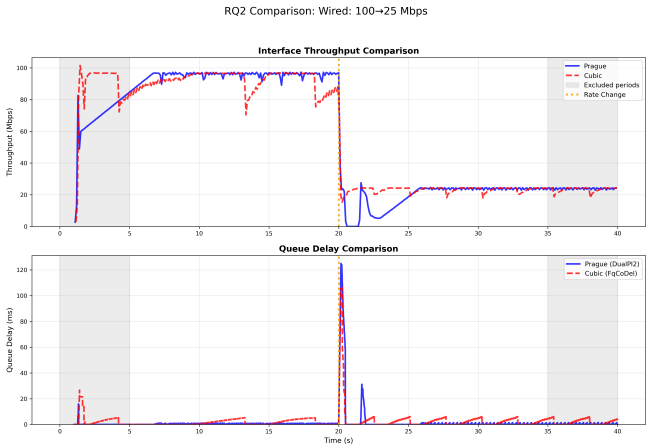

Figure 5: Throughput and queue delay for a wired 100→25 Mbps step. Both senders adjust promptly, but only Prague maintains stable delay.

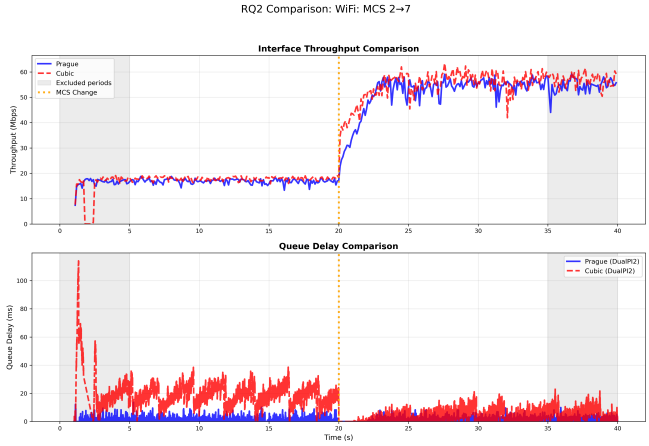

Figure 6: Throughput and queue delay for a Wi-Fi MCS 2→7 upgrade. Prague fills the new capacity faster and keeps delay low; Cubic oscillates with larger queue bursts.

Take-away. Both senders manage to reach the new bandwidth within a few seconds, but they do it very differently. Prague's ECN feedback trims the queue to sub-millisecond levels almost immediately after a rate change regardless of whether the pipe grew or shrank. Cubic achieves comparable throughput, but its loss-driven probes keep the queue swinging between a few and a few-dozen milliseconds (up to 8 ms on the wired link and 40 ms on Wi-Fi) for the rest of the run. For latency-sensitive traffic that margin is the deciding factor.

5.3 RQ3 – Mixed-Traffic Fairness

The last two sections looked at a *single* scalable flow. In real deployments, however, L4S queues will have to coexist with classic TCP. RQ 3 therefore asks: *How do* TCP PRAGUE *and legacy* TCP CUBIC *share capacity when they feed the* same DUALPI2 *queue on a wired path?*

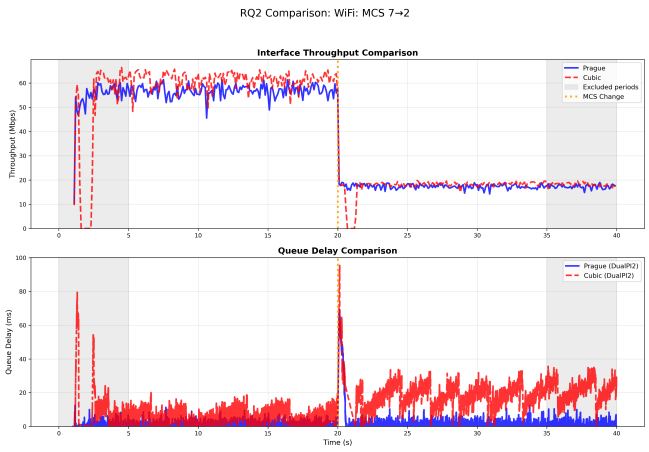

Figure 7: Throughput and queue delay for a Wi-Fi MCS 7→2 downgrade. Prague quickly stabilises; Cubic repeatedly fills the queue.

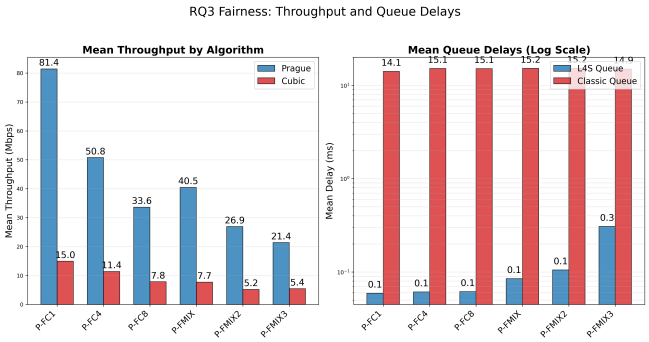

Figure 8: Mean throughput by algorithm (left) and mean queue delay (right) for each traffic mix on the wired link.

Setup - Wired

Traffic mixes. A 100Mb/s point-to-point link with 10ms one-way propagation delay (20ms RTT) carries seven mixes:

- *Coexistence* — 1P + 1/4/8C (P-FC1/4/8);
- *Balanced* — 2P + 2C, 3P + 3C, 4P + 2C (P-FMIX, FMIX2, FMIX3);
- *All-Prague* — 2/4/8 flows (P-FP2/4/8) as baselines.

Why DUALPI2? DualPI2 is the reference AQM for the IETF L4S architecture [14]. It couples two virtual queues that share the same buffer but apply different ECN-marking thresholds: a 15ms target for *classic* traffic (e.g. Cubic) and a far lower, sub-millisecond threshold for the *L4S* branch (Prague). Packets share one queue; only the marking threshold differs.

Results

Figure 8 plots mean per-algorithm throughput (left) and mean queueing delay (right); Table 3 lists the overall Jain's Fairness Index (JFI) and the Prague/Cubic throughput ratio.

Throughput fairness

For the 1P–1C coexistence mix (P-FC1) Prague achieves 81.4Mb/s while Cubic manages 15.0Mb/s (ratio 5.44, JFI 0.678). This indicates that DUALPI2 favours the scalable, ECN-aware branch over the classic, loss-based branch as soon as both share the same queue.

Table 3: Per-experiment fairness: Jain's index (JFI) and Prague/Cubic throughput ratio.

ID	JFI	P/C ratio
P-FC1	0.678	5.44
P-FC4	0.600	4.45
P-FC8	0.635	4.29
P-FMIX	0.684	5.24
P-FMIX2	0.684	5.17
P-FMIX3	0.820	3.94
P-FP2	1.000	—
P-FP4	1.000	—
P-FP8	1.000	—

Adding more Cubic senders does not help the classic side: with four Cubic flows (P-FC4) Prague still earns 4.45 × the aggregate throughput, and with eight Cubic flows (P-FC8) the factor is 4.29. The classic queue's 15ms target remains decisive, holding each Cubic flow to roughly 8Mb/s in the 1P–8C case.

Balanced mixes give overall JFI values between 0.68 and 0.82. Even in the Cubic-favoured 4P–2C configuration (P-FMIX3) Prague moves almost four times as much data (ratio 3.94).

All-Prague baselines (P-FP2/4/8) serve as a control: JFI = 1.00, each scalable sender gets approximately its exact fair share (around 50, 25 or 12.5 Mb/s).

Queueing delay

Across every mix Prague's mean sojourn time stays below 0.3ms, while Cubic consistently queues around the 15ms classic target. Delay isolation therefore holds regardless of traffic proportions.

Take-away. Prague combines sub-millisecond latency with a throughput share roughly at least four times that of Cubic under all mixed scenarios tested. The gap is a direct consequence of DualPI2's generous 15ms classic target. Narrowing the classic queue's 15ms target could likely reduce the imbalance and bring the two algorithms' shares closer, though this would still need to be verified.

5.4 RQ4 – Fairness between Scalable Controllers

With L4S becoming popular, multiple "scalable" TCPs may meet in the same low-latency queue. RQ 4 therefore asks: *When ECN-enabled BBRv3 and TCP Prague share a bottleneck, how fairly do they divide capacity and how much delay do they create?*

Setup - Wired

Topology and queue. A single 100Mb/s, 20ms-RTT link carries all traffic. Both algorithms use the *L4S* (low-latency) leg of one DUALPI2 queue, so every packet is ECN-capable and classed ECT(1); no classic queue is involved.

BBRv3 implementation. The ECN-enabled BBRv3 used here is based on the ns-3 model available at https://github.com/Aruuni/ns3-bbrv3/blob/main/src/internet/

model/tcp-bbr3.cc. We patched that code with the ECN-handling logic from the Linux implementation[4] so that BBRv3 reacts to DualPI2 marks in the same way as the upstream kernel version.

Traffic mixes. Seven 30-second runs cover the following combinations, always starting at $t = 1s$ with paced `BulkSend` flows (`MaxBytes=0`):

- symmetric: 1P–1B, 2P–2B, 4P–4B;
- Prague-scarce: 1P–{2,4}B;
- BBR-scarce: {2,4}P–1B.

Metrics. Per-flow throughput, Jain's Fairness Index (JFI), and sojourn-time statistics are averaged over the steady period $[8, 30)$ s.

Results

Figure 9 shows the per-flow throughput bars annotated with JFI; Fig. 10 plots mean queueing delay.

Throughput fairness

When one Prague and one BBRv3 flow share the link they settle on the same rate ($\approx 48 Mb/s$), yielding a perfect fairness score (JFI = 1.00).

Keeping the mix moderately unbalanced—1P–2B, 1P–4B, 2P–1B or 4P–1B—does not disturb sharing: every configuration up to a 1:4 imbalance maintains an overall JFI of at least 0.91, and throughput among flows of the *same* algorithm remains almost equal (algorithm-specific JFI ≥ 0.97).

With four Prague and four BBRv3 senders, fairness degrades only slightly. BBRv3 captures a marginally larger slice (best BBR flow 22.2Mb/s versus best Prague 20.0Mb/s), pulling the overall JFI down to 0.75, even though each controller is still internally fair (algorithm JFI 0.995).

Queueing delay

Mean sojourn time stays below 2ms in every mix, peaking at 2.01ms for the 4P–4B case (Fig. 10). DualPI2's L4S marking therefore prevents buffer build-up even with the more aggressive probing of BBRv3.

Unlike the Prague–Cubic pairing (RQ 3), two L4S-compatible controllers share a *single* DualPI2 queue without severe bias; only the extreme 4-vs-4 mix dips below a fairness of 0.8. Because both algorithms react to ECN rather than loss, the queue never exceeds a few milliseconds, satisfying the L4S latency objective.

Take-away ECN-enabled BBRv3 can coexist with TCP Prague in the same L4S queue with sub-millisecond delay and overall JFI ≥ 0.9 for up to a 1:4 flow imbalance. Only more heavily over-populated mixes show a noticeable skew, suggesting that mild coupling adjustments may be sufficient to ensure fair scalable-to-scalable coexistence.

5.5 RQ5 – Loss Sensitivity

As a final experiment we take a brief look at how random wireless losses affect throughput. RQ5 asks: *How much throughput degradation do* TCP PRAGUE *and* TCP CUBIC *experience when a single Wi-Fi hop introduces small, uncorrelated packet losses?*

[4]https://github.com/google/bbr/blob/v3/net/ipv4/tcp_bbr.c

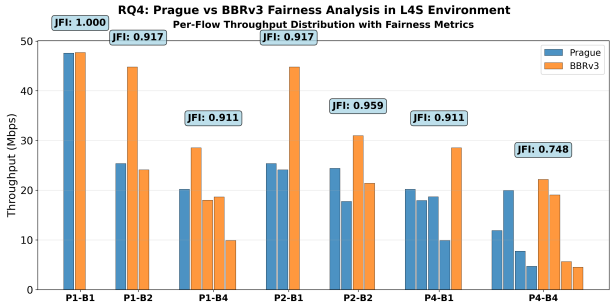

Figure 9: Per-flow throughput and Jain's Fairness Index for the seven Prague/BBRv3 mixes.

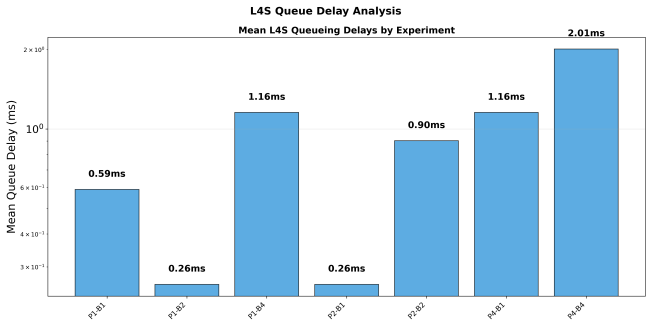

Figure 10: Mean queueing delay per experiment; log-scale y-axis.

Setup - Wi-Fi

The wired bottleneck is replaced by an 802.11ax AP/STA operating in a 20 MHz channel with one spatial stream (HE-MCS 2). A 10ms wired leg in front of the AP keeps the end-to-end RTT at approximately 20ms. Random losses are injected using `RateErrorModel`, applied to the *STA's Wi-Fi PHY layer* via `SetPostReceptionErrorModel`, to simulate uncorrelated per-packet loss at rates of 1, 5 and 10 %. Each loss rate is run once with Prague (P-WLS1–3) and once with Cubic (C-WLS1–3).

Results

Table 4: Mean and 95-th-percentile throughput on Wi-Fi versus random loss.

	Prague [Mb/s]			Cubic [Mb/s]		
Loss	1 %	5 %	10 %	1 %	5 %	10 %
Mean	34.0	32.1	29.8	36.1	34.6	32.4
P95	36.5	34.9	32.8	38.7	37.4	35.7

Table 4 shows that throughput falls roughly linearly with the drop rate for both controllers. Prague trails Cubic by only 5–8 % because it reacts to every detected loss, whereas Cubic halves its window at most once per RTT.

Take-away Both algorithms showcase graceful throughput degradation under random loss, but Cubic retains a slight advantage. Prague's throughput is consistently a few percent lower, likely due to its immediate response to each drop. Still,

this modest gap suggests that Prague's strict loss sensitivity does not significantly affect performance, at least under moderate, uncorrelated loss.

6 Responsible Research

This project follows the usual conventions for transparent, repeatable simulation work. All artefacts required to rebuild the figures and tables are openly available, and the limited ethical issues that arise in a synthetic study are addressed below.

6.1 Reproducibility

Complete public archive. The full ns-3 codebase, helper scripts and figures are published on GitHub under the branch https://github.com/AlexandruTabacaru/ns-3-dev-git-rp/tree/l4s-sims.

One-command rebuild. Each research question includes a short `experiments_rq<i>.md` explainer, a bash script (`run_experiments_rq<i>.sh`) to run the simulations, and a Python script (`analyze_rq<i>.py`) to regenerate all plots and tables. Running the driver script along with the provided analysis scripts reproduces the results reported in the text.

Accessibility. Because the experiments are simulation-only and require no physical testbed, the full study can be reproduced locally on any system with sufficient compute resources.

6.2 Limitations and Disclaimer

Our research relies on ns-3 simulations and is therefore a model-based approximation of real-world network behavior. While simulation offers reproducibility and precise control over variables, it may not capture all the nuances of real deployments.

In particular, our implementation of BBRv3-ECN is not based on an officially released ns-3 module. It was constructed by adapting open-source code made available in a GitHub repository[5] and patching it to reflect aspects of the Linux kernel implementation (as of early 2024). Despite best efforts to align behavior with known BBRv3 internals and ECN marking logic, the result may not fully reflect the performance characteristics of a production-grade implementation.

In addition, all of our experiments build upon the CableLabs ns-3 fork[6], particularly the `l4s-wired` and `l4s-wifi` scenarios. While this repository is widely referenced in the L4S research community, it is not part of the official ns-3 release and may contain unverified assumptions, bugs, or deviations from real hardware behavior. Our own modifications were implemented on top of this foundation, which compounds the simulation–reality gap.

Readers should treat our results as indicative and reproducible within a consistent simulation environment, but not necessarily representative of all real-world deployment conditions. Further validation in hardware testbeds or production networks is encouraged where possible.

[5]https://github.com/Aruuni/ns3-bbrv3/blob/main/src/internet/model/tcp-bbr3.cc

[6]https://github.com/cablelabs/l4s-wifi-ns3

6.3 Use of AI Assistance

Large-language-model tools were employed in supporting roles only:

- **Script generation —** parts of the bash scripts and Python post-processing scripts were produced automatically, then inspected and modified by the authors.
- **Writing aid —** preliminary versions of section overviews and wording alternatives were suggested by AI; every technical statement, number and reference in the final version was inserted or verified by the authors.

6.4 Ethical and Security Notes

The study is simulation-only, involves no personal data and raises no direct security concerns. Revealing unfair interactions between scalable and classic controllers is beneficial for network operators and does not facilitate abuse.

With open code, fixed revisions and scripted runs, we believe the work meets the community standard for *reproducible* experimental research in network simulation.

7 Conclusion and Future Work

This thesis presented a simulation-based analysis of TCP performance under challenging network conditions, focusing on the Low-Latency, Low-Loss, Scalable-Throughput (L4S) architecture. Using ns-3, we evaluated TCP Prague against legacy Cubic across scenarios including RTT jitter, bandwidth changes, mixed traffic, and wireless loss. We also analyzed how Prague competes with ECN-enabled BBRv3 in fully scalable L4S queues.

Our findings show that TCP Prague consistently achieves sub-millisecond delay and near line-rate throughput, outperforming Cubic especially on short-RTT and variable-bandwidth paths. However, fairness is a concern: when sharing a DualPI2 queue, Prague obtains significantly higher throughput than Cubic due to asymmetric marking thresholds. In contrast, Prague and ECN-BBRv3 share bandwidth fairly under most flow mixes, while maintaining low delay.

Future Work

Future work could validate these results through real-world testbed experiments. Our patched BBRv3-ECN implementation follows Linux logic at a high level, but a fully aligned ns-3 model would strengthen confidence. Coexistence fairness may also benefit from tuning AQM parameters such as the DualPI2 threshold. Additional input variations—like flow start times, packet sizes, or more diverse traffic mixes—could further test robustness.

Closing Remarks

While our simulation has its limits, this work offers a reproducible way to stress-test scalable congestion control. By analyzing L4S behavior across delay, throughput, and fairness, we provide practical insights for deployment. We hope these results support a more responsible and informed adoption of L4S in future networks.

References

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). volume 40, page 63–74, New York, NY, USA, August 2010. Association for Computing Machinery.

[2] M. Allman, V. Paxson, and E. Blanton. Rfc 5681: Tcp congestion control, 2009.

[3] Dejene BoruOljira, Karl-Johan Grinnemo, Anna Brunstrom, and Javid Taheri. Validating the sharing behavior and latency characteristics of the l4s architecture. *SIGCOMM Comput. Commun. Rev.*, 50(2):37–44, May 2020.

[4] Bob Briscoe, Koen De Schepper, Marcelo Bagnulo, and Greg White. Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture. RFC 9330, January 2023.

[5] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, 14(5):20–53, October 2016.

[6] Sally Floyd, Dr. K. K. Ramakrishnan, and David L. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, September 2001.

[7] Jim Gettys. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 15(3):96–96, 2011.

[8] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. volume 42, page 64–74, New York, NY, USA, July 2008. Association for Computing Machinery.

[9] Toke Høiland-Jørgensen, Paul McKenney, dave.taht@gmail.com, Jim Gettys, and Eric Dumazet. The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. RFC 8290, January 2018.

[10] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Commun. ACM*, 55(7):42–50, July 2012.

[11] Rong Pan, Preethi Natarajan, Fred Baker, and Greg White. Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem. RFC 8033, February 2017.

[12] Fatih Berkay Sarpkaya, Fraida Fund, and Shivendra Panwar. *To Adopt or Not to Adopt L4S-Compatible Congestion Control? Understanding Performance in a Partial L4S Deployment*, page 217–246. Springer Nature Switzerland, 2025.

[13] Koen De Schepper, Olga Albisser, Olivier Tilmans, and Bob Briscoe. Dual queue coupled aqm: Deployable very low queuing delay for all, 2022.

[14] Koen De Schepper, Bob Briscoe, and Greg White. Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S). RFC 9332, January 2023.

[15] Koen De Schepper, Olivier Tilmans, Bob Briscoe, and Vidhi Goel. Prague Congestion Control. Internet-Draft draft-briscoe-iccrg-prague-congestion-control-04, Internet Engineering Task Force, July 2024. Work in Progress.

[16] Belma Turkovic and Fernando Kuipers. P4air: Increasing fairness among competing congestion control algorithms. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–12, 2020.

[17] Danesh Zeynali, Emilia N. Weyulu, Seifeddine Fathalli, Balakrishnan Chandrasekaran, and Anja Feldmann. Promises and potential of bbrv3. page 249–272, 2024.

A Appendix A — RQ1: 40 ms and 80 ms RTT Cases

This section documents the remaining baseline delay experiments, extending the discussion in §5.1. Figures 11–16 show the corresponding time-series plots.

At **80 ms RTT**, Cubic behaves as follows: **C-H0** (Fig. 14) achieves ∼88 Mbps but shows recurring queue spikes ($Q_{95} = 4.28$ ms). **C-H1** (Fig. 15) sees little throughput change with ±1 ms jitter, but queueing remains bursty ($Q_{95} = 4.54$ ms). **C-H5** (Fig. 16) suffers a severe drop to ∼62 Mbps, and queueing nearly disappears ($Q_{95} \approx 0$ ms) due to frequent backoffs.

This effect is exacerbated by FQ-CoDel's default target delay (typically 5ms), which matches the jitter magnitude. With ±5 ms variation in RTT, packet sojourn times frequently cross the marking/dropping threshold even when queues are shallow. As a result, FQ-CoDel reacts to jitter as if it were congestion, triggering drops that suppress Cubic's window and prevent meaningful queue buildup.

In contrast, Prague holds delay low across all 80 ms cases: **P-H0/H1** maintain ∼89 Mbps throughput with queues consistently below 1 ms. **P-H5** sees throughput fall slightly to ∼72 Mbps, yet queuing remains negligible ($Q_{95} \approx 0$ ms).

At **40 ms RTT**, the trend persists: Cubic (**C-M0**, Fig. 11; **C-M1**, Fig. 12) performs well (>90 Mbps) under low jitter, but drops to ∼69 Mbps under ±5 ms jitter (**C-M5**, Fig. 13), where queue spikes diminish. Prague (**P-M0/M1/M5**) sustains >95 Mbps unless jitter is high, in which case it still achieves ∼78 Mbps with delays consistently under 1 ms. Table 5 summarises throughput and queueing across all appendix runs.

Table 5: Selected Metrics for RQ1 Appendix Cases

ID	Throughput [Mbps]	Q_{95} [ms]	Jitter
C-H0	88.05	4.28	0 ms
C-H1	88.93	4.54	±1 ms
C-H5	62.06	0.00	±5 ms
P-H0	89.31	0.89	0 ms
P-H1	89.39	0.79	±1 ms
P-H5	72.62	0.00	±5 ms
C-M0	92.44	4.84	0 ms
C-M1	91.31	4.56	±1 ms
C-M5	68.60	0.69	±5 ms
P-M0	95.71	0.93	0 ms
P-M1	95.72	0.97	±1 ms
P-M5	77.76	0.37	±5 ms

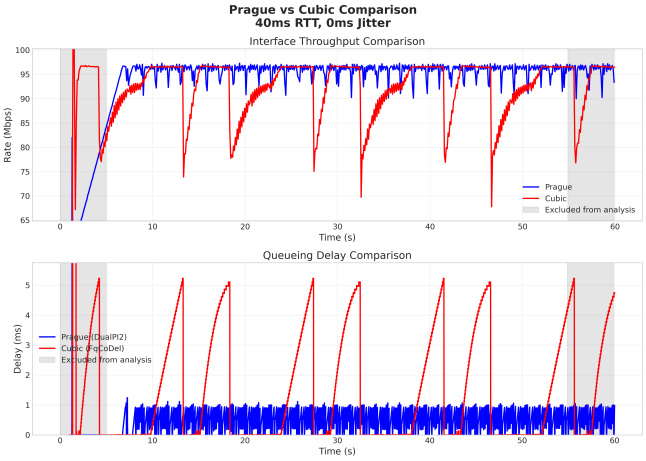

Figure 11: RTT 40 ms, no jitter. Prague vs Cubic.

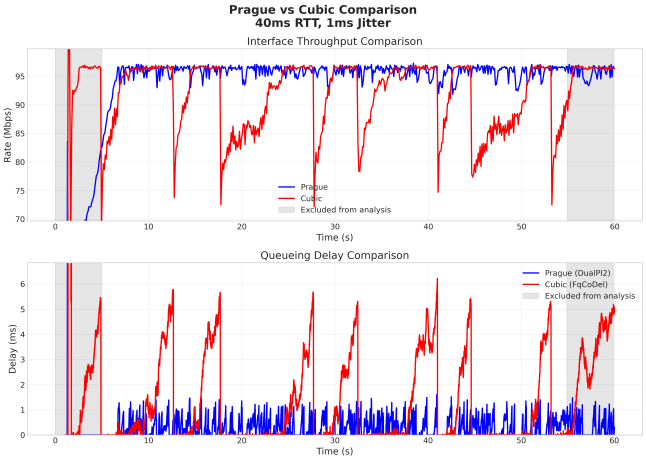

Figure 12: RTT 40 ms, ±1 ms jitter.

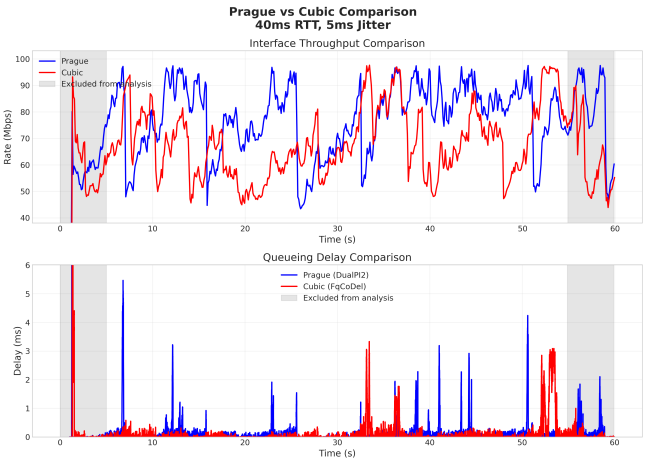

Figure 13: RTT 40 ms, ±5 ms jitter.

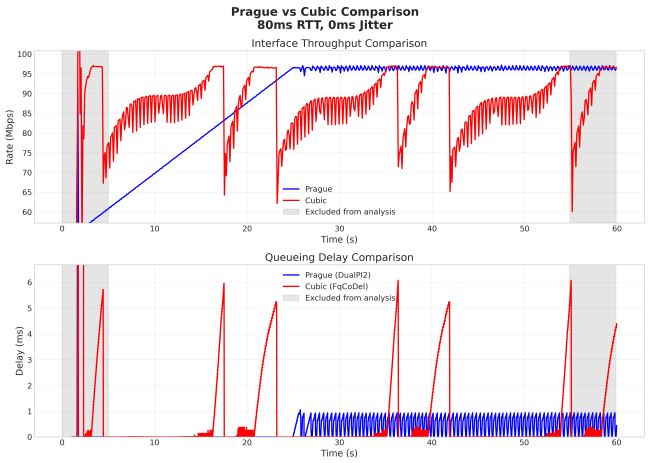

Figure 14: RTT 80 ms, no jitter.

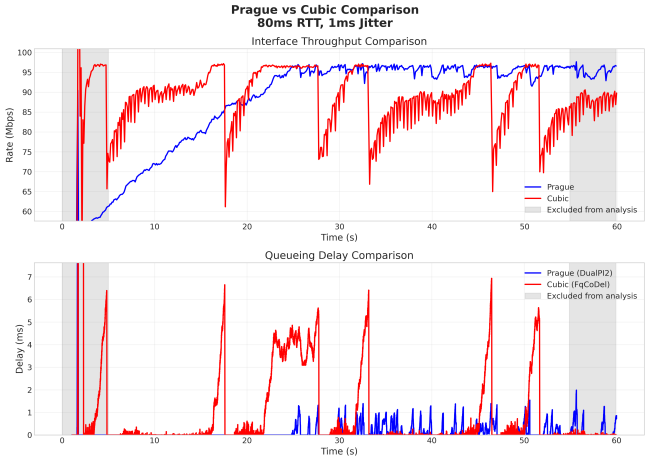

Figure 15: RTT 80 ms, ±1 ms jitter.

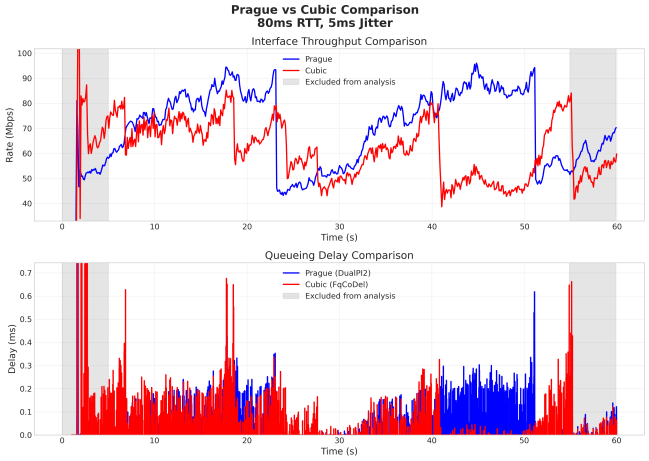

Figure 16: RTT 80 ms, ±5 ms jitter.

B Appendix B — RQ2: Remaining Wi-Fi scenarios

This appendix completes the evaluation of RQ2 by covering the remaining Wi-Fi rate-switch cases. Figures 17 & 20 show that the trends observed in the main text generalise: Prague adjusts promptly and maintains low delay, while Cubic's probe bursts inflate the queue.

In the two remaining **step-up** transitions (MCS 4→7 and 4→9), Prague rapidly ramps up and holds delay near 1 ms, as seen in Figures 17 and 19. Cubic also reaches full rate quickly, but delay remains elevated throughout, with oscillations between 10–30 ms.

For the two **step-down** cases (MCS 9→4 and 7→4), both shown in Figures 18 and 20, Cubic continues to fill the queue after the downgrade, keeping delay in the 10–40 ms range. In contrast, Prague exhibits one brief spike near the rate switch, then quickly settles to sub-2 ms queuing. Table 6 provides a numeric summary of all step-change cases.

Table 6: Wi-Fi Rate Switches – Mean Throughput and Queue Delay

	Throughput [Mbps]		Queue Delay [ms]	
ID	Before	After	Before	After
P-W3	34.44	70.29	0.74	0.46
C-W3	36.50	74.36	9.85	2.84
P-W4	75.36	34.67	0.47	1.09
C-W4	79.93	37.68	3.75	10.03
P-W5	34.44	55.06	0.74	0.58
C-W5	36.50	58.84	9.85	4.68
P-W6	56.79	34.16	0.58	0.86
C-W6	61.33	37.27	5.51	9.30

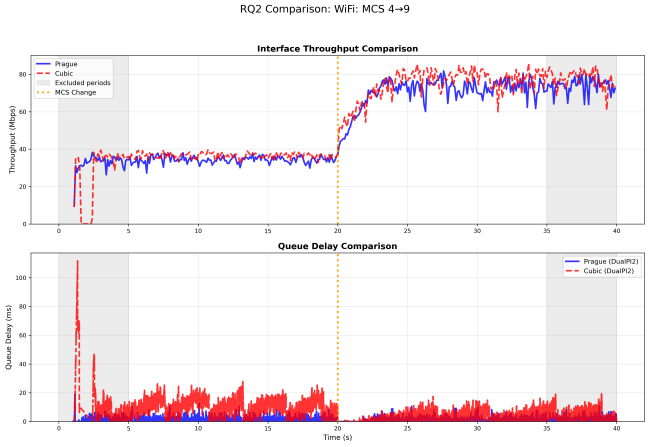

Figure 17: Wi-Fi step-up (MCS 4 → 9).

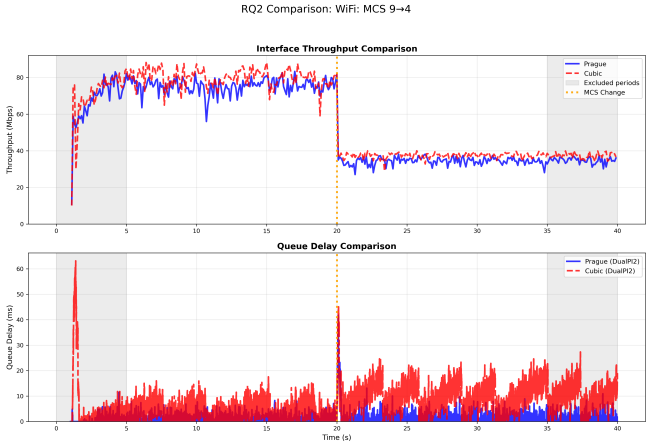

Figure 18: Wi-Fi step-down (MCS 9 → 4).

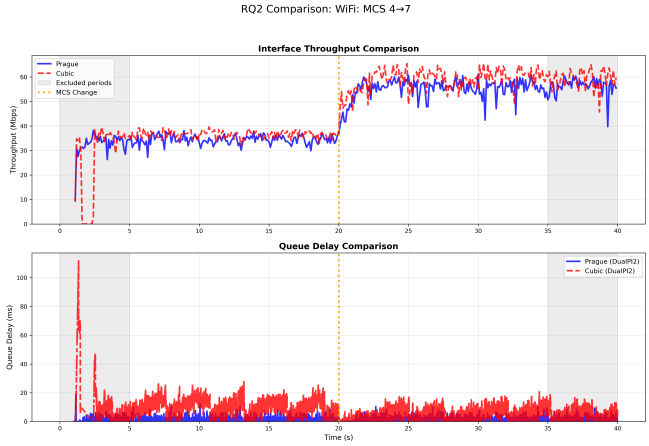

Figure 19: Wi-Fi step-up (MCS 4 → 7).

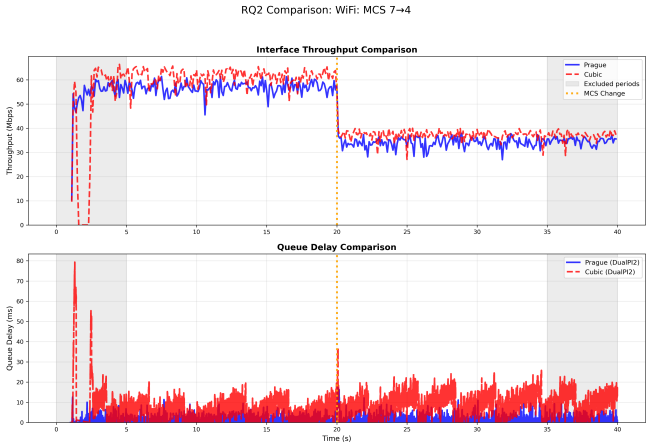

Figure 20: Wi-Fi step-down (MCS 7 → 4).