Making Congestion Control Algorithms Insensitive to Underlying Propagation Delays

Cyrus Illick
Columbia University, United States

Michael Roger
Columbia University, United States

Vishal Misra
Columbia University, United States

Dan Rubenstein
Columbia University, United States

Abstract

Underlying propagation delays, which significantly impact flow Round-Trip-Times (RTTs), fundamentally shape how congestion control algorithms (CCAs) allocate bandwidth among competing flows. Traditionally, lower RTTs were desirable, as a flow rate would decrease as RTT increased under conventional CCAs such as Reno and Cubic. More recently, algorithms like BBR reversed this behavior, favoring flows with larger RTTs. In modern heterogeneous networks, where competing flows may employ different CCAs and serve diverse application workloads, notions of fairness become ambiguous and RTT's role further complexifies addressing an already very challenging problem. We offer a simple but initially counter-intuitive remedy to address how all CCAs can remove this inherent sensitivity to propagation delay: have all flows "emulate" the same underlying propagation delay. Unsurprisingly, this idea is often met with a lot of resistance, since it is in many ways counter to many fundamental tenets of networking and congestion control. In this paper, we try to make the case that while the idea is not a silver bullet on its own, it can greatly simplify design and predictability across existing and future congestion control protocols going forward.

2012 ACM Subject Classification Networks → Transport protocols

Keywords and phrases Congestion Control, Propagation Delay

Digital Object Identifier 10.4230/OASIcs.NINeS.2026.27

Acknowledgements We want to thank anonymous reviewers and our shepherd Keith Winstein for valuable feedback that helped improve our paper.

1 Introduction

Internet congestion control algorithms (CCAs) prevent congestion collapse by having senders interpret feedback from receivers to gauge the status of the network over which their transmissions are sent, and adjust sending rates accordingly. TCP originally obtained its CCA based on the classical analysis of Chiu and Jain [15]. Van Jacobson and Mike Karels furthered the work by implementing the additive increase multiplicative decrease (AIMD) mechanism based on the work in 1988 [27], saving the Internet from congestion collapse. Although the original mechanism has undergone several refinements over the years, a flow's rate is impacted by both its round trip time (RTT) and loss probability, demonstrated empirically and analytically by several classical works [39, 46]. In particular, the dependence on RTT, known as "RTT unfairness", is perceived as "unfair" from the perspective of many well-formulated fairness measures, such as max-min fairness [29], which implicitly postulate that while a flow's rate must conform to fit within the constraining bottleneck, it should not be penalized with respect to other flows through the bottleneck simply for having a different

1st New Ideas in Networked Systems (NINeS 2026).
Editors: Katerina J. Argyraki and Aurojit Panda; Article No. 27; pp. 27:1–27:26
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

RTT. RTT is a dynamic aggregate of several delay components, but the main component that leads to RTT unfairness is the underlying *propagation delay* which, for a fixed path between sender and receiver, is defined as the minimum time required for a sender transmitting a data packet to receive an acknowledgment of its receipt. Propagation delay incorporates both the time the packet travels, as well as the (minimum) times the various routers and end-systems require to perform the necessary associated processing.

From an application-based perspective, while partitioning of bandwidth must be sensitive to the number of flows sharing a bottleneck, the manner in which this bandwidth is proportionally partitioned should really be a function of the relative applications' need and importance - measures that rarely depend on the underlying propagation delay. Assuming applications adopt CCAs that align with their needs, it's essential to understand how often those needs are met under varying propagation delays. Here, conventional measures of flow competition - such as friendliness, aggressiveness, and RTT-unfairness which compare the respective rates allocated to competing flows - can expose the fact that heterogeneous flows (with different CCAs, application workloads and propagation delays) do not share bandwidth equally but they do not quantify how predictable a sessions performance is with respect to propagation delay, or what we call the *sensitivity* to propagation delay: how much a flows allocation might change were the specific propagation delays of competing flows simply different. Our goal in this paper is to focus attention on *predictability* with respect to propagation delay and not fall into an argument over the definition or importance of Internet fairness.

A variety of CCAs [7, 37, 16, 17, 11, 8] already exist that are insensitive to propagation delay, i.e., they allocate shares at a bottleneck across flows irrespective of the underlying flows' propagation delays by introducing or modifying specific mechanisms and specialized parameters of the CCA. However, their success requires that they operate in what we call *CCA-homogeneous* environments, i.e. each flow uses the same CCA such that this CCA competes only with itself. It is well known that this insensitivity to propagation delay fails between flows using the same specially-engineered CCAs when competing in a *CCA-heterogeneous* bottleneck environment that contains flows that utilize alternate CCAs [7, 19, 32, 37, 22]. Presently, the allocations at a given bottleneck for the most prevalent CCAs utilized by popular applications, such as the immensely popular BBR [12, 14] and Linux Standard Cubic [23], remain affected by propagation delay, even in CCA-homogeneous environments [50, 26, 28, 47, 33, 6, 51, 4, 21].

In this paper, we propose an alternate approach to reduce the influence of propagation delay on CCA competition - suitable for Internet heterogeneity and application priorities. Rather than altering or introducing new designs to the utilized CCA, we propose altering the perceived propagation delay the CCA experiences by introducing *artificial delay* added to the acknowledgments along the network path. This unconventional, likely controversial approach can reduce the dependence on propagation delay for long-lived flows and alleviate the burden of designing complex mechanisms in the CCA's control loop. Given the controversial nature of the proposed idea, we begin in §2 addressing what we imagine would be a reader's knee-jerk reaction to why this is not a good idea, and address those concerns up-front in a qualitative manner that provides a guide for the subsequent more technical discussion in the remainder of the paper. §3 explains the recent reconsideration of how objectives of CCAs really go beyond simple notions of fairness, and how one might view variations in propagation delay as a significant impediment to these redefined objectives as well. We attempt to quantify the impact on allocations that varying propagation delay might have by constructing a novel δ-sensitivity metric in §4. We are then in the position to motivate more formally an argument for artificial delay in §5, and explore its preliminary impact on the sensitivity of BBR in §6 and its suitability for more complex heterogeneous and practical scenarios in §7. We continue discussion in §8, related works in §9, and §10 concludes the paper and proposes future work.

The main contributions of this paper are:

- A framing of the effects of propagation delay, including an attempt at a well formalized metric that measures CCA *sensitivity to delay*, beyond a mere notion of fairness.
- An argument for resolving the sensitivity of existing CCAs to propagation delay by injecting "artificial" delay.
- Our early experiences implementing the artificial delay mechanism and experiments demonstrating how CCA sensitivity to delay is thereby reduced.

2 Q&A

Our proposed idea is quite simple to state. Our goal is to take propagation delay out of the congestion control "equation" by delaying the response to ACKs at the sender. Conceptually, if a flow's propagation delay is d, and we require flows to emulate a delay of $\lambda > d$, what a sender simply needs to do in theory is, upon receiving an ACK at time t, wait until time $t + \lambda - d$ to respond to that ACK. Are there not drawbacks when flows artificially delay their response to valuable congestion feedback? In this section, we provide a qualitative Q&A that addresses these concerns up front, divided into three basic areas of concern: Performance, Network-wide adoption, and Implementation. Where possible, we connect answers to later sections where we provide formal metrics and experimental evidence.

2.1 Performance Concerns

Question 1: *What is meant in this paper by predictable performance?*
Answer: A flow's rate is clearly a function of the attributes of the bottleneck(s) as well as the number and attributes of the flows with which it competes for that bottleneck bandwidth. For the purposes of this paper, we informally define predictable to mean that with knowledge of these bottleneck and flow attributes, one can determine the throughput that each flow will obtain. Current reality is that these rates also depend on the respective flows' propagation delays: predictable means that it should not. We formalize this notion further in §4.1.
Question 2: *Isn't delaying responses to feedback just a bad idea?*
Answer: Not necessarily. Inflating RTTs inherently decreases the speed at which CCAs are able to receive and adapt to feedback. However, performance of an individual flow is often more reliant on how its own propagation delay compares to competing flows rather than the propagation delay itself. Later in §4 and §6, we show that once all flows are equalized to the same delay, their allocations become much more predictable and less variable, even though RTTs are inflated. In practice, consistency improves more than responsiveness suffers.
Question 3: *Some CCAs are delay based. How do they get their feedback signal?*
Answer: Our proposal is not to normalize RTT which is defined to include components of delay in addition to propagation delay (e.g. queuing). By normalizing flows' propagation delays, RTT delay experienced by ACKs still capture jitter and congestive delay i.e., the sender assesses the underlying propagation delay and then pads ACKs by a constant amount. As shown in §6, experiments confirm that CCAs relying on RTT variations continue to operate correctly under added artificial delay.
Question 4: *What about real-time traffic that is delay sensitive? Doesn't adding delay decrease responsiveness for applications like gaming or video-conferencing that depend upon low latency delivery?*

Answer: Note that when a sender has a data packet to send, we are not adding any delay between its departure from the sender and its arrival at the receiver. On a per-packet basis, delaying an ACK may affect the aggressiveness of startup but not payload latency. Experiments in §5.1 confirm this: adding 60ms to ACKs increases RTT but leaves application-level one-way data latency unchanged.

Question 5: *OK, so what about startup / transient effects (sender would react slower)?*

Answer: The behavior is equivalent to a flow with propagation delay λ, and most CCAs are designed to operate robustly across a wide delay range. While convergence during startup may be slower, this is already impacted by asymmetric delays in today's Internet. Moreover, results in §6 show that even with such transient effects, equalization substantially reduces variability in final allocations.

Question 6: *What about short-lived flows? Can delay equalization benefit a flow that never exits slow-start?*

Answer: Most flows are short-lived flows [10]. In this paper we focus on long-lived flows which adapt to congestion signals and share bandwidth. We suggest that this idea should not be used on short-lived flows and leave it to future work to study if some method of equalizing delay could benefit performance within slow-start.

Question 7: *What about flows entering/exiting the bottleneck? Won't the added delay interfere with stabilization?*

Answer: The majority of Internet traffic volume is long-lived video streaming [3]. Slower convergence may matter, but experiments in §7.2 show that even naïve artificial delay (e.g., adding a fixed constant) reduces variability across flows. If needed, a two-tier system could allow "rush ACK delivery" during unstable periods, then revert to delayed ACKs once equilibrium is reached.

Question 8: *How does one quantify the impact that this propagation delay normalization has on a heterogeneous suite of CCAs sharing a bottleneck link?*

Answer: We think a new measure is needed to capture this effect, as existing measures of fairness or application performance *with respect to varied propagation delays* has not received sufficient attention as discussed in §3. This motivates our formal definition of a new metric in §4, where we quantify sensitivity to propagation delay directly, independent of notions of fairness - which allows for complete evaluations in §6 and §7.

Question 9: *So you've gotten rid of the propagation delay issue. But you still can't really guarantee any "application-needed" performance? Why bother?*

Answer: We agree that heterogeneous propagation delay is only one performance concern. But delay sensitivity is a uniquely extraneous factor—not dictated by end-host choices—that introduces unpredictability. Neutralizing it makes the performance landscape more stable, giving applications and CCAs a cleaner foundation to optimize around their own objectives.

2.2 Network-Wide Adoption

Question 10: *What about network requirements? How would routers need to be reprogrammed?*

Answer: Our proposal is implemented at the data source (sender) such that no router changes are required. The sender does need to maintain some additional state about the ACKs it is holding: basically per ACK the sequence number and release time. Note that this information is quite minimal and there are several ways to compress this information efficiently (for starters, store the delta between consecutive sequence numbers and consecutive release times).

Question 11: *Even if the idea could improve predictability of performance, how do you get CCAs to adopt this approach?*
Answer: In the short term (as well as long term if needed), we show in §5.3 that the delaying mechanism could be implemented externally from the CCA via a *proxy* at the sender that captures ACKs and holds them for the appropriate time intervals prior to releasing them to the CCA. Furthermore, for some protocols like BBR whose throughput has been shown to lessen with lessening propagation delays, there is an impetus to mitigating this phenomenon and our experiments (§6) show clear gains in both throughput balance and stability across flows.
Question 12: *What will happen to my favorite CCA?*
Answer: Each CCA is impacted differently by propagation delay and performance in heterogeneous competition will depend on what λ propagation delay is inflated to. Once a target λ is chosen, CCAs converge toward allocations with much less variability due to delay. This means performance becomes more consistent across CCAs, rather than less. We discuss how an agreed upon λ can benefit the design of new CCAs in §8.
Question 13: *So how does one go about choosing a good value for λ?*
Answer: λ has to be large enough to allow for the vast majority of competing flows to participate. Flows in different classes of networks (datacenter, WAN, satellite) may have different viable ranges of propagation delay, so perhaps it make sense to have "delay classes" where flows compete within their class and have network support where different delay classes are allocated bandwidth. Alternatively, if some flows exist above the point of equalization it is likely that reducing the magnitude of propagation delay asymmetry will reduce delay sensitivity and improve consistency (discussed further in §7.2 and §8)
Question 14: *Should this idea be adopted everywhere right away? Are there instances where this idea will have a negative impact?*
Answer: In this paper, we argue for the viability of this idea and provide initial results that suggest its potential benefit. Future work is required for more robust evaluations across diverse application workloads and network settings. As noted in question 6, there are some instances, such as short-lived flows, where this idea is likely to cause more harm than good. We propose guidelines for when this idea should and should not be applied in §5.2 and discuss further limitations in §8.

2.3 Implementation Concerns

Question 15: *How would you suggest actually implementing this thing?*
Answer: We propose a simple proxy solution in §5.3, which we evaluate in both emulated and real-world settings in §6. In both cases, variability due to propagation delay drops dramatically while utilization remains high, showing that the implementation is practical.
Question 16: *What about a server that is the sender for many (thousands / millions) of flows? Can't state requirements get overwhelming?*
Answer: As noted in Question 10, there are ways to compress per-flow state to only a few bits per ACK that is actively being delayed. Compared to existing per-flow state that the server must maintain, such overhead is likely acceptable.

3 CCA Goals: Beyond Fairness

To reduce the impact of propagation delay in heterogeneous competition, we must first address a different challenge introduced by Internet heterogeneity: how competing alternate CCAs *should* compete.

A recent line of work motivating this paper involves an introspective reconsideration of what the underlying goals of CCAs should be when deployed in networks supporting heterogeneous applications with differing requirements and measures of success. Here, simply ensuring an equal division among flows constrained by the same bottleneck may not be in the best interest of the applications these flows are supporting [55, 49, 58, 9, 18]. Moreover, *the very notion of "fairness" in Internet bandwidth allocation is not clearly defined or enforced in practice.* Commercial arrangements between ISPs and content providers, or enterprise-specific peering agreements, frequently depart from broad fairness doctrines such as max-min fairness. Instead, they reflect *business relationships* and *performance objectives* that do not necessarily align with a universal fairness principle [9].

Regardless of how one might define an ideal division of bandwidth among competing heterogeneous flows, any sensitivity to propagation delay only complicates the ability to reach this point consistently. An exception arises in formulations where the desired allocation itself is explicitly defined as a function of propagation delay. For example, some arguments regard it as "acceptable" for a short-RTT flow to receive proportionally greater bandwidth than an otherwise identical long-RTT flow [23]. In such views, variation in allocation with propagation delay reflects design intent rather than a failure of competition. Our focus, however, is on the broad class of objectives—spanning many practical notions of fairness and application priorities—for which the intended operating point does not depend on propagation delay.

3.1 Propagation Delay Impacts Application Performance

Recent work [48] provides an in-depth evaluation of how competing popular applications share Internet bandwidth when congested at a common bottleneck, demonstrating that (1) applications do not share bandwidth equally and (2) practical evaluations *must* consider the full network stack as opposed to settings which isolate the underlying CCA mechanisms in controlled environments. *Notably, this prior work equalizes the competing flows' propagation delays at the bottleneck* to focus on how differences in rate achieved between applications was a function of the differences in their workload and chosen mechanisms (CCA, application layer protocols) but not from differences in respective propagation delays.

To illustrate the impact of propagation delay in practical settings, let us run 5 actual Internet experiments in which a YouTube (YT) session (utilizing a variant of BBRv3[14])[1] and a OneDrive (1D) session (utilizing Cubic[23]) compete across a 50 Mbps capacity bottleneck with the same moderate sized buffer in all experiments (equivalent to 2BDP assuming 15 ms delay), where the only change across the experiments are the propagation delays experienced by the respective sessions. For 1D, we download the same 5 GB file in each trial. For YT, we play the same video in each trial, selecting the 4k video quality option in YT, preventing the player from down-scaling due to adaptive bit rate logic (we found that YT was still impacted by propagation delay when adaptive bit rate logic is enabled).

To manipulate propagation delay, we introduce a fixed artificial delay at the receiver side - inflating the natural end-to-end propagation delay, allowing us to control delay asymmetrically across competing flows. In this case, we utilize artificial delay on the receiver end to emulate what would occur had the natural propagation delay been larger. We discuss the use of artificial delay to reduce such sensitivity (when applied at the sending side) and the validity of artificial delay in §5.

Each trial consists of an averaging over 4 repetitions, where each repetition is run for 150

[1] CCA utilized by YouTube reported in [13].

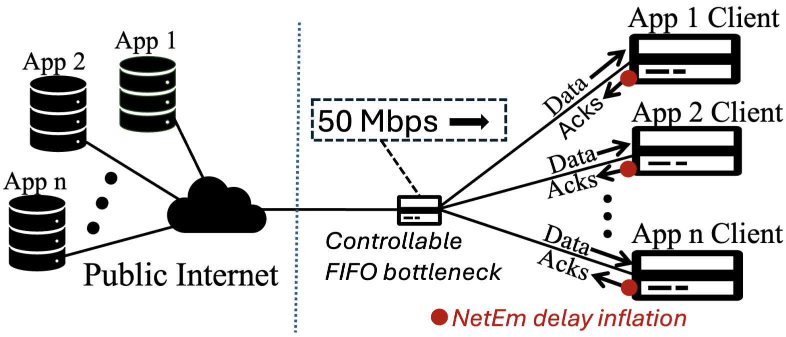


Figure 1 Application Experiment Setup: Competing flows sourced from application services (App 1, ..., App n) transmitting data to receivers (clients) behind a 50Mbps capacity bottleneck with moderate sized buffer.

Trial #	Delay (ms)		Rate (Mbps)		YT performance
	1D	YT	1D	YT	Stall Time (s)
1	7	7	33.0	15.8	35.4
2	12	12	30.7	17.9	7.7
3	12	37	30.6	18.3	5.9
4	37	12	35.4	12.4	72.2
5	37	37	25.7	18.2	6.8

Table 1 Performance of OneDrive (Cubic) vs YouTube (BBRv3) with varying propagation delays. **YT's total stall time varies over 12× across trials despite fixed workload and capacity, driven only by changes in delay.**

seconds - neither YT or 1D finish in this time. The topology of our testbed is included in Figure 1 and additional details on our full application experimental methodology is deferred to §7.3.

Table 1 presents, row-by-row, results for the five distinct trials, each differing by the propagation delays experienced by the applications, as described in columns 2 and 3 for 1D and YT respectively. Performance results include their respective average rates (columns 4 and 5), and an application-level measure of performance for the YT session in terms of total time stalled - including the initial startup delay and all subsequent rebuffering durations during playback (column 6), measured using Selenium web driver [1].

The trials reaffirm that application performance is indeed very sensitive to the propagation delays experienced by the competing flows. As shown in Table 1, Trials 3 and 4 reveal a 32% decrease in rate (from 18.3 to 12.4 Mbps) and a 12× increase in total stall time (from 5.9s to 72.2s)[2] for the YT session caused solely by a change in propagation delay. The workload and bottleneck are unchanged, but reversing the delay asymmetry between flows causes YouTube's performance to collapse. Performance across these trials depends not only on the CCAs utilized by the competing applications, but also on their respective workloads, application-layer mechanisms, and any unobservable factors inherent to real-world experiments. However, the qualitative trends observed by comparing performance between these trials mirror those seen in our controlled, CCA-isolated experiments involving a bulk-transfer Cubic flow competing against a bulk-transfer BBR flow under the same bottleneck

[2] Stall time ranges from 3.5s to 9.1s between repetitions of Trial 3 and from 65s to 86s between repetitions of Trial 4.

and propagation-delay conditions. In particular, we observe the the trend of the BBR flow between these settings: when both flows have identical propagation delays, the rate allocated to BBR increases as the common delay increases; when the Cubic flow's delay is held fixed and the BBR flow's delay is varied, BBR's rate increases with its own delay; and when the BBR flow's delay is held fixed while the Cubic flow's delay is varied, BBR's rate increases as the Cubic flow's delay decreases.

Note a single configuration (i.e., a single trial) is not enough to evaluate how a 1D and YT flow would share a given bottleneck. Before introducing our proposed metric (δ-sensitivity) in §4 and the potential of artificial delay §5, we distill a few key observations from these experiments: (1) Propagation delay significantly impacts performance across heterogeneous applications - even when all other underlying network conditions and workloads remain constant. (2) Properties of RTT-unfairness in homogeneous CCA competition do not always extend to heterogeneous settings; for example, between trials 2 and 4, the rate of 1D (utilizing Cubic) increases when its delay increases despite the dynamics of Cubic flows in homogeneous competition where more bandwidth is conceded to the shorter delayed flow [23]. (3) Equalizing propagation delay between competing applications does not isolate its effect in heterogeneous settings (for instance compare trials 1,2 and 5 where propagation delays across both flows are identical); application mechanisms and CCAs interact with delay in non-trivial ways, and the choice of equalization point itself influences outcomes. (4) Traditional metrics of RTT-unfairness or rate allocation that measure over a single instance (where propagation delays are not varied) fail to capture the variability in quality-of-experience observed in practice.

These observations motivate the need for a new, generalizable approach to quantify and address how sensitive a flow's performance is to propagation delay, independent of assumptions about fairness or homogeneity.

4 Delay-sensitivity (not Fairness!)

A challenge with CCA allocations being dependent on propagation delay is an issue that is beyond fairness. For a given configuration of flows, each using its preferred CCA, and a given bottleneck link, can we determine the *allocation*, i.e., the respective throughputs of these flows? If the answer were simply "yes", we could then utilize some measure of satisfaction to quantitatively gauge the acceptability of this allocation. Unfortunately, the allocation is often also a function of the respective propagation delays of these flows: the same set of flows using the same CCAs passing through the same bottleneck but with different propagation delays can often result in a very different allocation. Our goal here is not to assess the fairness of the set of allocations as a function of the propagation delays, but instead *to understand how much the variability of these propagation delays impacts the possible allocations.* In other words, we want to understand the *predictability* of performance with respect to network propagation delays.

In this section, we define δ-sensitivity to generalize this notion of sensitivity and apply to network settings involving any number of flows with any level of heterogeneity between them, and for performance metrics beyond average throughput[3].

[3] Metrics beyond average throughput discussed in §4.1.3

4.1 δ-sensitivity

Define $\mathcal{I} = \langle\{f_i\}, B\rangle$ to consist of n flows $f_1, ..., f_n$ passing through a bottleneck B. Each flow $f_i = (a_i, w_i)$ utilizes congestion control algorithm $a_i \in A$ (where A is the set of heterogeneous CCAs) and carries workload $w_i \in W$ (where W is the set of possible workloads)[4]. The bottleneck $B = (C, \ell, Q)$ is defined in terms of capacity C, buffer limit ℓ and queuing discipline Q (e.g FIFO). Note that $\mathcal{I}$ fixes a bottleneck, a set of flows through the bottleneck, the utilized CCAs and workloads of each flow *without* yet specifying the propagation delays assigned to these flows.

We define a *propagation delay instance*, $D = (d_1, d_2, ..., d_n)$ to be a specific assignment of propagation delays to these n flows. Armed with both $\mathcal{I}$ and D, one can model, simulate, or execute the system and determine the resulting rates $r_i = R_i(\mathcal{I}, D)$ of each of these n flows when operating in a network setting defined by $\mathcal{I}$ and D.

δ-sensitivity does not simply consider a single propagation delay instance, but instead is defined over a set $\mathcal{D} = \{D\}$ of permissible (to the current setting) propagation delay instances. Each distinct $D \in \mathcal{D}$ will yield its own set of resulting rates, such that $R_i(\mathcal{I}, D_1)$ need not equal $R_i(\mathcal{I}, D_2)$ for two distinct $D_1, D_2 \in \mathcal{D}$. The δ-sensitivity of flow i for the network setting described by $\mathcal{I}$ over the set of propagation delay instances $\mathcal{D}$ is,

$$\delta_{\mathcal{I},\mathcal{D}}(i) = \log(\max_{D\in\mathcal{D}} R_i(\mathcal{I}, D)) - \log(\min_{D\in\mathcal{D}} R_i(\mathcal{I}, D))$$

In other words, take this network of flows, sample using different propagation delays drawn from this set $\mathcal{D}$ of permissible propagation delay assignments, and for each flow f_i compute the log-ratio[5] between the largest and smallest allocations assigned to it across all $D \in \mathcal{D}$. $\delta_{\mathcal{I},\mathcal{D}}(i) = 0$ implies that for the network setting $\mathcal{I}$, the rate r_i achieved by flow i is unaffected by changes in propagation delay assignments in the set of propagation delay instances $\mathcal{D}$ considered.

δ-sensitivity is intended to apply on a case-by-case basis for a) a given set of flows assigned respectively to specific CCAs with specific workloads through a specific bottleneck link and b) the permissible propagation delay conditions that can be undertaken by those flows. Note that (a) is typical for most evaluations of congestion control protocols whereas (b) allows for variation and comparison across these varied propagation delays, where traditional evaluations often fix flows to specific propagation delays.

The reason for allowing the scenario to define the permissible set of delays that form $\mathcal{D}$ is because different networking environments may have different limits on the ranges and variances of permissible propagation delays. For instance, a data center environment might comprise flows which are guaranteed to have similarly low propagation delays with minimal variance [24]; a traditional wide area networking environment might consider a fairly broad range of propagation delays (up to on the order of a few hundred milliseconds) [59]; and space networks [5] might require consideration of delays that are even larger. Furthermore, there are scenarios in which the delays of flows from a content delivery network CDN are consistently small but can interact with competing flows which delays vary greatly.

For concreteness, we give some plausible ways to define the $\mathcal{D}$, e.g.,

- **Uniform delay range** $\mathcal{D} = \{(d_1, d_2, ..., d_n) : 10 \le d_i \le 100\}$: looking only at scenarios where flow propagation delay ranges between 10 and 100.

[4] The choice to define a flow based on both congestion control algorithm and workload inspired by [55].

[5] We find the log-ratio more semantically meaningful than a standard ratio because a flow being “insensitive” to delay maps elegantly to $\delta(i) = 0$.

- **Single-flow variation** $\mathcal{D} = \{(d_1, 10, ..., 10) : 10 \leq d_1 \leq 100\}$: looking only at scenarios where all d_i are fixed at 10 except d_1 which varies between 10 and 100.
- **Symmetric variation** $\mathcal{D} = \{(d_1, d_2, ..., d_n) : 10 \leq d_0 \leq 50, d_i = d_0\}$: looking only at scenarios where d_0 ranges from 10 to 50 and all d_i equal d_0.

Note that $\mathcal{D}$ can be defined in such a way that there may be an (exponentially or even infinitely) large set of propagation delay assignments such that it would not be feasible to exhaustively run all configurations to determine the true max and min. We note that experimentally sampling over a subset of configurations provides a lower bound to the delay sensitivity. The observed minimum m_0 would upper bound the true minimum ($m_0 \geq m$), and the observed maximum M_0 would lower bound the true maximum ($M_0 \leq M$), such that $\log(M_0/m_0) \leq \log(M/m)$.

4.1.1 Walking through δ-sensitivity: A simple 2 flow example

Consider the network setting ($\mathcal{I}$) between YT and 1D from the previous section where the two flows f_1 (YT), f_2 (1D) share a 50 Mbps bottleneck. Suppose we allow just two possible *propagation delay instances*:

$$D_1 = (12\,\text{ms},\ 12\,\text{ms}), \quad D_2 = (12\,\text{ms},\ 37\,\text{ms}).$$

$R_1(\mathcal{I}, D_1) = 17.9\,\text{Mbps}$ and $R_1(\mathcal{I}, D_2) = 12.4\,\text{Mbps}$, so $\max R_1 = 17.9, \min R_1 = 12.4$

$$\Rightarrow \ \delta_{\mathcal{I},\mathcal{D}}(1) = \log_2\Big(\frac{17.9}{12.4}\Big) \approx 0.53.$$

Hence, a δ-value of 0.53 means that flow 1's (in this case YT's) throughput can vary by 1.45 times. Larger δ indicates *greater sensitivity* to the range of delays in $\mathcal{D}$. In this case the variation in rate achieved by f_1 is due exclusively to variation in f_2's (1D's) propagation delay.

4.1.2 Interpreting High δ-sensitivity

High δ-sensitivity indicates that a flow's performance can vary substantially across plausible propagation delay configurations, even when all other factors - competing flows, utilized protocols, workloads, and bottleneck properties - remain fixed.

A flow's performance must depend on the number of competing flow workloads and the constraining bottleneck these flows share, and will depend on the choices application services make inline with their respective application needs.

But propagation delays are neither chosen by the end hosts or indicative of their needs. When δ is high, this lack of control translates into unpredictable performance. By contrast, low δ implies more robust, predictable behavior - a desirable trait for any networked applications. And $\delta = 0$ implies that the variation of propagation delays will have no effect at all on the allocation.

4.1.3 Multi-metric δ-sensitivity definition

Many applications prioritize QoE over raw throughput, and δ-sensitivity must adapt accordingly. Here we extend our definition of δ-sensitivity to allow for metrics of performance beyond throughput after convergence. Let $M \in \mathcal{M}$ be an arbitrary performance metric from a set of possible metrics $\mathcal{M}$ (e.g. QoE metrics such as Flow Completion Time, Buffering time, etc) such that $m_i = M_i(\mathcal{I}, D)$ defines the performance of flow i on metric M in network setting $\mathcal{I}$ with propagation delay assignment D. Thus, δ-sensitivity of a flow i on metric M is,

$$\delta^{M}_{\mathcal{I},\mathcal{D}}(i) = \log(\max_{D \in \mathcal{D}} M_i(\mathcal{I}, D)) - \log(\min_{D \in \mathcal{D}} M_i(\mathcal{I}, D))$$

For example, if we define $\mathcal{D}$ as the five propagation delay instances shown in Table 1, YT's Re-buffering δ-sensitivity is $\delta^{\text{stall}}_{\mathcal{D},\mathcal{I}}(\text{YT}) = 3.63$. Suppose, instead, that the range of practical propagation delays is more appropriately represented by restricting this range $\mathcal{D}$ to only include delays that are $\leq$ 12ms - i.e. $\mathcal{D} = \{(7,7),(12,12)\}$ - then $\delta^{\text{stall}}_{\mathcal{D},\mathcal{I}}(\text{YT}) = 2.2$ where stall time varies between 7.7 and 35.4 seconds across these two configurations. This generalization allows δ to track delay-induced variation in any performance metric that matters to applications, not just throughput.

5 An Argument for Artificial Delay

With a formal definition of δ-sensitivity established, centered around the delay range $\mathcal{D}$, we now consider the use of *artificial delay* as a mechanism to actively modify this range. The primary goal of introducing artificial delay is to *reduce variability* in the propagation delays a flow experiences. By narrowing the effective range of delays, we aim to lower a flow's sensitivity to RTT differences, thereby reducing performance variability.

In this section, we introduce a simple end-system-only mechanism that can be deployed by sending end-hosts that will improve δ-sensitivity, regardless of the CCAs in use. The idea is simple: delay processing of acknowledgments, thereby increasing the time from when a data packet is transmitted to the time its acknowledgment is received and thus gaining control over the propagation delay experienced by the utilized CCA. We will show that having senders delaying the processing of acknowledgments will reduce δ-sensitivity, thereby making a set of long-lived flows' shares of the bottleneck more consistent and predictable. If this mechanism is standardized across senders, δ-sensitivity can converge to 0. Delaying the processing of acknowledgments runs contrary to the dogma that exists for congestion control protocols: instinctively we want to provide feedback as quickly as possible. It is worth noting up front that there is no added delay to data packets going from sender to receiver (i.e., the mechanism can apply to real-time traffic as well, as we demonstrate in 5.1).

Figure 2 shows a simple model of how different delay contributors to total network RTT impact performance[6]. Data packets are sent from the sending end-host to the receiving end-host along the data path (solid blue line) data packets are delayed by network jitter, forward propagation delay $d^{\rightarrow}$ and congestion. Delaying data packets at the sender's egress or receiver's ingress (adding artificial delay on the data path) will obviously increase data-latency for the receiving application and thus negatively impact performance. But by delaying packet acknowledgments (adding delay on the packet acknowledgment path), the total RTT can be inflated without implicitly hurting the latency of actual data transfer. In this paper, we focus on sender added artificial delay λ^{add} as an additional tool available to sending host to work in conjunction with a CCA.

5.1 Viability of Artificial Delays

Figure 3 shows experimental evidence of our intuitive claim that added delay on packet acknowledgments will not implicitly hurt application data latency (i.e., the ability to perform

[6] The acknowledgment path (the dashed line in figure 2) can also experience congestive and jitter delays. We do not include this in the figure because it does not impact the use of artificial added delay.

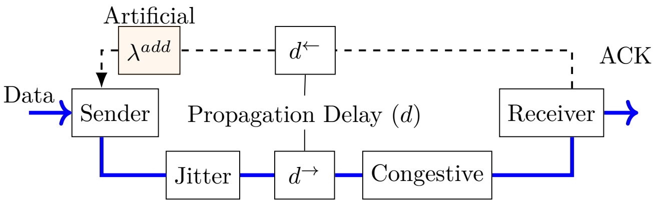


Figure 2 RTT Delay Contributors. Data latency is accrued along the solid blue path unaffected by added artificial delays on the dashed acknowledgment path.

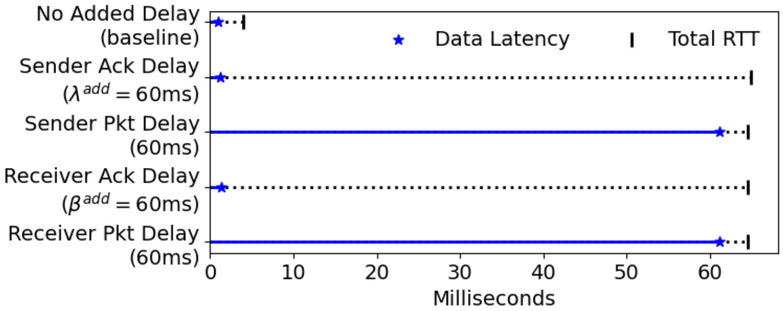


Figure 3 60 milliseconds of artificial delay added to the processing of acknowledgments (ACKs) increases RTT with negligible impact on data-latency.

timely data delivery from sender to receiver). This experiment is setup with a socket application that sends a packet every 1ms containing the timestamp of exiting the sending host's application layer and data is taken upon the receiver processing the packet at the application level. The solid blue line shows the duration of time (in milliseconds) until the receiving application processes the data packet and the dotted line is the duration until the acknowledgment for the packet reaches the sender (completing the total network RTT).

These experiments are intended to represent the perspective of an application that values real-time data transfer. The two hosts are placed on different floors of the same building with an average 1ms delay from sender to receiver and 3ms total RTT. There are four places where delay can be added: sender ingress (ACK delay λ^{add}), sender egress (packet delay), receiver ingress (packet delay) and receiver egress (ACK delay β^{add}). The results of adding 60ms of artificial delay at these four locations support our claim that compared to the baseline of no added delay, delaying acknowledgments at either the sender (λ^{add}) or receiver (β^{add}) inflates the perceived RTT without hurting sender to receiver latency. Delaying data packets has the obvious result in delaying sender to receiver latency.

These experiments are run over a network path with a capacity that exceeds the rate required by the application with minimal competition from competing streams. Because of this, there is negligible difference between running this application with Cubic or BBR. This shows that in ideal un-congested network scenarios, the addition of 60ms artificial delay does not degrade application performance even when the natural propagation delay is small (only 3ms) and artificial delay is large (60ms). When there is congestion, the choice of CCA has major performance implications and the use of intentional artificial delay becomes relevant.

5.2 Guidelines for Adding Delay

The purpose of introducing artificial delay is to remove the impact of propagation delay on how competing flows interoperate. This method is most suitable when the introduced artificial delay does not meaningfully degrade an *uncontested* flow's performance. Using δ-sensitivity, we can identify such cases: if a flow shows low δ-sensitivity over a delay range $\mathcal{D}$ in an uncontested network setting, then artificially inflated delays that remain within $\mathcal{D}$ has little impact when the flow is operating alone and is thus a suitable tool for reducing δ-sensitivity when flows compete. For example, 50ms of artificial delay can have negligible impact on file download time when a flow is bottlenecked alone but significant impact when the capacity of the bottleneck is shared with competing traffic. Conversely, 50 ms of artificial delay could have negative impact on short-lived flow such as loading a web-page and negligible impact on reducing δ-sensitivity in settings where this flow competes.

5.3 Removing δ-sensitivity with λ-proxy

Requiring existing CCAs to immediately incorporate artificial delay can be non-trivial. As an alternative (perhaps temporary, perhaps permanent) to induce artificial delay without changing the underlying structure of the network or the kernel's implementation of TCP, we propose a simple method to "trick" the CCA into thinking there are λ milliseconds of propagation delay by having the sender, artificially delay its own processing of packet acknowledgments (use λ^{add}). When λ is larger than any naturally occurring propagation delay, receiver attempts to delay their ACKs will either be subsumed within λ, cause reduced δ-sensitivity, or can be easily identified as greatly exceeding λ (and hence cannot be naturally occurring). In order to achieve λms propagation delay, we use an estimate $\hat{d}$ of minimum propagation delay d and then adjust added proxy delay, λ^{add}, to maintain $\lambda = \lambda^{add} + \hat{d}$. This way, the total propagation delay estimate will be λ milliseconds irrespective of actual propagation delay.

Algorithm 1 λ-Proxy

```
Init: $\lambda^{add} = \lambda$,  $W^{RTT}$ = time window
while connected do
    $d_{tot} \leftarrow \min(RTT_t)$  $t \in W^{RTT}$
    $\hat{d} \leftarrow d_{tot} - \lambda^{add}$
    Update $\lambda^{add} = \lambda - \hat{d}$
    wait $W^{RTT}$
```

The proxy protocol (Algorithm 1) repeatedly checks the minimum RTTs[7] to maintain that it is λ ms, making adjustments if necessary. We apply added delay λ^{add} by placing a netem [25] queue at the sender's ingress port for the connection. This delays packet acknowledgments at the sender before entering the transport layer and being processed by the sender's CCA. In situations where all competing flows have enabled the λ-proxy mechanism, no flow has an RTT advantage and all RTT-defined aspects of the CCA's design and approach to congestion will be equalized between flows.

[7] This procedure is based on how BBR[12] estimates propagation delay. The measure of minrtt can be attained via Linux kernel socket statistics [2] independent of utilized CCA, although measurement errors may be more likely with buffer-filling CCAs (discussed in Section 7.2).

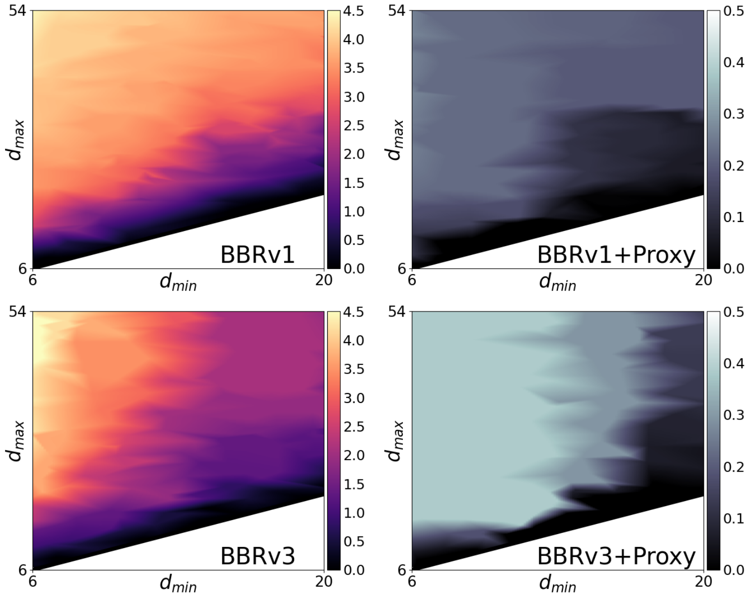


Figure 4 δ-sensitivity for $\mathcal{D} = \{(d_1, d_2) : d_{min} \leq d_i \leq d_{max}\}$. Shown for BBRv1 and v3 with and without proxy solution.

With an accurate estimate of propagation delay d, λ-proxy ensures zero δ-sensitivity for $\mathcal{D} = \{(d_1, ..., d_n) : d_i \leq \lambda\}$ if all competing flows enable the proxy. Compared to the difficulty of estimating queuing delay and network capacity, we find that propagation delay can be estimated consistently and accurately with increased accuracy for longer lived flows. We set λ at 100ms for all of our proxy experiments in §6.

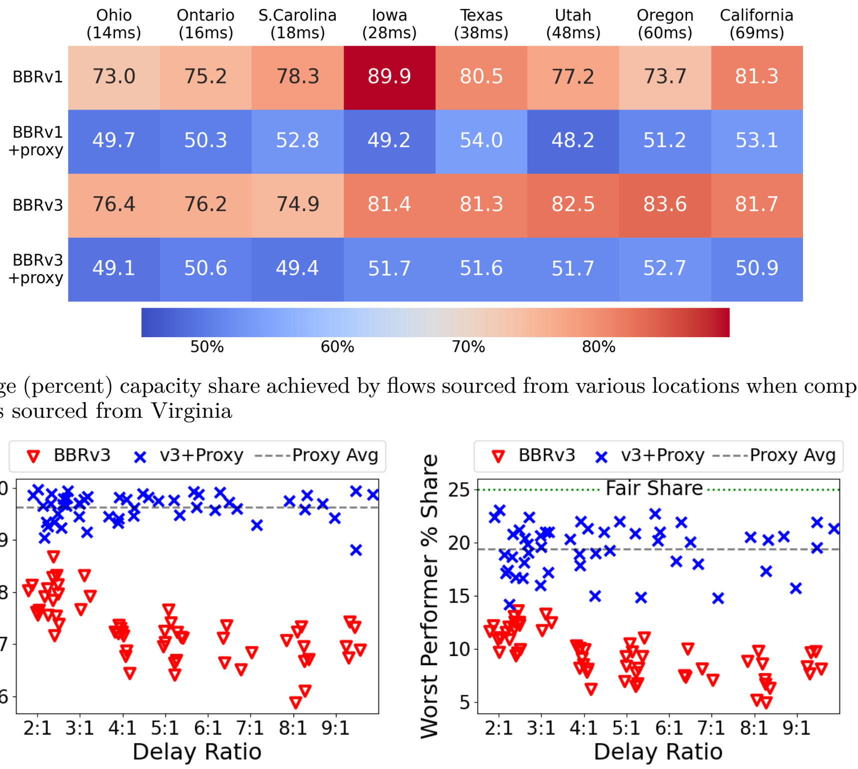
6 Case Study (BBR)

We begin with an evaluation (both emulated and real world) of introducing artificial delay to homogeneous BBR competition. Here, we chose BBR as it is one of the most prevalent CCAs used today and it has well documented properties of RTT-unfairness [21, 51, 44, 26]. Following this initial case study, we demonstrate the relevance extended to heterogeneous cases.

Emulated Validation. On an Ubuntu-Linux machine, using netem [25], network namespaces, and virtual NICs, we create emulated networks with varied propagation delay and bottleneck link capacity to test Linux implementations of CCAs including a kernel with BBRv3 enabled [53].

We use emulated experiments to measure δ-sensitivity across a range of $\mathcal{D}$ in controlled settings dictated by the CCA's behavior. Figure 4 shows δ-sensitivity for both versions of BBR with and without the application of λ-proxy. Each plot shows a visual representation of $\delta_{\mathcal{I},\mathcal{D}}$-sensitivity for the network setting $\mathcal{I}$ that consists of two identically provisioned flows utilizing the same CCA passing through a 200Mbps bottleneck and $\mathcal{D} = \{(d_1, d_2) : d_{min} \leq d_i \leq d_{max}\}$. d_{min} is varied on the x-axis and d_{max} is varied on the y-axis. The scale is reduced for the two proxy-enabled plots.

δ-sensitivity is computed from the results of 1600 experiments. Each experiment is between two flows with propagation delays ranging from 6 to 54ms. We measure performance after 60 seconds to observe how BBRv1 and BBRv3 converge to their stabilized rates



	Ohio (14ms)	Ontario (16ms)	S.Carolina (18ms)	Iowa (28ms)	Texas (38ms)	Utah (48ms)	Oregon (60ms)	California (69ms)
BBRv1	73.0	75.2	78.3	89.9	80.5	77.2	73.7	81.3
BBRv1 +proxy	49.7	50.3	52.8	49.2	54.0	48.2	51.2	53.1
BBRv3	76.4	76.2	74.9	81.4	81.3	82.5	83.6	81.7
BBRv3 +proxy	49.1	50.6	49.4	51.7	51.6	51.7	52.7	50.9

(a) Average (percent) capacity share achieved by flows sourced from various locations when competing with flows sourced from Virginia

(b) Other performance metrics for BBRv3 experiments. Jains Fairness Index and smallest % share of capacity allocated to one of the four competing flows.

Figure 5 Proxy effectiveness in noisy real world experiments.

irrespective to which flow starts first. The largest delay range considered in this experiment is $\mathcal{D} = \{(d_1, d_2) : 6 \leq d_i \leq 54\}$. With this $\mathcal{D}$ we find $\delta(i) = 4.58$ for BBRv1; $\delta(i) = 0.27$ for proxy-enabled BBRv1; $\delta(i) = 4.9$ for BBRv3; $\delta(i) = 0.38$ for proxy-enabled BBRv3 (note that in these experiments $\delta(i)$ is the same for $i = 1, 2$ because all delay instances (D_a, D_b) are repeated for (D_b, D_a)). We find that proxy-enabled BBRv1 competition is, on average, less δ-sensitive than proxy-enabled BBRv3 because of v3's larger estimation windows and probing strategies. In both cases, proxy-enabled two flow competition provides close to equal bandwidth allocations.

Real-world Validation. We extend our evaluation of proxy effectiveness to noisy real world network paths. Our experiments test data flow to a subnetwork of x86-processor PCs and ARM-processor Raspberry Pis. This subnetwork of receivers is connected to a Cisco switch located in New York City. We establish TCP connections from Google Cloud Provider (GCP) instances hosted in data-centers at different geographic locations. This provides our experiment with variable propagation delays ranging from 7ms to 75ms, observed using iproute2's *ss* log command [54]. We use bursts of UDP packets on a high capacity bottleneck (1Gbps) to provide background noise.

Our experiments consist of establishing 4 connections, two sourced from a data center located in Virginia (that has an average 7.46ms round trip propagation delay to our receivers in New York City) and two sourced from a data center located further away. We measure the average share of capacity held by the two longer delayed flows when they are sourced from different data centers. Figure 5a shows % of shared capacity achieved by the two longer delayed flows. The four rows show performance for BBRv1, BBRv1+proxy, BBRv3, BBRv3+proxy. The columns are labeled by the location the longer delayed flows are sourced from and includes the average round trip propagation time from that location. On average, for BBRv3, the two shorter delayed flows share 20.8% of bandwidth capacity compared to the two longer delayed flows which share 79.2%. With proxy-enabled, the two shorter delayed flows share 49% of capacity compared to the two longer delayed flows which share 51%.

These experiments represent a sampling of $D \in \mathcal{D}$, for the propagation delay range $\mathcal{D}$ that spans North America. The degree of sensitivity is impacted by buffer sizes, loss rate, and network jitter that occurs naturally and is specific to each network path. Without the addition of the proxy, the flows sourced from Virginia which transmit over a real world network path, have unpredictable rates that are are sensitive to the propagation delays of the competing flows that pass through the same bottleneck and are sourced from other locations. Proxy-enabled experiments are far more predictable, with flows sourced from Virginia achieving consistent rates across different experiments regardless of the natural propagation delay of the competing flows.

For comparison, Figure 5b plots commonly used metrics of performance: Jain's Fairness Index[30], and smallest % of shared capacity achieved among the four competing connections. In both cases, each experiment is positioned on the x-axis by the ratio between the largest and smallest of the four propagation delays and on the y-axis by the performance metric. Jain's Fairness degrades for BBRv3 as the ratio between the largest and smallest propagation delay increases. With proxy-enabled, our real world BBRv3 experiments have an average 0.95 Jain's Fairness. We plot the % of shared capacity achieved by the worst of the four performers to show that proxy-enabled BBRv3 is beneficial to ensuring that all flows have sufficient rate allocation. Compared to the 'fair' 25% capacity share, the worst performer in proxy-enabled BBRv3 achieves an average 19.3% of shared capacity compared to 9.4% without proxy-enabled.

Utilization Validation We confirm that our proxy-enabled experiments significantly reduce δ-sensitivity without degrading BBR's property of high network utilization. In real-world experiments we find that proxy addition marginally increases total throughput. We conjecture that this increase is due to larger RTTs making flows less affected by real-world network jitter. Utilization is unaffected by proxy addition in long term data transfer in our emulated tests.

7 Internet Suitability

Having shown that the proxy is able to reduce δ-sensitivity in competition among competing BBR flows, we now move to consideration of heterogeneous cases. In this section, after demonstrating the the difficulty of reducing δ-sensitivity through specific CCA adjusted mechanisms, we demonstrate the suitability of utilizing artificial delay to reduce δ-sensitivity in heterogeneous settings and the necessity of doing so for real world application performance.

As our heterogeneous experiments are conducted in emulation (except with applications in §7.3) we simply manipulate propagation delay and observe suitability and leave it to future works to implement λ-proxy solutions (or some new variant) in these and real-world network settings.

7.1 Heterogeneous Delay Complexity

In this subsection, we demonstrate an important phenomenon of CCAs such as Copa[7], Vegas[8], PCC[16, 17], and Libra[37] whose design aims to converge to unique equilibrium

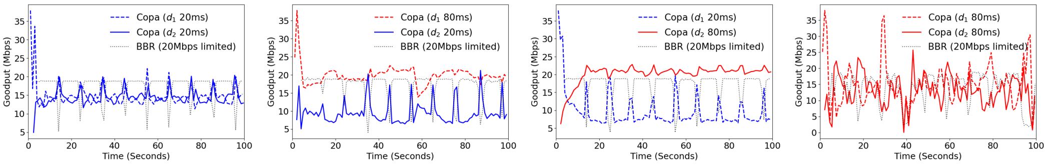


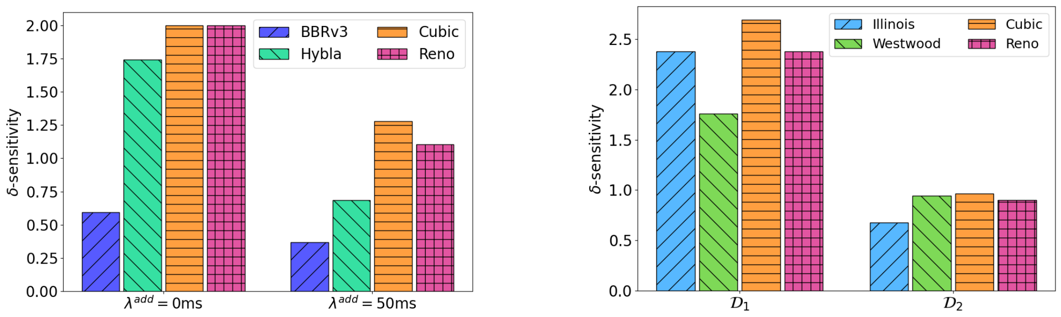
Figure 6 Demonstration that even CCAs which have minimal δ-sensitivity in homogeneous networks can exhibit δ-sensitivity in noisy real world environments. Shown here is Goodput achieved by two competing Copa flows bottlenecked at a 50Mbps capacity link which is shared by a (20Mbps max rate limited) BBR flow.

fixed points that do not depend on the individual flow's propagation delay (i.e., their δ-sensitivity is close to 0). We will show that when these flows compete across a bottleneck with alternate CCAs, then variations in *their* flow propagation delay (and not necessarily variations in the competing CCA's propagation delay) do in fact significantly alter their convergence points (i.e., their δ-sensitivity is no longer close to 0).

Our demonstration utilizes ns-3 simulation with an implementation of Copa and BBR [52] in a simple heterogeneous network setting $\mathcal{I}_{het}$ where two Copa flows, f_1 and f_2 send data over a 50Mbps bottleneck that is shared by a BBR flow f_3 which is is intended to emulate a video streaming application that has a maximum sending rate of 20Mbs [48]. Limiting the video flow (BBR) flow ensures that at least 30Mbps of available capacity is reserved for the two Copa flows. In the comparable homogeneous network setting $\mathcal{I}_{hom}$ which does not include the BBR flow, the two Copa flows consistently converge to identical rates independent of differences in their propagation delays. Figure 6 demonstrates sensitivity that occurs in the heterogeneous setting $\mathcal{I}_{het}$ considering $\mathcal{D} = \{(d_1, d_2, 5)\}$ with $d_1, d_2 \in \{20, 80\}$, i.e., Copa flow propagation delays of the two flows are any sampling of 20 and 80ms, and the BBR flow has propagation delay of 5ms.[8] Each plot shows goodput achieved over time for a given $D \in \mathcal{D}$. The first Copa flow (f_1) is plotted with a dashed line and starts 3 seconds before the other Copa flow f_2 which is plotted with a solid line. The rate limited BBR flow starts last and is plotted with a dotted gray line. When the two Copa flows have different propagation delays, they exhibit delay sensitivity with 19.7 vs 9.6Mbps and 21.0 vs 8.7Mbps average goodput achieved by the 80ms vs 20ms flow respectively. From these experiments we find $\delta_{\mathcal{I}_{het},\mathcal{D}}(i) \geq 1.3, i = 1, 2$. From experiments not shown in the figure we find Copa's sensitivity in the comparable homogeneous setting to be $\delta_{\mathcal{I}_{hom},\mathcal{D}}(i) \approx 0, i = 1, 2$.

A second important observation is that in a heterogeneous environment, when the Copa flows have identical propagation delays $\mathcal{D} = \{(d, d, 5)\}$ there is **negligible** δ-sensitivity, even when interacting with a flow that utilizes a different CCA (in this case BBR). For instance, with this restriction, $\delta_{\mathcal{I}_{het},\mathcal{D}}(2) = 0.19$. This importantly shows that the δ-sensitivity exhibited by Copa is with respect to variation of delay across the Copa flows. In other words, just having an additional CCA sharing the bottleneck (without changes to its propagation delay) significantly increases the δ-sensitivity of the Copa protocol when propagation delay variations are only among the Copa flows! While this phenomenon is demonstrated using Copa, the insight applies more generally across CCAs.

[8] We are focusing on how Copa's δ-sensitivity is affected by heterogeneous network settings: it is sufficient to show this with a single BBR where the different $D \in \mathcal{D}$ need not even vary the BBR flow's propagation delay.



(a) Naive approach

(b) Reducing Range

Figure 7 Reducing δ-sensitivity in heterogeneous CCA competition through altering the naturally occurring $\mathcal{D}$

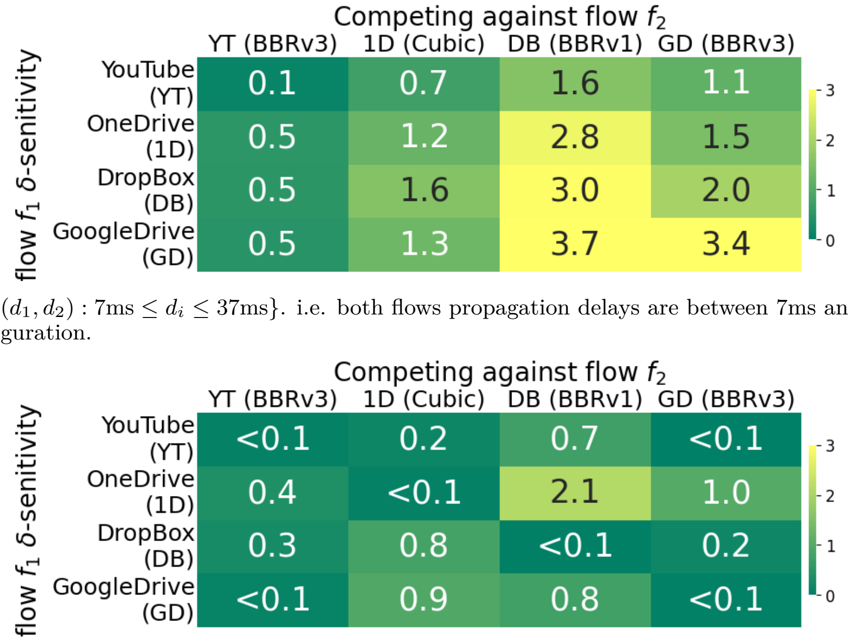
7.2 Simple applications of artificial delay

The goal of introducing artificial delay is to equalize propagation delays among competing flows to neutralize δ-sensitivity without requiring complex mechanisms implemented in the CCA, as demonstrated with the proxy in §6. In this section, we consider the robustness of this method by demonstrating that *even* naive approaches to adding artificial delay with errors in reaching the point of equalization can still reliably reduce δ-sensitivity and improve consistent performance.

The naive approach to introducing artificial delay is to add a constant fixed delay to all senders λ^{add} regardless of the naturally occurring propagation delay of the flow. With this method, any difference in natural propagation delays remains but the relative difference reduces leading to a reduction in δ-sensitivity. In other words, a flows sensitivity to the delay range $\mathcal{D}$ can be reduced solely by uniformly adding λ^{add} to all delays.

To demonstrate this phenomenon we run experiments between four competing flows utilizing four different CCAs in an emulated test-bed following that used in §6. Figure 7a plots the δ-sensitivity of each of the four flows managed by BBRv3, Hybla, Cubic, and Reno when congested by a single 100 Mbps capacity bottleneck over the range of delays $\mathcal{D} = \{10\text{ms} \leq d_i \leq 30\text{ms}\}$ without adding artificial delay (left) and then with 50ms of added delay (right). By adding 50ms of artificial delay, the range $\mathcal{D}$ that the flows operate over effectively becomes $\mathcal{D} = \{60\text{ms} \leq d_i \leq 80\text{ms}\}$. By reducing the proportional difference of asymmetric delay, the δ-sensitivity of all four flows is reduced.

Next, we demonstrate that δ-sensitivity is reduced by attempting to equalize propagation delays between competing flows even when the intended point of equalization is missed due to inaccurate estimates of the naturally occurring propagation delays. When all competing flows utilize buffer-filling loss-based CCAs, for example, it may be more difficult to make estimates of the underlying propagation delay without introducing additional mechanisms to make such estimates. Figure 7b shows the sensitivity of four competing flows managed by CCAs Illinois[35], Westwood[38], Cubic, and Reno when congested at a 100 Mbps bottleneck over range of delays $\mathcal{D}_1$ ranging from 5ms to 30ms, and $\mathcal{D}_2$ ranging from 35ms to 45ms to emulate scenario where the agreed upon point of equalization was 40ms with estimation errors of ±5ms. By artificially altering the delay range from $\mathcal{D}_1$ to $\mathcal{D}_2$, each flow's δ-sensitivity is reduced to less than 1, significantly improving performance consistency.



(a) $\mathcal{D} = \{(d_1, d_2) : 7\text{ms} \leq d_i \leq 37\text{ms}\}$. i.e. both flows propagation delays are between 7ms and 37ms in each configuration.

(b) $\mathcal{D} = \{(d_1, d_2) : 7\text{ms} \leq d_1 = d_2 \leq 37\text{ms}\}$. The same range as (a) above except restricted to the configurations where competing flows have identical delays.

Figure 8 δ-sensitivities of flow f_1 when competing against f_2 on a shared 50 Mbps bottleneck. **(a) shows full delay range; (b) isolates cases with equal delays. Comparing them reveals the impact of delay asymmetry versus point of equalization.**

7.3 Sensitivity in Applications

In this section, we return focus to popular application performance in practical settings and use δ-sensitivity to demonstrate that delay-induced unpredictability remains a significant impediment to predictable performance. As the sending hosts of popular applications are out of our control, we use artificial delay at the receiver end to control competing flow propagation delay.

Application Experiments Setup: Continuing with the experimental topology shown in Figure 1, we run experiments between competing simultaneous connections that are each sourced from one four applications - YouTube (YT), OneDrive (1D), GoogleDrive (GD), and DropBox (DB). The network configuration we use is influenced by that of [48] - setting the bottleneck capacity to 50 Mbps which reflects the median bandwidth of more than half the countries in the world. Similar to [48], our testbed varies propagation delay by artificially adding a fixed delay within the round trip. While [48] adds this delay at the bottleneck, we use NetEm [25] to add a fixed delay from all ACK departures at each receiver. The location of our receivers are such that, as observed via pings, all four applications have natural propagation delays below 7ms - allowing us to emulate propagation delays that are larger.

Note that DB is reported [43, 48, 42, 13] to utilize a variant of BBRv1 [12], 1D utilizes a variant of Cubic[23] and both GD and YT utilize variants of BBRv3[14].[9] Different

[9] However, applications often alter the implementations of CCAs in unknown ways [40] - furthering the importance of evaluating actual applications over CCA mechanisms alone.

applications have different throughput demands: YT transmits video data in chunks utilizing HTTP-chunking [20] where YT's throughput need *over time* to serve a 4k video does not exceed 20 Mbps but sends at rates larger than this when transmitting each chunk; file downloads DB, GD, 1D transmit a consistent stream of data until the file download is complete.

When considering ranges of delays (e.g. $\mathcal{D} = \{(d_1, d_2) : d_a \leq d_i \leq d_b\}$), $\mathcal{D}$ grows exponentially in the number of sessions as a function of the number of propagation delays considered per flow. For tractability, our $\mathcal{D}$ spaces propagation delays within ±5ms of one another, and includes the boundaries d_a and d_b, providing a lower bound on the measure of δ-sensitivity of all delays along the continuum.

δ-sensitivity reveals differences in robustness across application pairs. Figure 8a shows the δ-sensitivity of flow 1 ($\delta_{I,\mathcal{D}}(1)$) sourced from the application indicated on the left of the table when competing in a two flow network setting ($\mathcal{I}$) against flow 2 sourced from the application indicated above each column in the range $\mathcal{D} = \{(d_1, d_2) : 7\text{ms} \leq d_i \leq 37\text{ms}\}$. *We choose this range to ensure practicality as the maximum delay observed for services tested in [48] was 40 ms.* Figure 8b shows δ-sensitivity over $\mathcal{D}_{eq}$ - the subset of $\mathcal{D}$ where both competing flows have identical propagation delays. Note that each application exhibits sensitivities that indicate unpredictability in $\mathcal{D}$ with significant δ-values revealing how this depends on the CCA and workload of the competing application.

Comparison across (a) and (b) in Figure 8: While all flows show some level of sensitivity over $\mathcal{D}$ in (a), the restriction to equal delay configurations in (b) highlights which effects are due to differences in CCA alone. For instance, Dropbox's $\delta = 2.0$ when competing with Google Drive in (a) drops to $\delta = 0.8$ in (b), revealing the role of asymmetric delay.

The δ-sensitivity in (a) suggests that there is need for implementing a solution such as λ-proxy and the remaining δ-sensitivity in (b) would be mitigated by all agreeing on a specific point of equalization λ (larger than delays in $\mathcal{D}$).

8 Discussion

The widespread adoption of BBR by content providers is evidence that application performance is valued over global notions of fairness. By adjusting the perceived network path as opposed to the implementation of the employed CCA, artificial delay mechanisms avoid interfering with the performance goals the CCA is designed to achieve. Artificial delay can be applied by end-hosts regardless of the specific CCA that is being utilized. We have focused our experiments on BBR because of its dominating popularity in applications.

Our evaluation is not exhaustive with respect to the wide range of conditions encountered in real-world data transfers, and further investigation is required into issues such as convergence speed, behavior of short-lived flows, startup dynamics, and robustness over WiFi and cellular links with variable capacity. Nevertheless, our experiments provide evidence that artificial delay constitutes a viable tool available to end-hosts. Many of the potential challenges introduced by inflated propagation delay depend on the CCA in use as well as the propagation delay level chosen as the point of equalization.

Current CCAs are generally designed to adjust their sending rates in ways that maintain effectiveness across a broad spectrum of propagation delays. Consequently, inflating propagation delay to an agreed-upon value may cause certain CCAs to experience issues with convergence speed, startup behavior, or similar dynamics. However, these difficulties stem from the mechanisms of the CCA itself rather than from the inflated delay. Under the assumption that all flows operate with a standardized propagation delay, modifications to existing CCAs - or the design of entirely new ones - could incorporate mechanisms specifically tailored to that point of equalization.

Deciding in generality what constitutes the best point of equalization λ is a challenging question to answer. Larger λ values can equalize propagation delay of a wider range of plausible propagation delays whereas more modest λ-values could capture the majority of traffic while minimizing the delay on congestion signal feedback.

9 Related Work

Studies have shown the heterogeneous nature of the Internet [43, 41, 56], and reduced adoption of and attention to traditionally 'fair' CCAs [9, 55, 10, 34]. Measuring Internet heterogeneity has been explored by many works. [55] proposed a deployment threshold for a new CCA based on the *harm* done to an existing legacy CCA over all possible networks. [9] proposed a new point of throughput allocation inline with the respective service agreements of competing flows and independent of the utilized CCAs. [48] was the motivation for our application experimental setup and was the first to measure fairness (in the max-min sense) over heterogeneous real world applications.

RTT-unfairness in BBR congestion control has been widely researched, with theoretical analysis [50, 26] and experimental evaluations showing *RTT-unfairness* within BBRv1 [28, 26, 47, 33, 6], v2 [51, 4, 50], and v3 [21]. Studies have proposed delay-insensitive adaptions to BBR that include adjustments to the congestion window[47, 4, 45] and adjustments to pacing rates[57, 36]. Other CCAs include mechanisms for delay-insensitive equilibrium rates [8, 31, 7, 11, 37, 17, 16]. [60] proves that CCAs cannot guarantee fairness and a fixed queue delay at the same time. [32] shows how algorithmic mechanisms that intend to cause delay-insensitive performance cause issues with stability and scalability.

10 Conclusion/Future Directions

We introduced the notion of *δ-sensitivity* as a practical way to capture the degree to which propagation delays can influence the throughput allocations of competing flows. Unlike traditional "fairness" metrics that assess a particular bandwidth division at a single operating point, δ-sensitivity highlights *how a flow's performance might vary* as propagation delays change. Our experiments in controlled testbeds, real-world cloud environments, and application settings reveal that many modern congestion control algorithms are highly δ-sensitive, leading to significant performance imbalance among flows that differ in their minimum RTT.

We proposed *artificial added delay* mechanisms (particularly, the λ-proxy) to address this issue. By artificially inflating the RTT *only on ACKs*, these mechanisms reduce or even eliminate the dependency of throughput allocations on the true propagation delay for long-lived flows. As a result, when flows face different propagation delays, they can still converge to allocations unhampered by RTT bias. Our evaluations demonstrated that λ-proxy preserves high utilization and avoids implicit data-latency inflation.

Future Directions.

The notion of δ-sensitivity and its mitigation through artificial added delay open several avenues for further exploration:

- **Adaptive Delay Insertion.** While we demonstrated a static λ-proxy, future designs might automatically tune λ based on live network conditions, preventing unnecessary inflation of RTT once allocations are stable.
- **Application-Aware Configurations.** In practice, different flows (e.g., bulk file transfers vs. interactive sessions) have varying QoS needs. An application-aware approach could selectively apply artificial delay only when beneficial - balancing throughput, latency, and user experience.
- **Scalable Deployments.** Large-scale evaluations (e.g., across multiple data centers or CDN edge nodes) can uncover how λ-proxy behaves with dozens or hundreds of competing flows, including potential interactions among flows using different CCAs.
- **Integration with Next-Generation CCAs.** Artificial delay does not conflict with evolving congestion control schemes that integrate explicit signals (e.g., ECN or in-network telemetry). Studying how δ-sensitivity can be further minimized with more sophisticated feedback loops would extend the utility of our approach.

By exposing and mitigating δ-sensitivity through a simple, end-host tool, we hope this work spurs additional research and pragmatic solutions for managing RTT-induced performance inequities in modern networks.

References

1 Selenium. `https://www.selenium.dev/`. n.d.

2 ss(8) — socket statistics. `https://man7.org/linux/man-pages/man8/ss.8.html`. Linux manual page.

3 Global internet phenomena report. Technical report, Sandvine Incorporated ULC, 2023. URL: `https://www.prnewswire.com/news-releases/sandvines-2023-global-internet-phenomena-report-shows-24-jump-in-video-traffic-with-netflix-volume-overtaking-youtube-301723445.html`.

4 Akshita Abrol, Purnima Murali Mohan, and Tram Truong-Huu. Fairtt: An empirical approach for enhanced rtt fairness and bottleneck throughput in bbr. *arXiv preprint arXiv:2403.19973*, 2024.

5 Ian F Akyildiz, Özgür B Akan, Chao Chen, Jian Fang, and Weilian Su. Interplanetary internet: state-of-the-art and research challenges. *Computer Networks*, 43(2):75–112, 2003.

6 Venkat Arun, Mohammad Alizadeh, and Hari Balakrishnan. Starvation in end-to-end congestion control. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 177–192, New York, NY, USA, 2022. Association for Computing Machinery. `doi:10.1145/3544216.3544223`.

7 Venkat Arun and Hari Balakrishnan. Copa: Practical Delay-Based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 329–342, Renton, WA, April 2018. USENIX Association. URL: `https://www.usenix.org/conference/nsdi18/presentation/arun`.

8 L.S. Brakmo and L.L. Peterson. Tcp vegas: end to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995. `doi:10.1109/49.464716`.

9 Lloyd Brown, Albert Gran Alcoz, Frank Cangialosi, Akshay Narayan, Mohammad Alizadeh, Hari Balakrishnan, Eric Friedman, Ethan Katz-Bassett, Arvind Krishnamurthy, Michael Schapira, and Scott Shenker. Principles for internet congestion management. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 166–180, New York, NY, USA, 2024. Association for Computing Machinery. `doi:10.1145/3651890.3672247`.

10 Lloyd Brown, Yash Kothari, Akshay Narayan, Arvind Krishnamurthy, Aurojit Panda, Justine Sherry, and Scott Shenker. How i learned to stop worrying about cca contention. In *Proceedings*

of the 22nd ACM Workshop on Hot Topics in Networks, HotNets '23, page 229–237, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3626111.3628204.

11 Carlo Caini and Rosario Firrincieli. Tcp hybla: a tcp enhancement for heterogeneous networks. *Int. J. Satell. Commun. Netw.*, 22(5):547–566, September 2004. doi:10.1002/sat.799.

12 Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Communications of the ACM*, 60(2):58–66, 2017.

13 Neal Cardwell, Yuchung Cheng, Kevin Yang, David Morley, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Van Jacobson, Ian Swett, Bin Wu, and Victor Vasiliev. Bbrv3 overview and google deployment. Slides, IETF 119 Congestion Control Working Group Meeting, March 2024. IETF 119: Brisbane, March 21, 2024. URL: https://datatracker.ietf.org/meeting/119/materials/slides-119-ccwg-bbrv3-overview-and-google-deployment-00.

14 Neal Cardwell and Google BBR Team. BBRv3: algorithm bug fixes and public internet deployment. Technical report, Internet Engineering Task Force (IETF), 2023.

15 Dah-Ming Chiu and Raj Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.

16 Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. PCC: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 395–408, Oakland, CA, May 2015. USENIX Association. URL: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/dong.

17 Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-Learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 343–356, Renton, WA, April 2018. USENIX Association. URL: https://www.usenix.org/conference/nsdi18/presentation/dong.

18 Nandita Dukkipati and Nick McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, January 2006. doi:10.1145/1111322.1111336.

19 Eric Gavaletz and Jasleen Kaur. Decomposing rtt-unfairness in transport protocols. In *2010 17th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN)*, pages 1–6. IEEE, 2010.

20 Monia Ghobadi, Yuchung Cheng, Ankur Jain, and Matt Mathis. Trickle: Rate limiting YouTube video streaming. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 191–196, Boston, MA, June 2012. USENIX Association. URL: https://www.usenix.org/conference/atc12/technical-sessions/presentation/ghobadi.

21 Jose Antonio Gomez Gaona, Elie Kfoury, Jorge Crichigno, and Gautam Srivastava. Evaluating tcp bbrv3 performance in wired broadband networks. *Available at SSRN 4594375*, 2023.

22 Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. Elasticity detection: A building block for internet congestion control. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 158–176, 2022.

23 Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, jul 2008. doi:10.1145/1400097.1400105.

24 Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 29–42, 2017.

25 Stephen Hemminger. Network emulation with netem. Open Source Development Lab, April 2005. Available online: http://devresources.linux-foundation.org/shemminger/netem/LCA2005_paper.pdf.

26 Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of bbr congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10, 2017. doi:10.1109/ICNP.2017.8117540.

27 Van Jacobson. Congestion avoidance and control. *ACM SIGCOMM computer communication review*, 18(4):314–329, 1988.

28 Benedikt Jaeger, Dominik Scholz, Daniel Raumer, Fabien Geyer, and Georg Carle. Reproducible measurements of tcp bbr congestion control. *Computer Communications*, 144, 05 2019. doi:10.1016/j.comcom.2019.05.011.

29 Jeffrey Jaffe. Bottleneck flow control. *IEEE Transactions on Communications*, 29(7):954–962, 1981.

30 R. Jain, D. Chiu, and W. Hawe. A quantitative measure of fairness and discrimination for resource allocation in shared computer system. Technical report, Eastern Research Laboratory, Digital Equipment Corp., 1984.

31 Cheng Jin, David X Wei, and Steven H Low. Fast tcp: motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, volume 4, pages 2490–2501. IEEE, 2004.

32 Frank Kelly. Fairness and stability of end-to-end congestion control. *European journal of control*, 9(2-3):159–176, 2003.

33 Geon-Hwan Kim and You-Ze Cho. Delay-aware bbr congestion control algorithm for rtt fairness improvement. *IEEE Access*, 8:4099–4109, 2020. doi:10.1109/ACCESS.2019.2962213.

34 Ike Kunze, Jan Rüth, and Oliver Hohlfeld. Congestion control in the wild—investigating content provider fairness. *IEEE Transactions on Network and Service Management*, 17(2):1224–1238, 2020. doi:10.1109/TNSM.2019.2962607.

35 Shao Liu, Tamer Başar, and R. Srikant. Tcp-illinois: a loss and delay-based congestion control algorithm for high-speed networks. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06, page 55–es, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1190095.1190166.

36 Shiyao Ma, Jingjie Jiang, Wei Wang, and Bo Li. Fairness of congestion-based congestion control: Experimental evaluation and analysis. *arXiv preprint arXiv:1706.09115*, 2017.

37 Gustavo Marfia, Claudio Palazzi, Giovanni Pau, Mario Gerla, M. Y. Sanadidi, and Marco Roccetti. Tcp libra: exploring rtt-fairness for tcp. In *Proceedings of the 6th International IFIP-TC6 Conference on Ad Hoc and Sensor Networks, Wireless Networks, next Generation Internet*, NETWORKING'07, page 1005–1013, Berlin, Heidelberg, 2007. Springer-Verlag.

38 Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, MobiCom '01, page 287–297, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/381677.381704.

39 Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

40 Ayush Mishra, Sherman Lim, and Ben Leong. Understanding speciation in quic congestion control. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 560–566, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517745.3561459.

41 Ayush Mishra, Lakshay Rastogi, Raj Joshi, and Ben Leong. Keeping an eye on congestion control in the wild with nebby. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 136–150, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672223.

42 Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. The great internet tcp congestion control census. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(3), December 2019. doi:10.1145/3366693.

43 Ayush Mishra, Wee Han Tiu, and Ben Leong. Are we heading towards a bbr-dominant internet? In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 538–550, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517745.3561429.

44 Aarti Nandagiri, Mohit P. Tahiliani, Vishal Misra, and K. K. Ramakrishnan. Bbrvl vs bbrv2: Examining performance differences through experimental evaluation. In *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, pages 1–6, 2020. doi:10.1109/LANMAN49260.2020.9153268.

45 Charles Kihungi Njogu, Wang Yang, Humphrey Waita Njogu, and Adrian Bosire. Bbr-with enhanced fairness (bbr-efra): A new enhanced rtt fairness for bbr congestion control algorithm. *Computer Communications*, 200:95–103, 2023.

46 Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling tcp throughput: A simple model and its empirical validation. In *Proceedings of the ACM SIGCOMM'98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 303–314, 1998.

47 Wansu Pan, Haibo Tan, Xiru Li, and Xiaofeng Li. Improved rtt fairness of bbr congestion control algorithm based on adaptive congestion window. *Electronics*, 10(5), 2021. URL: https://www.mdpi.com/2079-9292/10/5/615, doi:10.3390/electronics10050615.

48 Adithya Abraham Philip, Rukshani Athapathu, Ranysha Ware, Fabian Francis Mkocheko, Alexis Schlomer, Mengrou Shou, Zili Meng, Srinivasan Seshan, and Justine Sherry. Prudentia: Findings of an internet fairness watchdog. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 506–520, 2024.

49 Adithya Abraham Philip, Ranysha Ware, Rukshani Athapathu, Justine Sherry, and Vyas Sekar. Revisiting tcp congestion control throughput models & fairness properties at scale. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, page 96–103, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3487552.3487834.

50 Simon Scherrer, Markus Legner, Adrian Perrig, and Stefan Schmid. Model-based insights on the performance, fairness, and stability of bbr. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 519–537, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517745.3561420.

51 Yeong-Jun Song, Geon-Hwan Kim, Imtiaz Mahmud, Won-Kyeong Seo, and You-Ze Cho. Understanding of bbrv2: Evaluation and comparison with bbrv1 congestion control algorithm. *IEEE Access*, 9:37131–37145, 2021. doi:10.1109/ACCESS.2021.3061696.

52 SoonyangZhang. ns3-tcp-bbr, 2021–. URL: https://github.com/SoonyangZhang/ns3-tcp-bbr.

53 The Google BBR Team. bbr. https://github.com/google/bbr, 2024.

54 Utilities for controlling TCP/IP networking and traffic project. iproute2. https://man7.org/linux/man-pages/man8/ip.8.html, 2011.

55 Ranysha Ware, Matthew K Mukerjee, Srinivasan Seshan, and Justine Sherry. Beyond jain's fairness index: Setting the bar for the deployment of congestion control algorithms. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 17–24, 2019.

56 Ranysha Ware, Adithya Abraham Philip, Nicholas Hungria, Yash Kothari, Justine Sherry, and Srinivasan Seshan. Ccanalyzer: An efficient and nearly-passive congestion control classifier. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 181–196, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3651890.3672255.

57 Shuang Yang, Yuquan Tang, Wansu Pan, Huadong Wang, Dandan Rong, and Zhirong Zhang. Optimization of bbr congestion control algorithm based on pacing gain model. *Sensors*, 23(9):4431, 2023.

58 Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. Axiomatizing congestion control. In *Abstracts of the 2019 SIGMETRICS/Performance Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '19, page 51–52, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3309697.3331501.

59 Bo Zhang, TS Eugene Ng, Animesh Nandi, Rudolf Riedi, Peter Druschel, and Guohui Wang. Measurement based analysis, modeling, and synthesis of the internet delay space. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 85–98, 2006.

60 Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. Ecn or delay: Lessons learnt from analysis of dcqcn and timely. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '16, page 313–327, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2999572.2999593.