POSTER: Expanding the Design Space for in-Network Congestion Control on the Internet

Ayush Mishra[⋆], Harsh Gondaliya[‡], Lingesh Kumaar[⋆†], Archit Bhatnagar[⋆], Raj Joshi[⋆◇], Ben Leong[⋆]
[⋆]National University of Singapore [†]BITS Pilani [‡]UC San Diego [◇]Harvard University

CCS CONCEPTS

• **Networks** → **Programmable networks**; **In-network processing**; **Transport protocols**; **Public Internet**.

KEYWORDS

Programmable Networks; p4; Congestion Control

ACM Reference Format:
Ayush Mishra, Harsh Gondaliya, Lingesh Kumaar, Archit Bhatnagar, Raj Joshi, Ben Leong. 2024. POSTER: Expanding the Design Space for in-Network Congestion Control on the Internet. In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM Posters and Demos '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3672202.3673733

1 INTRODUCTION

The Internet has historically been built on the assumption of a *smart edge* and a *dumb network*. This design philosophy is also reflected in end-to-end Internet congestion control. The stability and general fairness of the current Internet are heavily reliant on end hosts implementing congestion control algorithms (CCAs) correctly. Maintaining fairness between flows on the Internet with end-host-based algorithms is a hard problem. Even when the Internet was dominated by a homogeneous mix of loss-based AIMD CCAs, unfairness was an issue between flows with different RTTs [11]. While some delay-based algorithms were able to achieve RTT fairness, they became susceptible to unfairness arising from a late-comer advantage [1].

Today, the Internet has a more diverse mix of CCAs than ever before [13]. Algorithms with contrasting congestion control philosophies, such as CUBIC and BBR, coexist, leading to well-known fairness issues [8, 17]. Additionally, undocumented and rogue CCAs are being deployed [13]. The net result is that unfairness is a reality of the Internet today.

From a game-theoretic point of view, end hosts have little incentive to ensure fairness. It is therefore not surprising that new CCAs are deployed on the Internet with little regard to their impact on existing traffic while the fairness issues are addressed retroactively (e.g. the deployment of BBRv1 [5] and BBRv2 [6]). This is clearly not a healthy trend and we need a way to ensure that CCAs do not "overstep" their limits when they compete with other flows on the Internet, or we might risk another congestion collapse [9].

ACM SIGCOMM Posters and Demos '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0717-9/24/08.
https://doi.org/10.1145/3672202.3673733

1.1 In-Network Solutions to Internet Congestion Control

We argue that the network is the perfect place to deploy mechanisms that can "limit" the amount of unfairness that a flow can inflict on competing traffic. This is because the network has information on the state of congestion as well as the volume of competing traffic to determine the bounds within which each flow must operate. Note that our aim here is modest: we want to bound a flow's aggression and not attempt to maintain *perfect* per-flow fairness. This is because not all flows are the same and per-flow fairness is not always the most desirable outcome. For example, per-flow fairness can unfairly benefit applications that open a large number of connections. Ware et al. argued for a similar goal and defined a metric for aggression, but stopped short of providing a solution [16].

To this end, any practical in-network congestion control solution for the Internet must address the following challenges:

(1) **Agnostic to end hosts.** Given the current heterogeneity of the Internet's congestion control landscape, we should be end-host agnostic and work with all CCAs.
(2) **"Enforce" send rate reduction.** Since the end hosts have little incentive to maintain fairness, an ideal solution cannot rely on them actively collaborating with the in-network CCA. We should be able to slow senders down even when they ignore implicit and explicit congestion signals.

Several existing solutions can satisfy these requirements, with most of them targeting per-flow fairness. AFQ [15] and SP-PIFO [2] try to simulate fair queuing in commodity switches. However, they require multiple queues with many priority levels and do not scale to a large number of flows. Classic AQMs [7, 12] use admission control and packet drops to force a flow to back off. However, this approach would not work with the likes of BBRv1, which is largely loss-agnostic. Cebinae [18] aims to nudge competing flows towards long-term max-min fairness, but it takes seconds to converge and is hard to parameterize.

1.2 The Case for an `rwnd`-based Solution

The appropriate algorithm for determining the limits within which each flow must operate is likely to be subjective. However, we argue that the TCP receive window (`rwnd`) is a natural clamping mechanism for limiting a flow's aggression. While it is supposed to be used for flow control, it can be easily modified by network switches to artificially limit how much data a flow can send. Since, by default, the TCP stack does not allow a flow to keep more than `rwnd` bytes in flight, an `rwnd`-based solution would be end-host agnostic and effective without explicit collaboration from the end hosts. Moreover, as long as the set `rwnd` is not greater than the `rwnd` set by the receiver, it

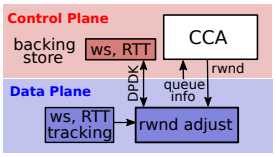

Figure 1: Flowtamer design.

would not interfere with the flow control mechanism. Therefore, we propose *Flowtamer*, a new[1] `rwnd`-based framework for deploying in-network CCAs on commodity programmable switches.

2 FLOWTAMER

Our goal is to provide an actuation framework to easily implement `rwnd`-based in-network CCAs on programmable switches. However, it is not straightforward to build such a framework because of the following challenges:

(1) **Window scaling:** The `rwnd` value in TCP packets is scaled by the window scaling (WS) factor which is communicated only during the 3-way handshake. Therefore, we need to keep track of the WS for each flow.

(2) **TCP Checksum:** Modifying the `rwnd` affects the TCP checksum whose computation includes the payload. We need to update the TCP checksum even when the switch dataplane pipelines do not have access to the payload.

(3) **Compensating for different RTTs:** Since competing flows can have varying RTTs, an `rwnd`-based in-network CCA would need to account for it for a fairer `rwnd` enforcement. Therefore, we need to measure and make available per-flow RTT information.

(4) **Scaling per-flow state:** Maintaining the required per-flow state (WS and RTT) is non-trivial due to memory constraints in the switch dataplane.

As shown in Figure 1, Flowtamer spans both the data and control planes on a switch and provides high-level APIs to implement `rwnd`-based in-network CCAs. In particular, the APIs provide the CCA real-time access to port queue lengths and a way to set `rwnd` that would be applied to TCP (ACK) packets flowing through the dataplane. Flowtamer also provides dataplane-based measurements of per-flow RTT and WS that is used for `rwnd` adjustment. At runtime, Flowtamer works in periodic rounds of configurable duration. In each round, the CCA logic uses queue size and other metrics and performs congestion control by adjusting the `rwnd`. The CCA can use the per-flow RTT info to proportionally scale the `rwnd` for flows with different RTTs. The final `rwnd` value that is set into the TCP packet header is adjusted by Flowtamer depending on the flow's WS.

Design. For tracking per-flow WS, Flowtamer intercepts the TCP 3-way handshake in the dataplane. Flowtamer measures the per-flow RTT in the dataplane by using a modified version of Dart [14]. After the new `rwnd` (as decided by the CCA) is applied to the packets, Flowtamer uses properties of 1's complement arithmetic to recompute the TCP checksum without accessing the payload.

Scalability. Although Flowtamer requires per-flow state, our key insight is that we do not need to maintain this state for short flows that last just an RTT. In the 2019 CAIDA Internet traces [4], we found that only 10% of the flows are longer than 10 packets (equivalent of initial `cwnd`) and 95% of the flows last for less than 10 seconds. We ran a simulation on these traces and found that we need only enough state for at most 65K flows if we tracked only the long flows (>10 pkts) and evicted the state for flows that were inactive for 10 seconds. We use this insight to maintain state for a limited number of flows in the dataplane. For the evicted flows' state, we implement a backing store in the control plane DRAM.

[1]While receiver-based `rwnd` adjustment has been previously used to limit bufferbloat in cellular networks [10], it does not work in the Internet context where the buffer is shared by thousands of flows.

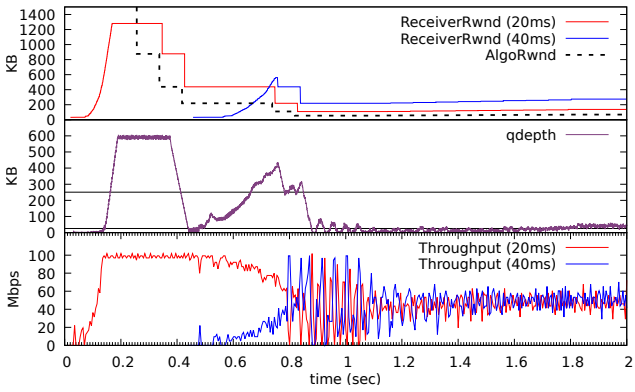

Figure 2: BBRv1 flows (20 ms, 40 ms) on a 100 Mbps bottleneck link with a PoC in-network CCA (§2.1).

Implementation. Flowtamer is implemented on an Intel Tofino switch using 1K lines each of P4 and C++ code. To facilitate a fast channel between the data and control planes for per-flow states, we use DPDK over two 10G switch-local interfaces between the two planes.

2.1 Proof of Concept (PoC) CCA

As a proof of concept, we implemented a simple threshold-based CCA in ~14 lines of C++ code. The algorithm's `rwnd` (AlgoRwnd) is initialized to a very high value (~2 MB). In each round, if the current queue size is greater than 250 KB, we reduce the AlgoRwnd by half. If the queue size falls below 25 KB, we increase it by 1500 B. The AlgoRwnd is updated in the dataplane at the end of each round. Every time the dataplane sees an ACK packet, it scales the AlgoRwnd by that flow's RTT and WS, and writes it to the packet's TCP window field. Figure 2 shows that this simple scheme is sufficient to alleviate RTT unfairness between two BBR flows.

3 DISCUSSION AND FUTURE WORK

Flowtamer runs on an individual switch without requiring any coordination with other switches and thus can be deployed incrementally. It is a work-in-progress and future work includes allowing the CCA to dynamically change the round duration, improving per-flow state management, estimating the number of concurrent flows, and handling SYN-flood attacks. Flows with asymmetric paths also pose a challenge to our methodology. Since our solution is `rwnd`-based, it also remains to be seen how our approach can be adapted for QUIC. When complete, we hope that Flowtamer would enable the efficient and scalable realization of future in-network CCAs such as RCS [3].

REFERENCES

[1] R. Al-Saadi, G. Armitage, J. But, and P. Branch. A survey of delay-based and hybrid tcp congestion control algorithms. *IEEE Communications Surveys & Tutorials*, pages 3609–3638, 2019.

[2] A. G. Alcoz, A. Dietmüller, and L. Vanbever. SP-PIFO: Approximating push-in first-out behaviors using strict-priority queues. In *Proceedings of NSDI*, 2020.

[3] L. Brown, G. Ananthanarayanan, E. Katz-Bassett, A. Krishnamurthy, S. Ratnasamy, M. Schapira, and S. Shenker. On the future of congestion control for the public Internet. In *Proceedings of HotNets*, 2020.

[4] CAIDA. Anonymized Internet Traces 2019. https://catalog.caida.org/dataset/passive_2019_pcap.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time. *Queue*, pages 20–53, 2016.

[6] N. Cardwell, Y. Cheng, S. H. Yeganeh, P. Jha, Y. Seung, K. Yang, I. Swett, V. Vasiliev, B. Wu, L. Hsiao, et al. Bbrv2: A model-based congestion control performance optimization. In *Proceedings of IETF 106th Meeting*, 2019.

[7] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, pages 397–413, 1993.

[8] M. Hock, R. Bless, and M. Zitterbart. Experimental evaluation of bbr congestion control. In *Proceedings of ICNP*, 2017.

[9] V. Jacobson and M. J. Karels. Congestion avoidance and control. In *Proceedings of SIGCOMM*, 1988.

[10] H. Jiang, Y. Wang, K. Lee, and I. Rhee. Tackling bufferbloat in 3G/4G networks. In *Proceedings of IMC*, 2012.

[11] Y.-T. Li, D. Leith, and R. N. Shorten. Experimental evaluation of tcp protocols for high-speed networks. *IEEE/ACM Transactions on networking*, pages 1109–1122, 2007.

[12] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *Proceedings of ICNP*, 2001.

[13] A. Mishra, L. Rastogi, R. Joshi, and B. Leong. Keeping an eye on congestion control in the wild with nebby. In *Proceedings of SIGCOMM*, 2024.

[14] S. Sengupta, H. Kim, and J. Rexford. Continuous in-network round-trip time monitoring. In *Proceedings of SIGCOMM*, 2022.

[15] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy. Approximating fair queueing on reconfigurable switches. In *Proceedings of NSDI*, 2018.

[16] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry. Beyond Jain's Fairness Index: Setting the bar for the deployment of congestion control algorithms. In *Proceedings of HotNets*, 2019.

[17] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry. Modeling bbr's interactions with loss-based congestion control. In *Proceedings of IMC*, 2019.

[18] L. Yu, J. Sonchack, and V. Liu. Cebinae: scalable in-network fairness augmentation. In *Proceedings of SIGCOMM*, 2022.