Small Packets, Big Difference

L4S Congestion Control at Low Bandwidth

Danesh Zeynali*
Max-Planck-Institut für Informatik
dzeynali@mpi-inf.mpg.de

Balakrishnan Chandrasekaran
Vrije Universiteit Amsterdam
b.chandrasekaran@vu.nl

Chia-Yu Chang
Nokia Bell Labs
chia-yu.chang@nokia-bell-labs.com

Koen De Schepper
Nokia Bell Labs
koen.de_schepper@nokia-bell-labs.com

Anja Feldmann*
Max-Planck-Institut für Informatik
anja@mpi-inf.mpg.de

Abstract

TCP Prague is a congestion control algorithm (CCA) for the low latency, low loss and scalable throughput (L4S) architecture, which targets very low queuing delay by leveraging fine-grained explicit congestion notification (ECN) signals. Most prior evaluations of Prague focus on high-rate scenarios and its TCP-friendliness. Prague can also, however, cope with low-bandwidth bottlenecks or compete with elastic and inelastic flows—leaving only a small fair share per flow. In such cases, Prague reduces its segment size to maintain approximately two packets per virtual round trip time (RTT). The current design derives the segment size from both the `pacing rate` and RTT, making it sensitive to noisy RTT conditions and potentially degrading performance. We redesign segment sizing to depend solely on the `pacing_rate` using three schemes—linear, exponential, and logarithmic. We implement these schemes in a modified Prague kernel and evaluate them in our testbed against both the default Prague segment-sizing algorithm and a fixed-segment-size baseline. On throttled links, our schemes reduce retransmissions by more than a factor of 20 and increase feedback granularity. Under both elastic (inter- and intra-CCA) and inelastic competition, they improve throughput stability and fairness.

1 Introduction

Despite the emergence of advanced networking technologies and the increase in available bandwidth for both mobile and fixed connections, network bottlenecks remain prevalent in the Internet. As a result, congestion control remains an active research challenge. Congestion control algorithms (CCAs) rely on network signals such as loss, delay, and explicit congestion notification (ECN) to operate near their optimal point. CUBIC, for example, is a loss-based CCA that probes for bandwidth by filling the queue and oscillating around full utilization [21]. BBR estimates bottleneck bandwidth and round-trip propagation time to operate at Kleinrock's optimal point [13]. DCTCP leverages finer-grained signals via ECN and smart queue management, but its deployment is largely limited to data center environments due to TCP-unfriendliness [2]. TCP Prague [10] extends this approach within the low latency, low loss and scalable throughput (L4S) architecture [9], using improved queue management to achieve TCP-friendliness while targeting ultra-low queuing delay.

Bottlenecks are particularly challenging when multiple flows compete for limited resources. Networks may experience reduced bandwidth, for instance, due to contention from bursty or inelastic cross-traffic at shared bottleneck links, a phenomenon observed in both Internet and datacenter networks [15, 18, 20]. The arrival of several concurrent flows at a bottleneck, such as in a data center incast [11], may also quickly and drastically reduce available bandwidth. In developing regions, low-bandwidth networks are often shared among many users, resulting in persistently constrained links [29]. Physical-layer effects, such as wireless interference, can further reduce a flow's available bandwidth [36–38]. A fundamental challenge in all such environments is the small number of packets in flight per round trip time (RTT), often fewer than two, which offers only a *sparse* (congestion) feedback to the sender.

Scalable CCAs such as Prague, which rely on fine-grained ECN signals, suffer most from such sparse feedback. To address this sparsity of feedback, Prague dynamically reduces its segment size below a certain bandwidth threshold to preserve feedback frequency at low rates. In this work, we redesign and evaluate Prague's dynamic segment sizing to improve performance over highly bandwidth-constrained links. We summarize our contributions as follows.

- ⋆ We identify that the default segment sizing mechanism couples segment size to the RTT, which introduces instability under noisy conditions. We redesign the mechanism to depend solely on the `pacing rate`.
- ⋆ We implement three pacing-rate-driven schemes—linear, exponential, and logarithmic—in the Linux Prague kernel, with runtime-configurable selection.
- ⋆ We demonstrate that our schemes reduce retransmissions by more than a factor of 20 on throttled links, shorten inter-packet gaps, and keep RTT close to the base (i.e., minimum) value.
- ⋆ We show that the logarithmic scheme achieves the best throughput stability under inelastic flooding and the highest fairness against Cubic and BBRv3 [12], at the cost of increased header overhead at very low rates.
- ⋆ We release our Prague implementation as an open-source artifact at https://inet-l4s.mpi-inf.mpg.de/.

2 Background

A CCA can react only to the signals it can observe or infer. The vast body of literature on CCAs has settled on three broad categories of congestion signals: packet loss [5, 21], queuing delay [6, 13], and

*Also affiliated with Saarland University.

ANRW '26, Vienna, Austria

ACM ISBN /2026/07
https://doi.org/10.1145/3822163.3827934

explicit congestion feedback [2, 3, 17, 19]. These signals differ in accuracy (i.e., when congestion is detected) and in the granularity at which they are reported. Together, they determine both *when* and *how aggressively* a CCA can respond to congestion.

Loss Loss-based controllers learn of congestion only *after* a bottleneck buffer (or queue) has overflowed, causing packet loss that duplicate `ACK`s or timeouts signal to the sender. The packet loss signal is unambiguous, but late: The queue is already full, and the only available response is a steep multiplicative reduction.

Delay Delay-based CCAs, in contrast, detect congestion earlier by inferring queue buildup from RTT growth. Compared to loss-based CCAs, they can reduce bottleneck buffer occupancy and enable the sender to respond *proactively*. RTT can, however, vary due to route changes, mobility, wireless MAC effects (e.g., random access and exponential backoff), and competing flows. The effectiveness of a CCA, therefore, rests on its ability to disentangle the congestion signal from all such "noise."

Explicit feedback An explicit feedback signal sidesteps both of the above limitations: It is unambiguous, and it can be generated before the queue grows large enough to hurt application performance. ECN, which was standardized more than two decades ago, provides such an explicit congestion feedback [30]. It has end-host support in every major operating system, although path-wide adoption has been slow, partly impeded by middlebox interference [4, 24, 28, 39]. L4S builds on this foundation by using the ECN codepoints to carry a fine-grained, low-latency signal, paired with an active queue management (AQM) that marks packets long before the bottleneck buffer or queue overflows.

2.1 L4S architecture

Realizing the L4S architecture [9] requires three distinct components working in concert: (i) an AQM that can produce a fine-grained congestion signal, (ii) an accurate and feature-rich congestion signal, and (iii) a *scalable* CCA that can exploit the signal to avoid inflating the queue. Today, a typical L4S implementation uses Dual-Queue Coupled AQM (DualPI2), Accurate ECN (AccECN), and TCP Prague, respectively, for these three components.

DualPI2 The DualPI2 AQM classifies each arriving packet by its ECN codepoint [1]. It dispatches L4S-marked traffic into a low-latency queue and all other traffic into a *classic* queue. Crucially, the two queues do not run independent control loops; instead, their congestion signals are coupled. The L4S queue marks packets early and often, while the classic queue applies the "harder," less frequent signals that loss-based and conventional-ECN controllers expect, i.e., packet drops or RFC 3168 markings. The coupling is calibrated so that a scalable CCA in the L4S-queue and a classic CCA in the other queue converge to similar throughput (over the long term), thus preventing *starvation* of classic flows when they contend for bandwidth with L4S flows [22, 33, 34].

AccECN ECN carries congestion feedback in a single bit, the `ECE` flag, so the sender learns at most that some packet was marked in the past RTT. AccECN [7], in contrast, reports the cumulative number of `CE`-marked packets it has observed, allowing the sender to learn the fraction of marked packets per RTT. This fraction is the key input that scalable CCAs such as DCTCP [2] and Prague [10] exploit.

Prague As a flow's rate grows, classic CCAs require increasingly long sawtooth recovery, which makes congestion marks less frequent and lets queue occupancy grow without bound. A scalable CCA, by contrast, keeps the average duration between congestion marks roughly constant as rate scales, bounding queue occupancy independently of throughput—a property DCTCP [2] first demonstrated in the data center and that Prague [10] applies to the public Internet. TCP Prague is an L4S CCA that estimates the extent of congestion from the fraction (frac) of packets marked with `CE` among the acknowledged packets and updates the congestion estimate α as follows: $\alpha = \alpha + g \cdot (\text{frac} - \alpha)$, where the gain factor, g, is typically $g = 1/16$. Prague updates α once per *virtual* RTT and uses α to scale the reduction of its primary sending variable. The virtual RTT is simply the maximum of the smoothed RTT and a reference value of 25 ms. It reduces RTT-dependence among flows with substantially different path delays and provides a stable reference timescale for low-latency operation.

2.2 Segment sizing in TCP

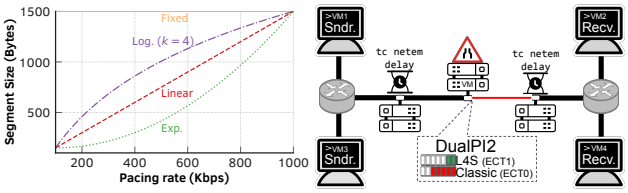
TCP determines the size of transmitted segments from the receiver's advertised `MSS` and its current path maximum transmission unit (PMTU) estimate. During data transmission, a router may determine that a packet exceeds the capacity of a downstream link and return an ICMP "Packet Too Big" (or "Fragmentation Needed") message to the sender. Upon receiving this feedback, the sender updates its PMTU estimate and reduces the size of subsequent TCP segments [14]. When such feedback is unavailable, the sender may instead refine its estimate through packetization-layer probing [27]. In general, TCP does not employ dynamic segment sizing as a congestion control mechanism; instead, it adapts the segment size primarily to satisfy path and receiver constraints.

3 On dynamic segment sizing

Prague in its current form operates in either a window-based mode or a rate-based mode. When the congestion window (in bytes) exceeds a threshold—four standard-sized packets by default, which is typical in high bandwidth scenarios—the controller updates a *fractional* `cwnd` and derives the pacing rate from it (window-based mode); this behavior resembles the standard TCP window-driven approach. Otherwise, it updates the pacing rate directly and derives the corresponding fractional `cwnd` (rate-based mode). In both modes, Prague enforces a minimum packet window of two segments, preserving a minimal packet "pipeline" and remaining compatible with the delayed-`ACK` behavior [5].

To limit queue buildup caused by bursts, Prague employs pacing to spread transmissions over time. In the Prague implementation, pacing acts as the primary transmission ("cadence") control, while the packet window serves mainly as an upper bound on in-flight data. Consequently, even when the packet window reaches its minimum value of two segments [8], the sender does not need to transmit two full-sized packets within one actual RTT; instead, it can pace them over a longer interval and thereby sustain a sending rate below what two full segments per RTT would imply.

This design, however, has a pernicious side effect at very low rates: When the sending rate is smaller than one `MSS` per pacing interval, the sender transmits *fewer than two packets* per virtual



(a) *Segment sizing schemes* **(b)** *Experimental topology*

Figure 1: *The three proposed schemes differ in how aggressively they reduce segment size as the pacing rate falls below 1 Mbps: logarithmic adaptation preserves larger packets at very low rates, while exponential keeps them smaller (left). All schemes are evaluated in a dumbbell testbed with symmetric delay emulation on either side of the bottleneck (right).*

RTT; the ECN feedback, as a consequence, becomes sparse. For instance, at an RTT of 25 ms and a rate of about 1 Mbps, the sender transmits roughly two full-sized packets per virtual RTT; below this rate, feedback frequency simply degrades.

To preserve feedback granularity, Prague reduces the segment size so that approximately two packets are sent per virtual RTT, while pacing controls the average rate. Let r denote the pacing rate and $\tilde{R} = \max(R,\ 25\,\text{ms})$ the virtual RTT. The target segment size is given by $\hat{S} \approx r\tilde{R}/2$, i.e., the value that yields two packets per virtual RTT at the current rate. We bound it by the MSS above and a floor of 150 bytes below. The standard permits reducing the segment size below the MSS, thereby defining the MSS as an upper bound on segment size [16].

> *Takeaways.* The virtual RTT, $\tilde{R}$, varies with competing flows and wireless conditions in the formulation in §2.1. As a consequence, it can cause segment size to fluctuate independently of the actual sending rate, motivating a pacing-rate-only design.

4 Design & Implementation

Below, we redesign Prague's dynamic segment sizing to depend solely on the pacing rate, removing the RTT dependency of the default implementation. We describe three new schemes and their realization in the Linux kernel, followed by the testbed used to evaluate them.

Dynamic approach We determine the segment size solely from the pacing rate, using 1 Mbps as the upper reference point. At this pacing rate and a virtual RTT of 25 ms, the sender transmits approximately two MSS-sized packets per virtual RTT. Below this threshold, we define the segment size as a function of the pacing rate alone, bounded by the MSS (derived from the PMTU) above and the *minimum* segment size (150 Bytes) below. This approach avoids segment-size fluctuations caused by RTT variations (for instance, due to competing flows or wireless channel conditions) and provides a more stable behavior relative to the fair share.

Segment sizing schemes We implement three segment sizing schemes: (a) *linear*, (b) *exponential*, and (c) *logarithmic*. Let r denote the current pacing rate, bounded between $r_{\min} = 100$ kbps and $r_{\max} = 1$ Mbps. We fix the minimum segment size to $S_{\min} = 150$ B, which corresponds to the minimum payload size used by the original Prague implementation. The MSS, denoted by $S_{\max}$, corresponds to the connection's PMTU-limited MSS. We normalize the pacing rate to the interval $[0, 1]$ as $x = (r - r_{\min})/(r_{\max} - r_{\min})$. We then compute the segment size $S(r)$ as follows.

$$S(r) = \begin{cases} S_{\max}, & \textit{(Fixed)} \\ S_{\min} + (S_{\max} - S_{\min})\, x, & \textit{(Linear)} \\ S_{\min} + (S_{\max} - S_{\min})\, \dfrac{\ln(1+kx)}{\ln(1+k)}, & \textit{(Logarithmic)} \\ S_{\min} + (S_{\max} - S_{\min})\, x^{p}, & \textit{(Exponential)} \end{cases}$$

where k is the logarithmic shape parameter and p is the exponential shape exponent. In our implementation, we use $k = 4$ and $p = 2$. The linear approach scales the segment size linearly with the pacing rate. The exponential approach uses smaller segment sizes at low rates and increases them more aggressively as the rate approaches $r_{\max}$. The logarithmic approach does the opposite: It increases segment sizes more aggressively at low rates and more conservatively as the rate approaches $r_{\max}$ (Fig. 1a). For all schemes, the segment size remains bounded between $S_{\min}$ and $S_{\max}$, ensuring that only the mapping between pacing rate and segment size differs across the evaluated approaches. We evaluate all three against the default Prague implementation and a fixed segment size baseline.

Implementation We implemented all three schemes in Prague within the Linux kernel. We extended the Prague kernel module to make segment sizing configurable at runtime, allowing the active scheme to be selected without recompilation. Our implementation builds on the L4STeam Linux TCP Prague kernel [26] (branch `l4steam-6.6.y`).[1]

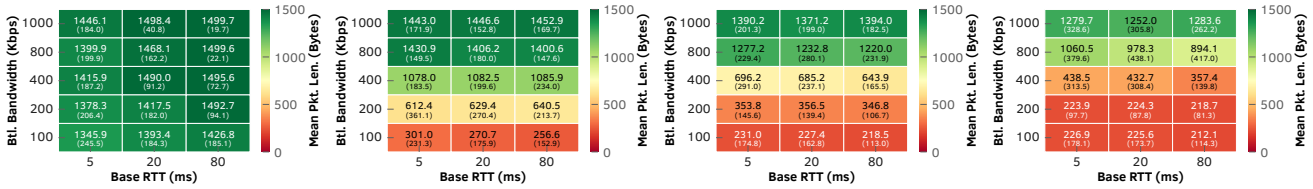
5 Evaluation

We evaluate all schemes in a custom testbed consisting of seven machines: four act as senders and receivers, one serves as the bottleneck and traffic shaping node, and two introduce emulated delay. The shaping node runs Linux kernel 5.15.72 with `iproute2` from the L4STeam repository [25][2], ensuring no unintended bottlenecks arise outside the middle node. The Prague hosts use the modified kernel described in Section 4, while BBRv3 experiments use the Google-patched kernel (`6.4.0+v3`).

Fig. 1b shows the dumbbell topology; all machines are interconnected via commodity switches. To emulate base RTT, we split the target delay into four equal parts and apply it symmetrically to both data packets and ACKs on either side of the bottleneck, using `tc-netem` [23]. Traffic is generated with `iperf2`; flooding scenarios use `hping3` [32].

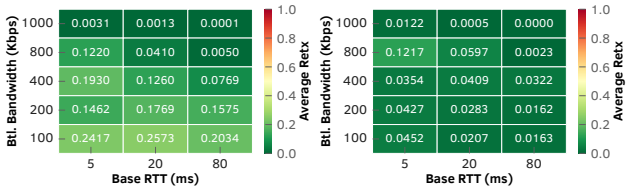
We evaluate all the schemes across four scenarios that reflect conditions under which Prague is likely to encounter low fair shares: a single flow on a (highly) throttled link (§5.1), multiple homogeneous Prague flows under contention (§5.2), competition with inelastic traffic (§5.3), and competition with heterogeneous CCAs (§5.4). In each scenario, we compare packet sizing behavior, retransmission rate, latency, and throughput stability across all schemes.

[1] Commit `a76b708b`.
[2] Commit `724c4fbe`.



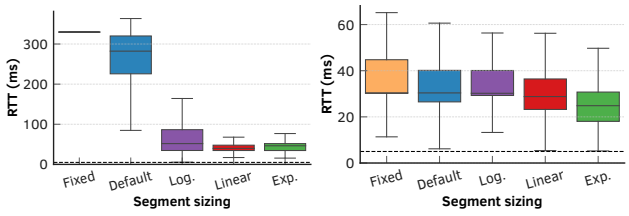
(a) *Default* **(b)** *Logarithmic* **(c)** *Linear* **(d)** *Exponential*

Figure 2: *The default implementation keeps packet sizes near the* MSS *even at 100 Kbps, while pacing-rate-driven schemes adapt segment size to the available bandwidth. Each cell shows the average (top) and standard deviation (bottom) of packet lengths.*



(a) *Default* **(b)** *Logarithmic*

Figure 3: *Pacing-rate-driven segment sizing reduces retransmissions by more than a factor of 20 across the tested range, while the default scheme shows no significant improvement. Linear and exponential results are similar.*



(a) *100 Kbps,* 5 ms *base RTT* **(b)** *800 Kbps,* 5 ms *base RTT*

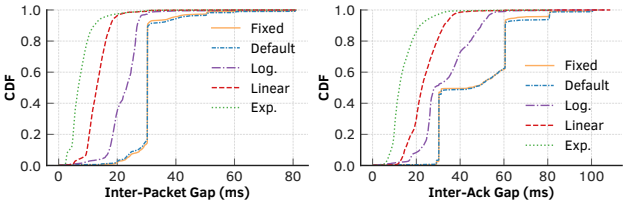
Figure 4: *Pacing-rate-driven schemes keep average RTT close to the base RTT; the logarithmic approach achieves a lower tail compared to the default, despite serialization delay dominating at these rates.*

5.1 Single flow

We first consider a single Prague flow on a link shaped to between 100 Kbps and 1 Mbps—the regime in which dynamic segment sizing is active. We test three base RTT values: 5 ms, representative of short paths observed in RIPE Atlas measurements [31]; 20 ms, just below the virtual RTT floor of 25 ms; and 80 ms, which exceeds it. Since competing flows are absent, the RTT-fairness effects of the virtual RTT do not apply here.

We generate traffic with `iperf` over a link shaped by `tc` HTB. The bottleneck runs DualPI2 with a 300 KB buffer[3] and a `step threshold` of 1 ms, which controls the AQM's transition from marking to dropping.

Per Fig. 2, the default implementation retains packet sizes near the MSS even at 100 Kbps, thus failing to adapt to the available bandwidth. The three pacing-rate-driven schemes, by contrast, reduce



(a) *400 Kbps,* 80 ms *base RTT* **(b)** *400 Kbps,* 80 ms *base RTT*

Figure 5: *Pacing-rate-driven schemes transmit more packets and receive more* ACK*s per RTT than the default and fixed baselines, increasing feedback granularity for rate adaptation.*

segment size proportionally to the rate, producing substantially smaller packets across the entire tested range.

The pacing-rate-driven schemes reduce retransmissions by more than a factor of 20 across the tested range (Fig. 3) using the default DualPI2 configuration without tuning the PI2 controller or drop/marking thresholds, thereby directly lowering the RTT experienced by the flow (Fig. 4). At these rates, serialization delay dominates, so RTT differences across schemes are modest. The logarithmic approach, according to this figure, consistently achieves a lower RTT tail compared to the default. Smaller packets also reduce the inter-packet and inter-ACK gaps (Fig. 5): At 400 Kbps with 80 ms base RTT, the pacing-rate-driven schemes deliver more packets and ACKs per RTT, offering the CCA with fine-grained feedback for rate adaptation.

Smaller segments increase header overhead relative to payload. Fig. 7 shows the normalized goodput[4] for the default and logarithmic schemes: Despite reducing retransmissions significantly, the logarithmic approach yields lower normalized goodput due to its smaller packet sizes.

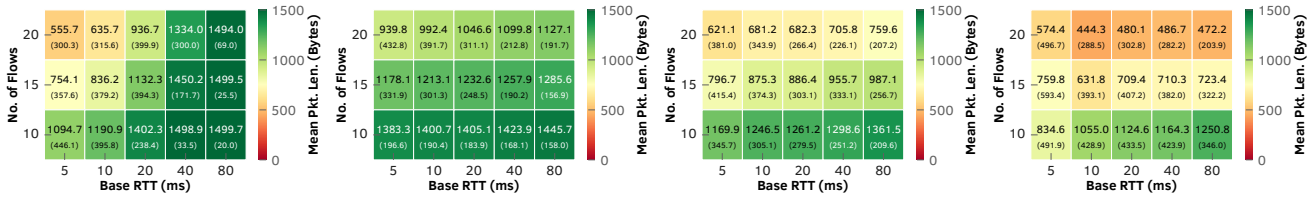
> *Takeaways.* The default scheme keeps packet sizes near the MSS at low rates, starving Prague of ECN feedback. Pacing-rate-driven schemes cut retransmissions by over 20×, tighten inter-packet gaps, and keep RTT close to the base—at the cost of increased header overhead. Among the three, the logarithmic scheme best balances feedback granularity and packet efficiency.

5.2 Multiple homogeneous flows

We extend the evaluation to a 10 Mbps bottleneck shared by 10, 15, or 20 homogeneous Prague flows, with base RTT values ranging

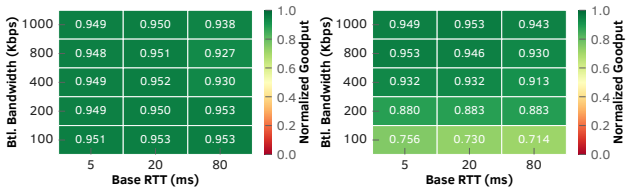
[3]Sufficient for approximately 200 MSS-sized packets.

[4]Total received payload divided by the theoretical maximum payload without headers.



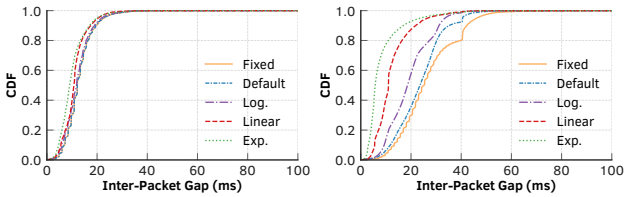
(a) *Default* **(b)** *Logarithmic* **(c)** *Linear* **(d)** *Exponential*

Figure 6: *Dynamic segment sizing reduces packet sizes across all algorithms under multi-flow bottleneck contention. The default implementation adapts to both RTT and the pacing rate, whereas the logarithmic approach produces larger packets on average and remains largely independent of RTT.*



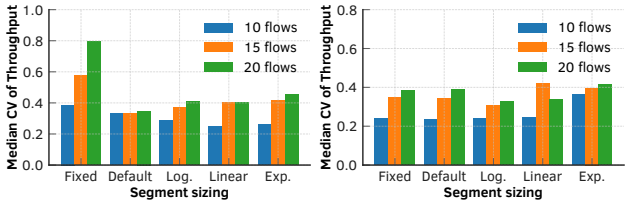
(a) *Default* **(b)** *Logarithmic*

Figure 7: *Smaller packets increase header overhead: the logarithmic scheme achieves the lowest retransmission rate but pays a goodput penalty relative to the default.*



(a) *10 flows,* 40 ms *base RTT* **(b)** *20 flows,* 40 ms *base RTT*

Figure 8: *At smaller fair shares, default and fixed approaches show similar inter-packet gaps, whereas dynamic segment sizing yields smaller gaps via better packet size adaptation.*



(a) 10 ms *base RTT* **(b)** 40 ms *base RTT*

Figure 9: *At higher RTT, pacing-rate-driven schemes reduce throughput CV relative to the default and fixed baselines; at* 10 ms*, the default remains competitive.*

from 5 ms to 80 ms. This setup produces per-flow fair shares below 1 Mbps, placing each flow in the dynamic-segment-sizing regime, while avoiding the extreme serialization delays of the single-flow scenario. As expected, more competing flows reduce the fair share per flow and decrease packet sizes (Fig. 6); the pacing-rate-driven schemes maintain more consistent packet sizes across different base RTT values than the default. Retransmissions are negligible across all cases—early L4S feedback prevents overshoot—and RTT remains close to the base; we omit these results for brevity.

Fig. 8 shows that at 10 flows—where each flow receives roughly 1 Mbps—all schemes produce similar inter-packet gaps. At 20 flows, the default behaves like the fixed baseline, while the pacing-rate-driven schemes achieve tighter (or smaller) gaps through smaller packet sizes, thus delivering more frequent feedback to the CCA.

To quantify throughput stability, we compute the coefficient of variation (CV[5]) across all flows and report the median over three runs. At higher RTT (e.g., 40 ms), pacing-rate-driven schemes reduce throughput CV relative to the default and fixed baselines (Fig. 9). The logarithmic scheme shows the greatest improvement at 20 flows. At 10 ms, the default achieves lower CV: This behavior is consistent with its tendency to produce smaller packets at short RTT values, naturally increasing feedback frequency.

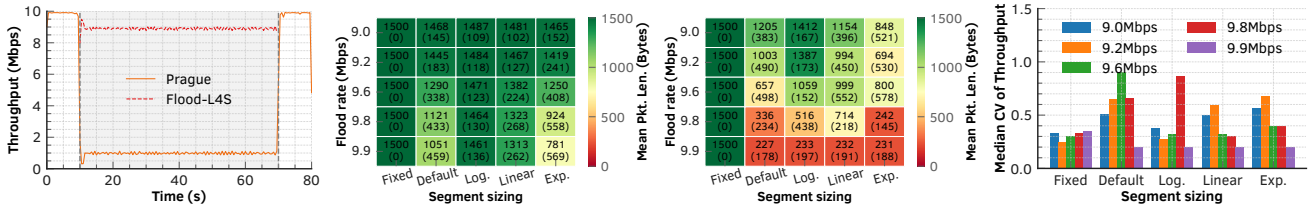
Takeaways. Under multi-flow contention with sub-1 Mbps fair shares, pacing-rate-driven schemes reduce inter-packet gaps and stabilize throughput. These benefits are more pronounced at higher RTT, while the default implementation remains competitive at lower RTT.

[5]CV = σ/μ, where σ and μ are the standard deviation and mean throughput over 50 ms bins.

5.3 Competition with inelastic traffic

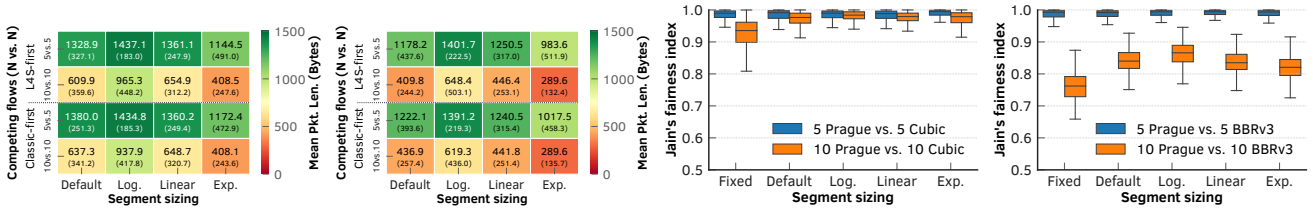
Consider a 10 Mbps bottleneck shared by a single Prague flow and an inelastic SYN flood generated using `hping3`. The SYN flood starts after 10 s and its rate varies between 9 Mbps and 9.9 Mbps, which leaves a residual bandwidth between 1 Mbps and 100 Kbps for Prague. We test two conditions: In the first, flood packets are `ECT(0)`-marked, and in the second, `ECT(1)`-marked SYN flood packets enter the L4S queue alongside the Prague flow. Fig. 10a illustrates the throughput of flows in the 9 Mbps setting.

Under `ECT(0)` flooding, DualPI2's coupling mechanism drops excess classic-queue traffic before it can starve the L4S queue [35], so Prague retains most of the bottleneck bandwidth and packet sizes remain near the MSS (Fig. 10b). Under `ECT(1)` flooding, the flood shares the L4S queue directly with Prague, offering no such protection; packet sizes drop sharply as Prague is squeezed to a small residual share (Fig. 10c). The logarithmic scheme achieves



(a) *Flood of 9 Mbps* **(b)** *ECT(0) flood on a 10 Mbps link* **(c)** *ECT(1) flood on a 10 Mbps link* **(d)** *ECT(1) Flood of 9-9.9 Mbps*

Figure 10: *Under* `ECT(0)` *flooding, DualPI2 protects Prague; under* `ECT(1)` *flooding, Prague only obtains a small residual share. The logarithmic scheme achieves the most stable throughput in both cases.*



(a) *N × Prague vs. N × Cubic* **(b)** *N × Prague vs. N × BBRv3* **(c)** *Cubic first* **(d)** *BBRv3 first*

Figure 11: *Under inter-CCA competition with sub-1 Mbps fair share, dynamic segment sizing in Prague adapts packet sizes without harming competitiveness and improves Jain's fairness index (JFI) against both Cubic and BBRv3.*

the most stable throughput in both conditions (Fig. 10d), with the exception of the SYN 9.8 Mbps flood rate.

> *Takeaways.* DualPI2 shields Prague from classic (`ECT(0)`) floods via queue coupling, but offers no such protection against L4S (`ECT(1)`) floods. Dynamic segment sizing remains stable under both conditions and improves throughput stability in most cases, particularly with the logarithmic scheme.

5.4 Multiple heterogeneous flows

We evaluate N Prague flows competing with N Cubic or BBRv3 flows ($N \in \{5, 10\}$) over a 10 Mbps bottleneck with 20 ms base RTT. We select Cubic as the default Linux CCA, and BBRv3 as a widely deployed, aggressive alternative [40]. In both cases, we disable ECN, causing these flows to compete as classic CCAs. Each experiment runs in two orders—Prague first or the competing CCA first—and flows compete for 60 s. As contention increases, Prague's fair shares fall below 1 Mbps, activating dynamic segment sizing (Fig. 11a and Fig. 11b). The competition with BBRv3 yields smaller packet sizes than with Cubic, which reflects BBRv3's aggressive behavior. All dynamic-segment-sizing schemes improve Jain's fairness index (JFI) over the fixed baseline in both competition scenarios (Fig. 11c, 11d). Against Cubic, fairness remains close to 1 (i.e., ideal). Against BBRv3 at 10 flows per CCA, the median JFI degrades significantly, yet the logarithmic scheme consistently achieves the highest JFI even in these scenarios. The order of flow arrivals has *no* significant effect on the results.

> *Takeaways.* Dynamic segment-sizing schemes preserve the TCP-friendliness of Prague and improve fairness under inter-CCA competition, even against aggressive flows such as BBRv3.

6 Concluding remarks

Prague struggles at low bandwidth scenarios because transmitting full-sized packets at very low rates makes the ECN feedback sparse, which in turn degrades rate adaptation. The default segment sizing approach partially addresses this issue but ties segment size to the RTT, introducing instability under noisy conditions.

We redesigned our dynamic segment-sizing approach to depend solely on the pacing rate and implemented three schemes—linear, exponential, and logarithmic—in the Linux kernel. Across four evaluation scenarios, our sizing schemes reduce retransmissions by more than a factor of 20 on throttled links, reduce inter-packet gaps, and stabilize throughput under both elastic and inelastic competition. The logarithmic sizing scheme consistently outperforms the others: It achieves the lowest retransmission rates, the best throughput stability under inelastic floods, and the highest fairness against Cubic and BBRv3. These benefits come at the cost of increased header overhead, which reduces normalized goodput at very low rates. Evaluating a single aspect of a CCA, as we do in this paper, remains challenging because many configurable parameters influence performance. We design our experiments to isolate the component under study and make the evaluation reproducible. Generalizing the results, however, requires further exploration of the parameter space. We leave this investigation to future work.

Acknowledgements

We thank the reviewers profusely for their constructive feedback. Danesh Zeynali conducted part of this work during his internship at Nokia Bell Labs, Belgium. This work was partially funded by the imec.icon project MAGNOLIA, co-financed by imec and Flanders Innovation & Interpreneurship (VLAIO) under project nr. HBC.2025.0436.

References

[1] Olga Albisser, Koen De Schepper, Bob Briscoe, Olivier Tilmans, and Henrik Steen. 2019. DUALPI2 - Low Latency, Low Loss and Scalable Throughput (L4S) AQM. In *Technical Conference on Linux Networking (Netdev)).*

[2] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). In *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).*

[3] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. 2023. Bolt:{Sub-RTT} Congestion Control for {Ultra-Low} Latency. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI).*

[4] Steven Bauer, Robert Beverly, and Arthur Berger. 2011. Measuring the State of ECN Readiness in Servers, Clients,and Routers. In *ACM Internet Measurement Conference (IMC).*

[5] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. 2009. TCP Congestion Control. RFC 5681.

[6] L.S. Brakmo and L.L. Peterson. 1995. TCP Vegas: end to end congestion avoidance on a global Internet. *IEEE Journal on Selected Areas in Communications* (1995).

[7] Bob Briscoe, Mirja Kühlewind, and Richard Scheffenegger. 2026. More Accurate Explicit Congestion Notification (AccECN) Feedback in TCP. RFC 9768.

[8] Bob Briscoe and Koen De Schepper. 2019. Scaling TCP's Congestion Window for Small Round Trip Times. https://arxiv.org/abs/1904.07598

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

[10] Bob Briscoe, Koen De Schepper, Olivier Tilmans, Mirja Kühlewind, Joakim Misund, Olga Albisser, and Asad Sajjad Ahmed. 2019. Implementing the 'Prague Requirements' for L4S. In *Technical Conference on Linux Networking (Netdev)).*

[11] Christopher Canel, Balasubramanian Madhavan, Srikanth Sundaresan, Neil Spring, Prashanth Kannan, Ying Zhang, Kevin Lin, and Srinivasan Seshan. 2024. Understanding Incast Bursts in Modern Datacenters. In *ACM Internet Measurement Conference (IMC).*

[12] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Steven Yang, David Morley, Soheil Hassas Yeganeh, Ian Swett, Bin Wu, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, and Van Jacobson. 2023. BBRv3: Algorithm Bug Fixes and Public Internet Deployment. https://datatracker.ietf.org/meeting/117/materials/slides-117-ccwg-bbrv3-algorithm-bug-fixes-and-public-internet-deployment-00. [IETF 117; CCWG].

[13] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* (2016).

[14] Dr. Steve E. Deering and Jeffrey Mogul. 1990. Path MTU discovery. RFC 1191.

[15] Martin Duke and Gorry Fairhurst. 2025. Specifying New Congestion Control Algorithms. RFC 9743.

[16] Wesley Eddy. 2022. Transmission Control Protocol (TCP). RFC 9293.

[17] Seifeddine Fathalli, Emilia N. Weyulu, Danesh Zeynali, Balakrishnan Chandrasekaran, and Anja Feldmann. 2026. Network-Assisted Congestion Feedback. (2026).

[18] Ehab Ghabashneh, Yimeng Zhao, Cristian Lumezanu, Neil Spring, Srikanth Sundaresan, and Sanjay Rao. 2022. A microscopic view of bursts, buffer contention, and loss in data centers. In *ACM Internet Measurement Conference (IMC).*

[19] Fernando Gont. 2012. Deprecation of ICMP Source Quench Messages. RFC 6633.

[20] Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. 2022. Elasticity detection: a building block for internet congestion control. In *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).*

[21] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review.*

[22] Lukas Hagen and Anna Brunstrom. 2025. Reproducing and Solving a Fallback Issue in TCP Prague. In *Applied Networking Research Workshop (ANRW).*

[23] Stephen Hemminger, Fabio Ludovici, and Hagen Paul Pfeifer. 2023. tc-netem(8) — Linux manual page. https://man7.org/linux/man-pages/man8/tc-netem.8.html. [Last accessed: April 19, 2024].

[24] Mirja Kühlewind, Sebastian Neuner, and Brian Trammell. 2013. On the State of ECN and TCP Options on the Internet. In *Passive and Active Measurement (PAM).*

[25] L4S Team. 2026. IPRoute2. https://github.com/L4STeam/iproute2.git. [Accessed: April 23, 2026].

[26] L4S Team. 2026. TCP Prague. https://github.com/L4STeam/linux/tree/l4steam-6.6.y. [Accessed: April 23, 2026].

[27] Matt Mathis and John Heffner. 2007. Packetization Layer Path MTU Discovery. RFC 4821.

[28] Alberto Medina, Mark Allman, and Sally Floyd. 2004. Measuring Interactions Between Transport Protocols and Middleboxes. In *ACM Internet Measurement Conference (IMC).*

[29] Ayush Pandey, Matteo Varvello, Syed Ishtiaque Ahmed, Shurui Zhou, Lakshmi Subramanian, and Yasir Zaki. 2025. MAML: Towards a Faster Web in Developing Regions. In *The Web Conference (WWW).*

[30] K. Ramakrishnan, S. Floyd, and D. Black. 2001. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168. (Sept. 2001).

[31] RIPE NCC. 2026. RIPE Atlas Internet Maps. https://atlas.ripe.net/maps/. Accessed: 2026-04-30.

[32] Salvatore Sanfilippo and the hping project. 2026. hping3. [Accessed: April 23, 2026]. https://github.com/antirez/hping

[33] Fatih Berkay Sarpkaya, Fraida Fund, and Shivendra Panwar. 2025. To Adopt or Not to Adopt L4S-Compatible Congestion Control? Understanding Performance in a Partial L4S Deployment. In *Passive and Active Measurement (PAM).*

[34] Fatih Berkay Sarpkaya, Ashutosh Srivastava, Fraida Fund, and Shivendra Panwar. 2024. To switch or not to switch to TCP Prague? Incentives for adoption in a partial L4S deployment. In *Applied Networking Research Workshop (ANRW).*

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

[36] Anika Schwind, Florian Wamser, Stefan Wunderer, Christian Gassner, and Tobias Hoßfeld. 2019. Mobile Internet Experience: Urban vs. Rural – Saturation vs. Starving? https://arxiv.org/abs/1909.07617

[37] Srikanth Sundaresan, Xiaohong Deng, Yun Feng, Danny Lee, and Amogh Dhamdhere. 2017. Challenges in inferring internet congestion using throughput measurements. In *ACM Internet Measurement Conference (IMC).*

[38] Xinlei Yang, Hao Lin, Zhenhua Li, Feng Qian, Xingyao Li, Zhiming He, Xudong Wu, Xianlong Wang, Yunhao Liu, Zhi Liao, Daqiang Hu, and Tianyin Xu. 2022. Mobile access bandwidth in practice: measurement, analysis, and implications. In *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).*

[39] Danesh Zeynali, Denis Eminov, Taha Albakour, Balakrishnan Chandrasekaran, and Anja Feldmann. 2026. The State of ECN Adoption in the Internet.

[40] Danesh Zeynali, Emilia N. Weyulu, Seifeddine Fathalli, Balakrishnan Chandrasekaran, and Anja Feldmann. 2024. Promises and Potential of BBRv3. In *Passive and Active Measurement (PAM).*