Improving Evaluation of Heterogeneous Congestion Control Algorithm Interactions

Ranysha Ware
Swarthmore College
Swarthmore, PA, USA

Haochen Zhang
Tsinghua University
Beijing, China

Isabel Suizo
Carnegie Mellon University
Pittsburgh, PA, USA

Srinivasan Seshan
Carnegie Mellon University
Pittsburgh, PA, USA

Justine Sherry
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

A critical part of new congestion control algorithm (CCA) proposals is an evaluation that a new CCA will reasonably share with already widely-deployed CCAs. However, the methodology for this evaluation is both inconsistent and inefficient due to the complexity of inter-CCA interactions under highly diverse network settings. We address these challenges by developing new metrics for evaluating fairness, developing an algorithm for determining when experiments converge, and applying this methodology to automated evaluation tool Mahak. In addition, we present HarmGen, a new tool that can efficiently explore a large search space using a genetic algorithm, to find network settings and workloads where there are poor interactions between heterogeneous CCAs. We show that with an identical budget of 300 experiments, HarmGen outperforms hill climbing and random search, and finds high harm values similar to a parameter sweep of 3500 experiments. With Mahak and HarmGen, we identify trends in BBR's evolution as well as issues with new L4S deployments.

CCS Concepts

• **Networks** → **Transport protocols**; *Network measurement*; *Network performance analysis*; Network experimentation.

Keywords

congestion control, fairness, harm, heterogeneous CCA interactions, BBR, L4S, network evaluation methodology, convergence, genetic algorithms, testbed experimentation

ACM Reference Format:
Ranysha Ware, Haochen Zhang, Isabel Suizo, Srinivasan Seshan, and Justine Sherry. 2026. Improving Evaluation of Heterogeneous Congestion Control Algorithm Interactions. In *ACM SIGCOMM 2026 Conference (SIGCOMM '26), August 17–21, 2026, Denver, CO, USA.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3789240.3829156

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGCOMM '26, August 17–21, 2026, Denver, CO, USA
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-2467-1/26/08
https://doi.org/10.1145/3789240.3829156

1 Introduction

With the growing heterogeneity of congestion control algorithms (CCAs) on the Internet [30, 32, 33, 51], it is crucial to ensure that heterogeneous algorithms *share* network resources. To this end, researchers have proposed different forms of fairness [9, 21, 22, 28, 35, 54], recommended bounds for how much 'harm' CCAs might be permitted to cause each other [25, 49], and measured how unfairness between CCAs impacts application performance [39].

While inter-CCA fairness evaluation is critical, it is also difficult, as we saw with the development and deployment of BBR by Google. BBRv1 was first proposed and deployed in the Linux kernel in 2016, to fix issues with bufferbloat and loss-based CCAs. However, researchers consistently found problems with how BBRv1 flows interact both with other BBRv1 flows [50] and with loss-based CCAs [18, 24, 42, 50] despite widespread deployment of BBRv1 at major content providers [32, 33, 51]. These undesirable fairness outcomes (as well as other issues with BBRv1 like excessive retransmissions) were significant enough for Google to develop BBRv2 and later BBRv3 to fix these issues. As of 2025, all Google sites now use BBRv3 with plans to push it into the Linux kernel, replacing BBRv1 [13]. Even so, recent work shows that BBRv3 is *still unfair to Cubic* [56] (we will revisit these findings in §3). Better coexistence with Cubic is currently a requirement for BBRv3 standardization at the IETF [14]. The example of BBR highlights both the importance of evaluating fairness for CCA developers and the difficulty of that evaluation.

How can we improve inter-CCA fairness testing? Let us first consider what that evaluation looks like today. A survey of new CCA proposals reveals a common methodology: authors select a set of network settings, run several competing flows concurrently for a fixed duration, and analyze metrics to judge whether the resulting allocation is sufficiently fair [5, 7, 17, 18, 36, 53]. These studies make a variety of choices in each of these dimensions: different network settings, different experiment durations, and different fairness metrics [4, 19, 24, 29, 31, 39, 41, 42, 48, 56].

This variety is a consequence of the challenges researchers face when trying to design a comprehensive and scientifically sound evaluation. Inter-CCA interactions are often volatile, with large oscillations and long convergence times. There is no precise definition of 'steady-state', which is reflected in the wide array of choices in experiment duration. What is considered 'fair' also varies with the workload and whether it consists of short or long flows. It is also unclear which network settings are the 'right' ones to test.

Our contributions. To address these challenges, we define a new framework for evaluating interactions of heterogeneous congestion control algorithms that is consistent with what we want to measure and will help algorithm designers and researchers find the settings where an algorithm does and does not perform well. In this methodology, we define a new metric, *relative harm*, for both long-flow and short-flow workloads, extending prior work [49]. We discuss the impact of experiment duration on results and develop a novel *convergence algorithm* to find the time when an experiment reaches equilibrium. Rather than cherry-picking a few network settings, we apply our methodology to automated tools for exploring a large state space of possible network settings. We extend a powerful existing tool, Mahak [37], which uses active learning to predict performance in a search space. In our evaluation of Mahak, we find that it has low prediction error for experiments with long-running flows, but it faces some challenges in exploring the search space for more complex short-flow workloads.

In addition to applying our methodology to Mahak, we develop a new tool, HarmGen, which uses a genetic algorithm to find worst-case harm scenarios for both long and short-flow experiments. In our evaluation of HarmGen, we find it outperforms random search and hill climbing, while finding a diverse set of high-harm scenarios in comparison to a parameter sweep over the whole space. Applying our methodology to two different kinds of tools demonstrates its broad applicability. Lastly, we use Mahak and HarmGen to show the efficacy of our methodology by considering two cases. In the first case study, we compare BBRv1's interactions with Cubic to BBRv3's interactions with Cubic. While we find a large performance improvement with BBRv3, we also find cases where there is no improvement. In our second case study, we investigate TCP Prague, a CCA designed for L4S, and how it interacts with Cubic.

Code. Implementations of Mahak and HarmGen can be found here: `https://github.com/cmu-snap/HarmGen` .

The rest of this paper is organized as follows. First, we motivate using *relative harm* rather than JFI as the metric for poor outcomes, for both long-flow and short-flow workloads, in §3. Second, we discuss how the duration of experiments can impact outcomes and develop an algorithm to determine when CCA interactions have "converged" in §4. We then apply our methodology to Mahak in §5. We present HarmGen in §6, discussing the design and evaluation. We show an example use case of Mahak and HarmGen in §7 studying BBRv3 interactions with Cubic and L4S. Finally, we discuss related work and alternatives to empirical evaluation in §8 and conclude in §9.

2 Challenges in Evaluating CCA Interactions

There are several challenges in evaluating inter-CCA interactions. We outline these as challenges we need to address in this work.

The metric matters: Definitions of "fairness" are consistently under debate with arguments that achieving equal flow-rate fairness for inter-CCA interactions is neither an achievable nor realistic goal [9, 11, 49, 54]. Publications with experiments showing interactions between recently proposed CCAs and Cubic, all report slightly different metrics: These metrics include JFI [56], throughput ratio [18, 24], throughput share of Cubic [42, 50], friendliness ratio[5], ratio of throughput to ideal fair share [7], and novel metrics like *fness* in Kunze et al. [29] and *FCT slowdown* in Zapletal et al. [54]. Ultimately, most (but not all) of these metrics are trying to quantify the same thing: is there equal rate-fair sharing, and how far away from that goal are the outcomes? Is the new CCA too unfair to Cubic? Is this CCA safe to deploy alongside Cubic? We take inspiration from Ware et al. [49] and Islam et al. [25], and argue that harm is the right metric to quantify deployability. Harm as a metric has seen some recent adoption in studies of inter-CCA interactions [19, 36, 41] for long-flow workloads. We further motivate using *relative harm* as our metric in §3 and define relative harm for both long-flow and short-flow workloads.

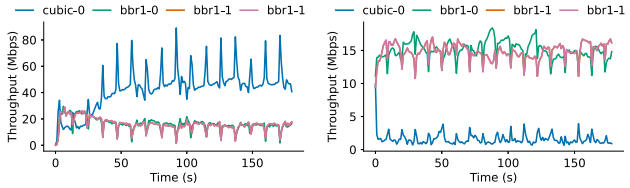
How long we run experiments matters: CCA performance evaluations must happen over some workload. In this work, we design experiments for both long-flow and short-flow workloads. While most flows on the Internet are short (and fast) [8, 27, 54, 57], we typically see past work focusing on "long-running flows" as an approximation of "infinitely backlogged flows" running for some duration. In recent CCA proposals and evaluations, the duration of flows to measure inter-CCA interactions ranges from 10-60 seconds [7, 41, 48] to 2-10 minutes [4, 5, 19, 24, 39, 42, 50, 53, 56];

As we will show in §3, inter-CCA interactions may take a long time to "converge" with key metrics changing depending on the period over which an observation is made. Indeed, we see two concrete examples of researchers explicitly noting the importance of what happens after convergence. Pappone at al. [36] argue their new CCA proposal has 'good enough' fairness with Cubic by considering convergence: "Over time, Mutant converges to the fair share of the link capacity". Abbasloo et al. [4] run experiments for 120s as they notice experiments taking a long time to reach 'steady state' and wanted to "make sure the results can present meaningful TCP-friendliness scores".

In this work, we show how definitions of convergence for intra-CCA interactions that focus on converging to a fixed value, like equal fair share, do not apply to inter-CCA interactions. Further, we develop an algorithm to determine if flows have converged in an automated way. We run experiments for 3 minutes and use our convergence algorithm to find the point when we are reasonably certain the throughput has stabilized, and compute harm after this point. We describe this algorithm and rationale in §4.

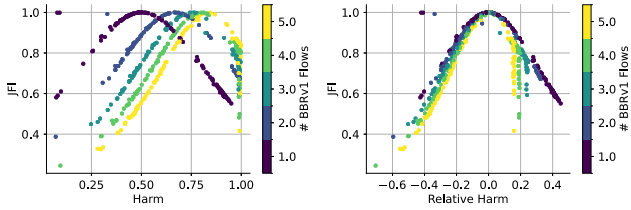
In addition to considering these long-flow workloads, we make a first step towards determining a relative harm metric that works for workloads with short flows. We distinguish the two by convergence: long flows run long enough to reach a measurable steady state, while short flows complete before converging. Defining a short-flow duration in terms of CCA behavior is impractical, so we set the duration of short flows to fixed values well below convergence.

What network settings we test matters: For CCA developers that do attempt this evaluation, they test in just a few "common" case scenarios. However, the outcomes are highly dependent on the scenario. Further, there are many possible "realistic" scenarios given the diversity in network quality around the world. For example, the range of access link bandwidth varies by three orders of magnitude across regions of the world [16, 47]; depending on access medium there might be high or low background loss; ISPs may have different settings for buffer capacity, drop policies, or rate



(a) 100bw-320rtt-1.5bdp: -0.2 relative harm and 0.8 JFI **(b) 50bw-20rtt-1.5bdp: 0.2 relative harm and 0.8 JFI**

Figure 1: 3 BBR flows vs. 1 Cubic flow throughput over time. Both of these scenarios have the same JFI, yet in scenario (a) Cubic dominates, while in scenario (b), BBR dominates.



(a) Absolute harm **(b) Relative harm**

Figure 2: Comparison of absolute and relative harm for 1 Cubic flow vs. 1-5 BBR flows for varying network settings.

shaping – and each of these parameters can impact performance outcomes. Testing a few scenarios is not necessarily going to illuminate the bad scenarios CCA developers care about remedying. Therefore, developers are faced with the challenge of identifying under what scenarios their CCA coexists well with others and under what scenarios it might fail—a large space of possible scenarios that is intractable to test exhaustively.

As workloads and CCAs themselves grow more complex, manually predicting these scenarios will not suffice. Thus, we formulate this problem as an optimization problem: Given a metric (e.g., harm), we want to find the network settings that maximize it. While existing frameworks, such as Mahak [37], can be extended to predict performance over a large search space (which we describe in §5), for complex short-flow workloads, they struggle to explore enough of the search space to have low prediction error.

Hence, we propose HarmGen, a genetic algorithm designed specifically for evaluating interactions between flows of various lengths from different CCAs: Given the harm as the objective function (input with network setting and output is the harm), what settings have the highest harm when two different CCAs compete? We describe HarmGen in §6.

3 Measuring Harm

The first step in refining our framework for evaluating heterogenous algorithm interactions is defining metrics. The Congestion Control Working Group of the IETF articulates the principle underlying our metric: "*In contexts where differing congestion control algorithms are used, it is important to understand whether the proposed congestion control algorithm could result in more harm than algorithms published in previous Standards Track RFCs, and to flows sharing a common bottleneck.*" [20] This argument was first presented by Ware et al. [49] and later supported by Islam et al. [25]. In addition, it has seen recent adoption in CCA evaluations [19, 36, 41].

In contrast to flow-rate fairness metrics like JFI, absolute harm [49] directly measures the performance degradation caused by competing flows when new flows enter the network. More specifically, harm computes the fractional difference between a CCA when it competes "alone" β_{solo} versus when it is under competition $\beta_{compete}$ with another CCA for some metric m[1]:

$$Harm = \frac{m(\beta_{solo}) - m(\beta_{compete})}{m(\beta_{solo})} \quad (1)$$

Fig. 1 illustrates an example of absolute harm's advantage over JFI.[2] In both scenarios, 3 BBRv1 flows compete against one Cubic flow, and the JFI is 0.8. However, in scenario 1a, Cubic dominates the bandwidth allocation, while in scenario 1b the opposite occurs. Operators would certainly be interested in both cases: an outcome where a novel CCA is underperforming and one where the CCA harms competition are both causes for concern. However, using JFI as a measure conflates these distinct scenarios. In contrast, the harm in scenario 1a is negative when Cubic harms BBR, and in scenario 1b is positive when BBR harms Cubic.

Using absolute harm as a metric, an operator can configure automated tools like Mahak (§5) and HarmGen (§6) to discover scenarios where a new CCA causes significant slowdown to competitors. The operator can also configure these tools to discover scenarios where a new CCA itself suffers unexpectedly severe slowdowns. Using JFI as a metric, these tools cannot distinguish these two problems.

Unfortunately, absolute harm also fails to help us identify significant testing scenarios that are worth human attention. The problem lies in expectations: in a scenario with 3 flows with a 4th flow entering the network, we would expect to see 25% harm on the existing flows; in a scenario with 2 flows with a 3rd flow entering, we would expect to see 33% harm. However, we would not say that the second scenario is 'worse' than the first (indeed, both scenarios behave exactly as desired).

3.1 Relative Harm for Long Flows

The example above illustrates why comparing harmfulness across different scenarios is challenging, especially when there are different numbers of flows. Hence, *we need to normalize somehow by the desired outcome, given the network setting and number of competing flows.* To do this, we introduce relative harm.

We compute relative harm by computing the harm if there was perfect fair sharing with the number of flows and subtracting *expected absolute harm* from *observed absolute harm.* With N_β flows and N_α flows, we compute the expected harm as the expected performance for the β flows if there was equal sharing:

$$RelativeHarm = \frac{m(\beta_{solo}) - m(\beta_{compete})}{m(\beta_{solo})} - (1 - \frac{N_\beta}{N_\beta + N_\alpha}) \quad (2)$$

[1]This is for a 'more is better' metric like throughput. Absolute harm can also be used to test metrics other than throughput – by measuring the performance degradation in latency, or loss rate, e.g. – JFI cannot measure these [49]

[2]Figure captions encode the network setting as `Xbw-Yrtt-Zbdp`: X is the bottleneck bandwidth (Mbps), Y the base RTT (ms), and Z the bottleneck queue size as a multiple of the bandwidth-delay product (BDP).

A value of 0 relative harm means that the harm matches expectations; anything above 0 means there is some harm done to β by α flows, and a negative value means there is no harm done to β. Fig. 2 contrasts absolute harm, relative harm, and JFI. Fig. 2b shows that relative harm centers values with no harm and perfect fairness (high JFI) at 0, whereas Fig. 2a shows how absolute harm alone provides a range of values for equal sharing.

A final question is how to aggregate relative harm values when there are multiple flows in competition. Here, the operator has several natural choices for how to analyze harm: they might look for scenarios in which the worst-case relative harm is high (*i.e.*, how much harm does the worst-performing flow incur?), they might also consider the average relative harm across all flows. Any of these choices is compatible with our methodology, and operators can decide what they care about.

3.2 Relative Harm for Short Flows

Recent literature often overlooks the performance degradation that short flows inflict upon long-running flows [4, 24, 36, 41]. Unfortunately, relative harm as defined above does not easily apply to workloads with short flows. Because sampling short-flow start times randomly would make the search space intractable, we set them to fixed times for feasibility and reproducibility. Specifically, we configure one long-running flow with a duration of 120 seconds, utilizing the first 60 seconds to allow the long flow to reach steady state. Short flows are introduced at $t = 85$s (during the second minute) and persist for a duration of 10 seconds. We construct specific short-flow scenarios for particular kinds of harm; protocol designers can add their own.

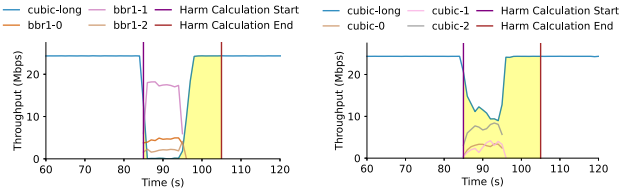
Adapting relative harm to short flows raises three complexities. First, over what time period should harm be calculated? While limiting the measurement to the short flow's lifecycle seems intuitive, we have observed, as illustrated in Fig. 4a, that the performance impact on long flows occasionally persists even after the short flows have completed. To capture this lingering effect, we define the observation window as twice the duration of the short flow's lifecycle.

Second, what baseline should be subtracted from the observed absolute harm? In the standard definition of relative harm for long flows, we use "fair sharing" as the steady-state ideal to normalize the metric. However, short flows are transient and may not reach convergence quickly enough to apply steady-state fairness models. Therefore, aligning with the proposal in Ware et al. [49], our metric is status-quo biased. We define the "ideal" scenario as the harm short β flows do to other β flows. We compare the absolute harm caused by the new CCA α short flows against this baseline. Thus, the metric is defined as follows:

$$RelativeHarm = Harm_{\alpha_{short} \rightarrow \beta_{long}} - Harm_{\beta_{short} \rightarrow \beta_{long}} \quad (3)$$

Third, how should we define the absolute harm term ($Harm$) in this context? Leveraging the multi-metric flexibility of the original harm framework, we propose two distinct formulations based on application requirements.

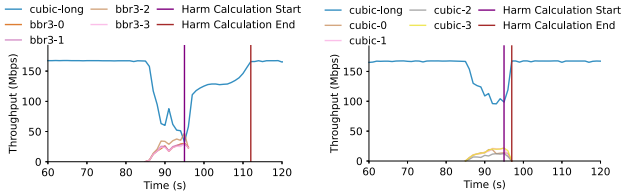
Download Bytes: The first metric targets *bulk transfer applications*, where total data volume is the primary utility. In this case, we adopt the formal absolute harm definition, as shown in Equation (1), where m represents the total bytes downloaded within our observation window (twice the duration of the short flow's lifecycle). An example is provided in Fig. 3. In Fig. 3a, the long-running Cubic flow is starved by BBRv1 short flows, resulting in an absolute harm of 0.53. Conversely, in Fig. 3b, the Cubic long flow experiences significantly less degradation from Cubic short flows (absolute harm 0.26). Consequently, the total relative harm is derived as $0.53 - 0.26 = 0.27$.



(a) Cubic vs. BBRv1 **(b) Cubic vs. Cubic**

Figure 3: 25bw-26rtt-1.0bdp: 1 Cubic long running flow competing with 3 BBRv1/Cubic short flows with harm metric Download Bytes.

Recovery Time: The second metric focuses on the *lingering effect* inflicted upon the long flow. We posit that the faster the long flow recovers its original throughput, the lower the harm. Let $t_{recover}$ denote the recovery duration after short flows complete, and let t represent the short flow lifecycle. Given our total observation window is $2t$, the harm is defined as: $Harm = \frac{t_{recover}}{2t}$. As seen in Fig. 4a, the Cubic long flow requires more time to recover after competing with BBRv3 short flows compared to Cubic short flows. This disparity results in a high total relative harm of 0.75.



(a) Cubic vs. BBRv3 **(b) Cubic vs. Cubic**

Figure 4: 175bw-70rtt-1.0bdp: 1 Cubic long running flow competing with 4 BBRv3/Cubic short flows with harm metric Recovery Time.

Although we propose these specific relative harm metrics and short flow workloads, the modularity of our system enables CCA designers to easily define their own relative harm metrics and workloads tailored to specific use cases.

4 Duration of Experiments & Convergence

In addition to providing Mahak and HarmGen with a metric to maximize, we must also provide our testbed with guidance on *when and for how long* to measure bandwidth. There is no broadly agreed-upon guidance on this challenging problem.

At the heart of the problem is the fact that CCA behavior under competition takes some time to 'warm up', and, depending on the workload, this warm-up period may or may not be an issue. Operators with fixed-size transfers are typically interested in *both* the warm-up period and the converged behavior. In settings with

web traffic, for example, many flows never exit slow start – and they do not reach a stable state in congestion avoidance. In this case, the metric of interest is typically flow completion as opposed to stable-state throughput. For these operators, simply measuring flow completion time suffices to understand the impact of a novel CCA on their workload. However, other operators with long-lived or persistent connections are more interested in the post-warmup, converged behavior. Indeed, most congestion control research aims to study algorithm behavior after convergence [4, 15, 18].

Unfortunately, there is no clear definition for how to measure convergence, and this choice can significantly affect the results. For example, in observing Fig. 9, an operator might choose to measure throughput in the period from 0-50s to measure converged behavior – but measuring this same interval over BBRv1 would capture the algorithms still during their warm-up period, inaccurately capturing the converged behavior.

Although attempts to programmatically compute a boundary between warm-up and convergence exist, they all suffer from problematic assumptions about long-term flow behavior. Due to space limitations, we leave the analysis of previous work on computing convergence to Appendix A.1. Ultimately, we find that prior solutions fail to work across the variety of observed inter-CCA interactions, including *large oscillations*. Thus, we present a novel standard-deviation-based algorithm, PACE (Point of Algorithm Convergence Estimator), to determine whether flows have converged. We say a trace has converged once the mean of the throughput's standard deviation has stabilized over time, allowing for periodic oscillations that we commonly observe in inter-CCA interactions. We describe the details of how PACE determines when and if a flow converges in the following section and provide pseudocode for PACE in Appendix A.3.

4.1 Convergence Algorithm

PACE takes an experiment with N flows and returns T, where T is a timestamp after convergence has occurred. Thus, the algorithm allows us to conclude with reasonable certainty that all flows have converged after time T. We identify a point *after* convergence, not the exact convergence time; measuring throughput, and thus harm, from any such point should yield approximately the same value.

We have two requirements for our algorithm. First, we do not want to assume that the flows will arrive at a fixed throughput that we know a priori (like equal-rate fair sharing). Second, we want our algorithm to allow for large oscillations in throughput as long as those oscillations are consistent and stable over time. (Appendix A includes an example of flows with consistent oscillations in throughput.)

To determine convergence for each experiment, we take the PCAP output of an experiment with competing flows and compute throughput over time over windows of 1 second. Then we identify the convergence time f_t of each flow independently. If PACE says that any of the flows in an experiment did not converge, we say the experiment did not converge. Alternatively, if all the flows do converge, then we say the point at which we will say convergence has occurred is the latest time t of all flows f.

We identify a point of post-convergence f_t for a flow's throughput trace by recursively applying a widely used change point detection algorithm, Ruptures-Dynp, on a rolling window of the

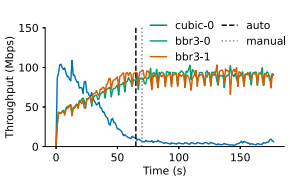


Figure 5: Manual convergence label vs. auto label from PACE.

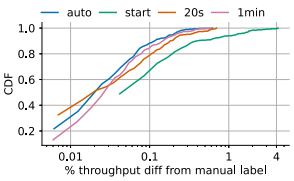


Figure 6: % diff of manual and auto label from PACE, 20s, 1min into experiment (log scale).

trace's standard deviation until it has stabilized. Ruptures-Dynp is a dynamic-programming-based method that identifies a break point, the point where the mean of the signal changes, by minimizing the sum of squared errors when approximating the signal by a piece-wise constant signal [46]. Using a change point detection subroutine, PACE can dynamically find a point in time where the standard deviation has possibly had a significant change.

This works as follows: PACE computes the standard deviation of throughput over a rolling window size of *win_size*. In the first iteration, PACE sets the possible convergence point at time s_0. Letting $s = \{s_0 \dots s_m\}$ be the sequence of rolling standard-deviation values, PACE calls the `ruptures` library [46] on s to find a single breakpoint in the series. We determine that the standard deviation has stabilized once the difference in mean of the left signal and right signal is within either an absolute or percent-based threshold. The discussion for parameter tuning these thresholds is left to Appendix A.4. If the difference is not within either threshold, we rerun Ruptures-Dynp on the subsection of the trace from the proposed change point to the end. An example of these iterations can be found in Fig. 22 in Appendix A.3.

PACE computes the rolling standard deviation over some window size *win_size*, but how do we set that size? A fixed window size is unlikely to work across different kinds of CCAs; thus, a key component of PACE is a loop over varying window sizes until it finds a size that results in convergence. We set the maximum size to half the length of the trace.

To evaluate the accuracy of PACE, we manually label a random sample of 100 experiments across various CCA combinations.[3] Fig. 5 shows an example of our labeling results. This example trace has 1 Cubic flow competing with 2 BBRv3, which we manually label as converging at 65 seconds, whereas our algorithm labels the point of post-convergence as 70 seconds. In either case, the throughput after these points is nearly identical. When comparing the average throughput from the auto-labeled vs. the manually labeled experiments across our random sample of 100 experiments, we find that our method can get the lowest percent difference in throughput from the manually labeled results. We can see in Fig. 6 that our auto labeling results in the tightest fit, with over 80% having less than a 10% difference in average throughput from our manual labels. In addition, we see that the difference in throughput between our manual labels and just ignoring the first 1 minute of the trace is also very close. Consequently, throughout this work, we run our experiments for 3 minutes and run the convergence algorithm to decide over what interval to compute relative harm. If

[3] Cubic vs. BBRv1, BBRv3, Reno, or Cubic; Reno vs. Westwood; and BBRv1 vs. BBRv3.

our algorithm says the flows converged, we compute relative harm from the point of convergence until the end of the trace, ignoring the time before our convergence point. If the algorithm says the flows do not converge, we compute harm from time 60s to the end of the trace, ignoring the first 1 minute of the experiment.

4.2 Does convergence matter?

Having developed a convergence algorithm, we now ask whether accounting for convergence is necessary at all. More specifically: does taking convergence time into consideration affect the conclusions we draw from our experiments?

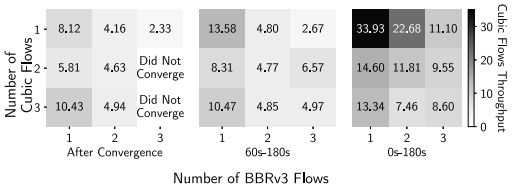


Figure 7: 200bw-160rtt-0.75bdp: Cubic throughput for BBRv3 experiments, computed after the PACE convergence point, over the final two minutes (first minute removed), and over the entire 180s trace.

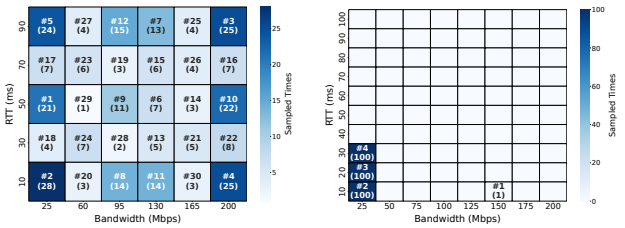
Fig. 7 compares average Cubic flows throughput for one network setting (200bw-160rtt-0.75bdp), computed three ways: after the PACE convergence point, over the final two minutes (first minute discarded), and over the full 180s trace.

Findings that do not consider convergence may be misleading. Numerous studies show that BBRv1 flows starve Cubic flows in small queues [12, 24, 50]. Without considering convergence, the throughput for 1 Cubic flow vs. 2 BBRv3 flows is 22.68 Mbps, which is a major improvement over starvation. However, when considering the throughput only after convergence, it is only 4.16 Mbps. Although any improvement over starvation is welcome, the gain is far less drastic, indicating that substantial work remains to address these issues with BBR. This is one of many such cases among the 3,500 BBRv3 vs. Cubic experiments we ran. PACE helps researchers detect when metrics shift significantly during an experiment and report results after convergence.

To check whether PACE improves on a simpler heuristic, Fig. 7 also reports a fixed cutoff that discards the first 60 seconds (the *60s–180s* panel). This cutoff is often close to PACE (4.80 vs. 4.16 Mbps for 1 Cubic vs. 2 BBRv3), but PACE still helps where flows converge after 60s (1v1: 13.58 vs. 8.12 Mbps) and flags experiments that never converge (2v3 and 3v3), which a fixed cutoff cannot.

To further motivate PACE, we measured, for every flow in our 7,000 Cubic vs. BBR experiments (3,500 each against BBRv1 and BBRv3), the percent difference in throughput between the PACE and 60s-cutoff averages. Across these 31,984 flows, 17% differ by at least 10%, 5% by at least 20%, and 0.5% by at least 50%. The full distribution and the most-affected examples appear in Appendix A.5.

We stress that we do not claim harm matters only after convergence. Because inter-CCA interactions are volatile and slow to converge, determining whether convergence has occurred is precisely what lets an evaluator decide whether they care about behavior before convergence, after it, or not at all. It is possible, of course, that CCA developers do not care about long-term behavior



(a) Long Flows Workload **(b) Short Flows Workload**

Figure 8: Mahak's sampling heatmap of BBRv3 vs. Cubic across workloads. In each block, #n = sampling sequence and (x) = number of samples.

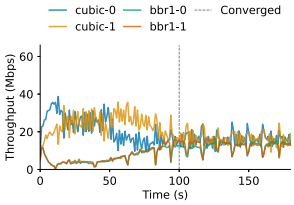


Figure 9: 60bw-70rtt-4.75bdp: 2 Cubic vs. 2 BBRv1 flows.

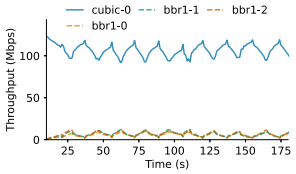


Figure 10: 130bw-10rtt-4.75bdp: 1 Cubic vs. 3 BBRv1 flows after convergence with JFI.

and rather care about flows of a specific duration (as most flows are short). In that case, results must be qualified with the duration of the flows. We cannot make broad generalizations about inter-CCA dynamics for "long-running flows" without considering convergence. Now we have an algorithm to help us decide how long we may want to run experiments if we care about making statements about long-term behavior.

5 Extending Mahak

Mahak [37] is an Active Learning (AL) framework designed to automate the performance assessment of network protocols without exhaustive testing. Employing a Gaussian Process Regressor and uncertainty sampling, it models high-dimensional performance spaces and can achieve a low prediction error (0.06 RMSE) using only ~4% of total space.

We applied our methodology to Mahak by replacing the single-CCA performance metric with our proposed *relative harm metric* and conducting experiments with long-running flows. We expanded the search space to include bandwidths of [25, 200] Mbps, RTTs of [10, 100] ms, queue sizes of [0.5, 5]× BDP, and [1, 5] alpha/beta flows. We pick this range because it is similar to the range of values found in much of the prior work, including Pazhooheshy et al. [37], with the addition of the number of flows. We describe our testbed in §7. Mahak's prediction error is low after training on 4% of the search space: 0.0456 MAE and 0.0627 RMSE, comparable to the prediction error reported in the original Mahak evaluation [37]. Appendix D shows error rates for other search space fractions. We next compare these results to using Mahak without relative harm and Mahak without the convergence algorithm.

First, we compare Mahak using relative harm to Mahak using JFI as the metric. Mahak exhibits higher prediction error when modeling JFI than it does with relative harm. We observe a mean absolute error (MAE) of 0.0456 using harm vs. 0.1360 using JFI and a root mean square error (RMSE) of 0.0627 with harm vs. 0.1608 with JFI. Furthermore, JFI can be a misleading proxy for inter-CCA fairness because it does not indicate which CCA is being harmed. As illustrated in Fig. 10, a low JFI signals that an allocation is unfair, but does not reveal which flow is harmed.

Second, we compare Mahak's results without the convergence algorithm to results with the convergence algorithm. We find discrepancies in harm predictions when not considering convergence. For example, in Fig. 9, while Cubic and BBRv1 eventually converge to a nearly fair share after 100 seconds, measurements taken without the convergence algorithm erroneously include the unsteady state where Cubic dominates the link. This results in a misleading harm value of −0.18, whereas the true steady-state harm is near zero (−0.07). This discrepancy demonstrates that without the convergence algorithm, Mahak samples imprecise data, ultimately generating untrustworthy performance maps. These results confirm that our methodology is compatible with existing automated assessment tools and correctly guides these tools compared to alternatives.

Lastly, we see how Mahak performs with the short-flow workload and metric described in §3.2. Since Mahak was not originally designed to target inter-CCA fairness, we see where it can struggle with a more complex workload. In Fig. 8, we find that for short flow workloads, it gets stuck running experiments in a small area rather than exploring more of the space, leading to predicting the harm near 0 for all the space it did not explore. In contrast, for long-flow workloads, Mahak visits many parts of the space and has low prediction error. We hypothesize that the complexities of these heterogeneous workloads are more challenging for the AL algorithm to predict.

To study short-flow workloads and demonstrate the applicability of our methodology to other tools, we develop HarmGen, a genetic algorithm-based automation tool specifically tailored to accurately and efficiently discover harmful network settings in a diverse set of inter-CCA experiments. We detail HarmGen in §6.

A recent Mahak update [38] employs Random Forest and Greedy Diversity Sampling, potentially addressing the short-flow issues we encountered. However, due to code unavailability at the point of submission, this paper relies on the original framework [37]. Our results align with prior findings, serving to demonstrate the portability of our harm metric to existing tools. We anticipate improved efficiency with the new version and plan to include it in future comparisons.

6 HarmGen: Finding Worst-Case Scenarios

In this section, we describe HarmGen, our genetic algorithm (GA)-based approach for automatically finding worst-case outcomes for CCA interactions. While alternative approaches like simulated annealing can also find optimal values of unknown functions, we found that GAs perform especially well for three reasons. First, GAs are less susceptible to getting stuck in local minima because they return a random population of results rather than converging to a single value. Second, GAs are amenable to constraints that force them to discover high-harm settings with a wide range of network parameters. This is particularly applicable in cases where high-harm scenarios may be biased towards a single network parameter like bandwidth. In this case, we can employ stratification of initialization to force our algorithm to come up with high-harm settings across different bandwidths, rather than only coming up with settings with a single bandwidth. Lastly, GAs do not require knowledge of the underlying function (or for the function to be differentiable), we need only the inputs (network settings and workload) and outputs (relative harm). Thus, we select a GA-based solution since it can identify worst-case high-harm scenarios and return a diverse set of them rather than just one.

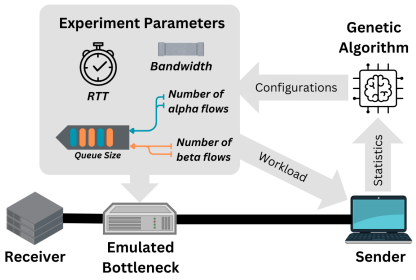


Figure 11: HarmGen architecture and testbed.

While GAs work well in solving a variety of problems, they also come with the drawback of a large volume of parameter choices. We favor the simplest option at each such choice. We do not claim our choices–uniform crossover, weighted mutation, and so on are optimal. We pick simple variants and confirm robustness via our sensitivity analysis. We primarily draw inspiration from Haupt and Haupt's popular book and chapter on implementing continuous genetic algorithms [23]. We describe the set of parametric choices: population size, mutation probability, and experiment budget in the following description of our genetic algorithm. In §6.1, we explore the impact of such choices with sensitivity analysis to demonstrate that the algorithm performs well under the selected conditions. We explain the impact of mutation probability, population size with experiment budget, and repeats in Appendix B.1, Appendix B.2, and Appendix B.3 respectively.

Testbed: To implement HarmGen, we instrumented a testbed with a simple dumbbell topology including a sender, receiver, and emulated bottleneck as shown in Fig. 11. We implemented the emulated bottleneck using the BESS [1], which allows us to support high-speed traffic while controlling the RTT by delaying packets, slowing the egress port to limit bandwidth, and setting a bottleneck queue size. The GA decides which experiments to run and passes those experiment settings, including the network settings and the mix of flows, to the emulated bottleneck. We try to run as many experiments in parallel as we can (subject to the total bandwidth available between each of the nodes which is 10GB), so we launch multiple `iperf` flows in parallel and separate individual experiments, each with its own bottleneck queue, with the settings for RTT, bandwidth, and queue size determined by the GA. All flows are run for three minutes. After running all the experiments in a given generation, HarmGen takes PCAPs at the sender to compute throughput after convergence time and then computes relative harm. We compute

relative harm as described in §3.1 and use the average throughput amongst β flows as our metric m.

Initialization: A GA is initialized with an initial random population of 60 chromosomes. Our chromosomes consist of 5 genes: bottleneck bandwidth $\in [25, 400]$ Mbps, RTT $\in [10, 320]$ ms, queue size $\in [0.25, 5]$ BDP, as well as the number of competing α and β flows $\in [1, 5]$[4]. This is essentially a vector with 5 elements. Note, this encoding works both for the long-flow and short-flow workloads described in §3. We fix the start times of all the short flows and the number of β flows to 1. Once we have an initial population, we need to compute the harm, which in our case entails running 2 or 3 experiments in our testbed. For long-flow experiments, we need to run an experiment with these network settings with the number of β flows alone and a second experiment with the number of competing β and α flows to compute relative harm. For short-flow experiments, we need to run 3 experiments as described in §3.2.

Selection: Once we have an initial population and the harm values for each chromosome, we have to decide which chromosomes will mate. There are many possible selection algorithms, another parametric choice. HarmGen uses "natural selection", selecting the top 50% of chromosomes (where the fitness is the relative harm, the higher the harm, the more fit the chromosome) for mating. Because our population size is 60, this will be 30 chromosomes selected for mating. Chromosomes are randomly shuffled into pairs.

Mating: Once we select the chromosomes we want to mate, we need to make two offspring for each pair of parents by performing crossover. [5] There are many possible crossover algorithms, another parametric choice. HarmGen uses "uniform crossover". For each gene, we flip a fair coin (50% probability) to decide whether to apply crossover to that gene. For example, we may decide we are going to crossover the bandwidth genes, b_1 and b_2, of parent 1 and parent 2. Because the variables in our genes are continuous, we use a standard blending method to combine these genes to make two offspring by picking a random value $\beta \in [0, 1]$:

$$\begin{aligned} b_{\text{child1}} &= b_1 - \beta(b_1 - b_2) \\ b_{\text{child2}} &= b_2 + \beta(b_1 - b_2) \end{aligned} \tag{4}$$

Intuitively, this will introduce new genes into the population by choosing a random value between the two parents. Each gene has an equal likelihood of being selected for crossover. We never blend across genes; bandwidth genes will only ever cross with other bandwidth genes, the same for RTT and so on. In the extremes, it is possible to crossover every gene or to crossover none of them. In the case that there is no crossover, the children are copies of the parents. There are ways to prevent this possibility, but the most basic GA implementation does not force unique children.

Mutation: Once we have two children, the next step is to perform mutation. Mutation happens with some probability, another parametric choice. We choose to perform mutation with a probability of 30%. Given our population of 30 children, from mating, this would, in expectation, mutate 9 children. For each gene within a child, we mutate it with probability 40%, so in expectation we mutate 2 genes. We deliberately choose a large mutation probability. In the mating phase, we use the values of the parents' genes. While this is good for narrowing down to a specific solution, it could cause HarmGen to get stuck in local minima if we are not introducing enough new possible chromosomes in each generation. HarmGen uses a random mutation algorithm, taking the mutated gene and assigning it a random value.

Replacement: Once we have children, we need to decide what chromosomes will "die off" and which will remain to form the new population for the next generation. HarmGen combines the parents who were selected for mating (the Top 50% from the current population) and the offspring from these children to form the new population. For the children, we compute the relative harm for these possibly new settings.

Stopping condition: HarmGen stops when running another generation would exceed the experiment budget, which we set to 300 experiments. We conduct sensitivity analysis on the experiment budget in §6.1 and show that with just this small budget of experiments, HarmGen cannot only find worst-case harmful scenarios but also a *diverse set of scenarios.*

6.1 Evaluation

While we use BBRv1 and Cubic to benchmark Mahak to adhere closer to Pazhooheshy et al.'s [37] original evaluation, for HarmGen, we stress test our system by identifying CCA pairings whose higher harm scenarios are more challenging to find. To determine which pairs of CCAs to test HarmGen, we run experiments with 300 random network settings for various pairs of CCAs. We show the CDF of relative harm for each of these pairs in Fig. 12. Each of these lines refers to β vs. α. The distribution of random experiments gives a rough estimate of the difficulty of finding poorly performing scenarios. For our evaluation, we focus on Reno vs. Westwood because the majority of the settings found from a random search have low harm, but there is a longer tail than the other pairs, suggesting there are some harmful scenarios that may be hard to find. Our metric for success for HarmGen is to find both harmful scenarios *and* a diverse set of them with a small budget of experiments.

6.1.1 Baselines To evaluate HarmGen, we compare its results with three other approaches for finding high-harm scenarios: a parameter sweep, local search, and random search.

Ideal Approach-Parameter Sweep: We use a parameter sweep as an approximation of an exhaustive search that spans the range of values that we allow the GA to search. Because the parameter space is far too large to enumerate exhaustively, we instead visit exponentially spaced points within it, for a total of 3500 experiments.

Local Search Approach-Hill Climbing: In contrast to the GA's population-based approach, the hill climbing algorithm is a simple, single-solution metaheuristic optimization algorithm. The implementation of our algorithm works as follows. First, we define a search space. In this case, the search space includes all scenarios explored by the parameter sweep. Next, we randomly select an

[4]The step size of our ranges is log scale because CCA behavior varies by factors of change. For example, there is going to be a significant difference in outcomes between doubling the bandwidth rather than increasing it by 1.

[5]Technically, there is an additional parameter: the probability of crossover. In our case, we ignore this and set the probability to 1.

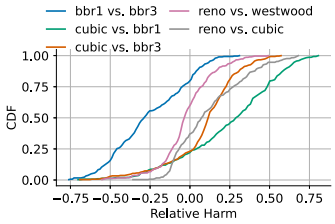


Figure 12: Relative harm of 300 random experiments.

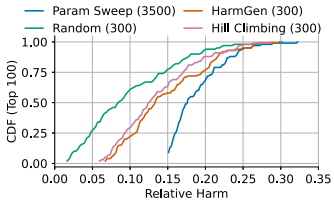


Figure 13: Comparison of top 100 relative harm values.

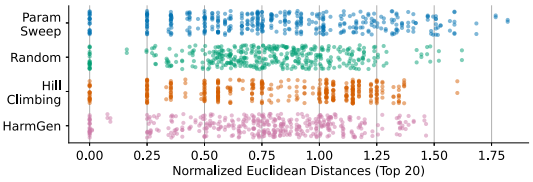


Figure 14: Comparison of the euclidean distances between the top 20 harmful scenario's vectors.

initial state from the space. Then, for each individual parameter, we pick the neighboring states above or below (i.e. if the current state is [100bw, 15rtt, 32q, 1α, 1β] and we are considering the bandwidth parameter, we test neighbors [50bw, 15rtt, 32q, 1α, 1β] and [200bw, 15rtt, 32q, 1α, 1β]). If either of the neighboring states produces a higher harm value, we transition to that state. If we iterate over all parameters (in this case, bw, rtt, q, α, β) and do not move, we restart the algorithm from a new random state until we meet our experiment budget of 300.

Naive Approach-Random Search: For random search, we simply initialize our search space to match the scenarios explored by the parameter sweep and randomly select a scenario from the set. We select 300 distinct experiments to match the 300 experiment budget for our GA.

6.1.2 Results To show that HarmGen can find high-harm scenarios, we compare the Top 100 scenarios found by HarmGen to those found by the parameter sweep, random search, and hill climbing. We show that HarmGen is better at finding harmful scenarios than random search and hill climbing in Fig. 13 where we plot the distribution of the Top 100 harm values found by each method. We denote that the parameter sweep results are from 3500 experiments, while random, hill climbing, and HarmGen are from 300 experiments. In a pairwise comparison, 98% of the Top 100 scenarios found by HarmGen have a higher harm than random search with a median percent difference of 45.4% when compared to the random harm values. When compared to hill climbing, 99% of HarmGen's Top 100 scenarios have higher harm with a median percent difference of 8.1%. Though it seems like hill climbing is comparable to HarmGen, we find that the algorithm is extremely volatile between repetitions. In 10 repetitions of hill climbing, the median percent difference ranges from 5% to 51% when compared to HarmGen. This volatility is illustrated by Fig. 33 in Appendix B.3. HarmGen is also able to find harm values on par with parameter sweep, with a median percent difference of only 27.9% lower than the parameter sweep harm values with 3500 experiments.

In addition to finding high-harm scenarios, we also want to find a diverse set of scenarios. It is possible to find high-harm scenarios that are all similar, in a small section of the search space. To show HarmGen finds a diverse set of high-harm scenarios we compare the Top 20 scenarios found by HarmGen to those found by the parameter sweep, random search, and hill climbing. We take the Top 20 harm scenarios' chromosomes and compute the Euclidean distance between them. We max-min normalize each feature of the vector, so the ranges of the values do not impact the distance. A small distance means scenarios are very similar. In Fig. 14, we see that the high harm scenarios found by HarmGen are as varied as those found by random search. This indicates HarmGen is finding a good variety of high-harm scenarios. The median Euclidean distances between the parameter sweep is 0.79, 0.77 for random search, 0.78 for HarmGen, and 0.90 for hill climbing.

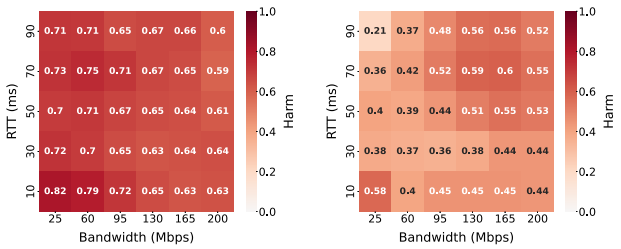
While our evaluation considers a single run of HarmGen, we provide a sensitivity analysis in Appendix B which explores the impact of experimental parameters, including mutation probability, population size, experiment budget, and repetitions on the quality of HarmGen's results. Through this analysis, we find that a mutation probability of 30% yields the highest harm scenarios. We also determine that 300 experiments reasonably balances the trade-off between finding high-harm scenarios and computation time. HarmGen is able to find the highest harm from the parameter sweep (PS Highest) within only 200 experiments, and even higher harm scenarios than PS Highest when the budget is increased to 500 experiments. Finally, we demonstrate that despite HarmGen's stochastic nature, repetitions consistently identify diverse, high-harm scenarios: in pairwise comparisons, 100% and 95% of HarmGen's top 100 settings are more harmful than those found by random search for runs 2 and 3, respectively.

7 Case Studies

Our evaluation is conducted on Vagrant-managed Ubuntu 22.04 VMs (8 cores, 64 GB RAM). We utilize specific kernels: Linux 6.13.7 [3] for BBRv3 and Linux 5.15.72 [2] for TCP Prague and DualPI2 Active Queue Management (AQM). Network parameters (bandwidth, RTT, queue size) are emulated using Linux Traffic Control (`tc`).

7.1 Cubic vs. BBRv3

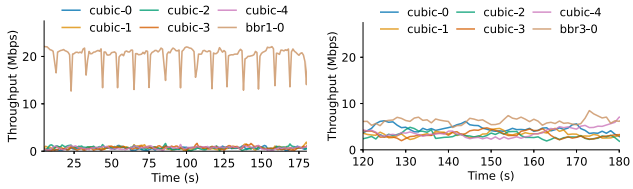
In this study, we use Mahak to evaluate harm for Cubic vs. BBRv3 for short and long running flows. We compare these results to the output of Cubic vs. BBRv1 experiments to track improvements (or lack thereof) between releases. Mahak returns a prediction of the harm across the whole search space, so we show a slice of results in two dimensions in Fig. 15. We see substantial improvement over most regions. We observe that harm done to Cubic decreases significantly between BBRv1 and BBRv3, especially for lower-RTT, lower-bandwidth sections of the search space. In the highest harm region of Cubic vs. BBRv1 experiments (25 Mbps, 10 ms RTT), we see an 81% decrease in the same region for Cubic vs. BBRv3 experiments. This improvement, however, is not uniform, as harm values in the high-RTT, high-bandwidth region remain roughly similar. Additional heatmaps of more parameters are available in Appendix E.



(a) Cubic vs. BBRv1 (b) Cubic vs. BBRv3

Figure 15: Mahak's prediction on the search space (BWxRTT). The number in each block shows the highest harm predicted in this region.

We illustrate an example of BBRv3's significant improvement in Fig. 16 for low bandwidth environments. While BBRv1 initially does very high harm to the Cubic flows in Fig. 16a (harm = 0.68), BBRv3 appears to be more fair to Cubic flows in Fig. 16b with a small harm of 0.09. We hypothesize that this improvement is due to changes introduced in BBRv2, which allow it to react to loss/congestion signals, as well as the tuning of control parameters like `cwnd_gain` and `pacing_gain` for better harm outcomes [56].



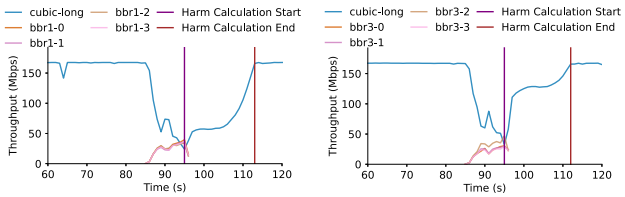
(a) Cubic vs. BBRv1 (b) Cubic vs. BBRv3

Figure 16: 25bw-90rtt-0.25bdp: 5 Cubic flows competing with 1 BBR flows after convergence. In this case, BBRv3 causes less harm to Cubic.

In addition to measuring long-running throughput harm, we investigate recovery time harm for short flows. Our experiments reveal cases where there is little improvement in the highest harm cases from Cubic vs. BBRv1 experiments. For example, we re-run one of the highest harm network settings for BBRv1 (Fig. 17a) using BBRv3 (Fig. 17b) and observe a negligible change in harm. While the long-flow experiments reveal significant improvements for BBRv3, these short-flow examples also shed light on new avenues for improvement, like recovery time for short-flow workloads.

7.2 L4S: TCP Prague vs. Cubic

To support high throughput and ultra-low latency applications, the L4S architecture [10] combines a scalable congestion control algorithm (ex: Prague), accurate Explicit Congestion Notification (AccECN), and isolation of L4S and classic flows in a dual-queue coupled AQM to ensure coexistence. A recent evaluation of L4S investigates fairness between scalable CCAs and classic flows in the custom queue [41]. Conducting such an evaluation provides an opportunity to demonstrate how our methodology extends to different queue disciplines and CCAs designed for specific kinds



(a) Cubic vs. BBRv1 (b) Cubic vs. BBRv3

Figure 17: 175bw-70rtt-1.0bdp: 1 Cubic long running flow competing with 4 BBRv3/BBRv1 short flows with the harm metric Recovery Time.

of applications. We run HarmGen and report some of the findings here.

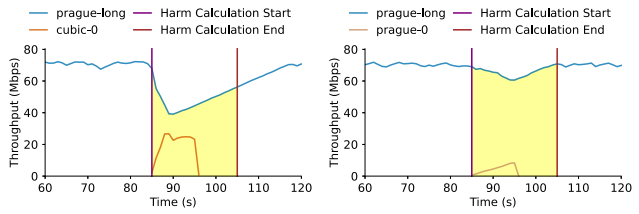
We investigate the following scenarios: suppose that a live video streaming service wants to use Prague as its CCA to obtain the benefits of ultra-low latency. We consider both the harm Prague does to Cubic and the harm Cubic does to Prague. For a fairer comparison, we configure Cubic to respond to ECN signals. It is important to note that there is flow isolation in the AQM: Prague flows will go to the L4S queue, while Cubic flows will go to the classic flows queue.

First, we explore one of the highest harm settings found by HarmGen for the harm that long flows using Prague incur due to competing with long Cubic flows (visualized in Fig. 44). Here, we ask how adding Prague flows affects existing traffic. In this example, three Prague flows get more throughput than the three Cubic flows with a relative harm of 0.26. While there is unfairness here, we do not find that this unfairness is much worse than the relative harm Cubic does to itself, and it is not nearly as harmful as the interactions we see when Cubic competes with BBR. For example, the highest harm value HarmGen found for long Prague flows was 0.26 while the highest harm for BBR was 0.81.

In Fig. 18, we highlight one of the highest harm settings found by HarmGen for the harm that short Cubic flows do to a long Prague flow. Here, we ask how well Prague flows carrying low-latency video perform alongside aggressive Cubic flows, and whether the flow isolation helps. We see that when a long Prague competes with other Prague short flows, the harm is low at 0.07. However, when competing with Cubic, Prague flows see a more significant drop in the amount of downloaded bytes during the short flows' lifetime. Further, because Prague is based on Reno, we see additive increase takes a long time to recover.

8 Related Work

There are two notable ways prior work evaluates CCA performance: 1) formal approaches that model CCA behavior 2) empirical methods that evaluate real implementations of CCAs. Formal approaches include verification-based techniques like CCAC [6] and analytical modeling-based techniques [34, 45, 50, 55] Overall, formal techniques enable a deep understanding of CCA mechanisms. However, they are limited by the fidelity of their models, which are often constrained to make modeling practical. If the CCA model and network model do not account for multiple flows or do not account for all the kinds of CCAs we want to test, the proofs will not translate well to the realistic network settings and real CCA



(a) Prague vs. Cubic **(b) Prague vs. Prague**

Figure 18: 73bw-100rtt-0.25bdp: 1 Prague long-running flow competing with 1 Cubic short flow with harm metric Download Bytes.

implementations with notable outcomes we are searching for with HarmGen and Mahak.

An alternative to operating on models of CCAs is to operate on implementations of CCAs, testing a variety of network settings using empirical methods. While some past work, such as Pantheon [52] or Prudentia [39], focuses on platforms for *manual testing*, HarmGen and Mahak are designed for automatically search the state space of possible scenarios. Other work targets more replicable methodology broadly: TriScale [26] offers statistically grounded tests for experiment convergence, and congestion control benchmarking [4] proposes standardized CCA benchmarks. Our methodology is complementary, focused on inter-CCA interaction.

ACT [43] and CCFuzz [40], use fuzzing to test CCA implementations using network simulator NS3[6]. CCFuzz uses a genetic algorithm to generate bandwidth traces that trigger certain undesirable behavior from a CCA. The main challenge with ACT and CCFuzz is guiding the search of large state spaces in their single CCA settings. In multi-flow scenarios this state space grows dramatically, making the search even more difficult and possibly inefficient and infeasible.

Ultimately, the issue with all the prior work is that none of these frameworks were designed with evaluating and proving inter-CCA interaction guarantees as a first-order goal, and extensions vary in practicality. While we are able to extend Mahak, we also design HarmGen with the specific purpose of efficiently evaluating inter-CCA interactions.

9 Conclusion

This work presents a comprehensive methodology that CCA developers and researchers can use to find harmful interactions between CCAs under competition, easing the task of finding worst-case scenarios that they can either improve or, as a recent RFC requires, declare unsafe environments for deployment [20]. We also demonstrate the efficacy of our methodology by conducting a case study investigating the evolution of BBR and the implications of partial deployment of L4S. In future work, we hope to improve HarmGen by supporting other workloads and exploring other options for its many parameters and configurations.

Ethics: This work does not raise any ethical issues.

[6] ACT uses NS3 enabled with DCE [44] to execute actual Linux kernel implementations, while CCFuzz operates on CCA implementations in NS3.

Acknowledgments

We thank our anonymous reviewers for their helpful comments and suggestions. This work was funded by NSF Awards 2007733 and 2212390.

References

[1] BESS: A Software Switch. https://github.com/NetSys/bess. (????).
[2] Linux kernel tree with L4S patches. https://github.com/L4STeam/linux/tree/testing. (????).
[3] TCP BBR v3 Release. https://github.com/google/bbr/tree/v3. (????).
[4] Soheil Abbasloo. 2023. Internet Congestion Control Benchmarking. (2023). arXiv:cs.NI/2307.10054 https://arxiv.org/abs/2307.10054
[5] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. 2020. Classic Meets Modern: a Pragmatic Learning-Based Congestion Control for the Internet. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 632âĂŞ647. https://doi.org/10.1145/3387514.3405892
[6] Venkat Arun, Mina Tahmasbi Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. 2021. Toward Formally Verifying Congestion Control Behavior. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 1âĂŞ16. https://doi.org/10.1145/3452296.3472912
[7] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical Delay-Based Congestion Control for the Internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 329–342. https://www.usenix.org/conference/nsdi18/presentation/arun
[8] Simon Bauer, Benedikt Jaeger, Fabian Helfert, Philippe Barias, and Georg Carle. 2021. On the evolution of internet flow characteristics. In *Proceedings of the 2021 Applied Networking Research Workshop (ANRW '21)*. Association for Computing Machinery, New York, NY, USA, 29–35. https://doi.org/10.1145/3472305.3472321
[9] Bob Briscoe. 2007. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM Computer Communication Review* 37, 2 (2007), 63–74.
[10] Bob Briscoe, Koen De Schepper, Marcelo Bagnulo, and Greg White. 2023. Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture. RFC 9330. (Jan. 2023). https://doi.org/10.17487/RFC9330
[11] Lloyd Brown, Ganesh Ananthanarayanan, Ethan Katz-Bassett, Arvind Krishnamurthy, Sylvia Ratnasamy, Michael Schapira, and Scott Shenker. 2020. On the Future of Congestion Control for the Public Internet. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 30–37. https://doi.org/10.1145/3422604.3425939
[12] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR Congestion Control. In *Presentation in ICCRG at IETF 97th meeting*. https://www.ietf.org/proceedings/97/slides/slides-97-iccrg-bbr-congestion-control-02.pdf
[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. 2023. BBRv3: Algorithm Bug Fixes and Public Internet Deployment. https://datatracker.ietf.org/meeting/117/materials/slides-117-ccwg-bbrv3-algorithm-bug-fixes-and-public-internet-deployment-00. (26 July 2023).
[14] Neal Cardwell, Ian Swett, and Joseph Beshay. 2025. BBR Congestion Control Draft. https://datatracker.ietf.org/meeting/124/materials/slides-124-ccwg-bbr-update-00. (1 November 2025).
[15] Dah-Ming Chiu and Raj Jain. 1989. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Comput. Netw. ISDN Syst.* 17, 1 (June 1989), 1–14. https://doi.org/10.1016/0169-7552(89)90019-6
[16] Cloudflare. 2024. Cloudflare Radar. https://radar.cloudflare.com/. (2024).
[17] Mo Dong, Qingxi Li, Doron Zarchy, P. Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, Berkeley, CA, USA, 395–408. http://dl.acm.org/citation.cfm?id=2789770.2789798
[18] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 343–356. https://www.usenix.org/conference/nsdi18/presentation/dong
[19] Rebecca Drucker, Gauri Baraskar, Aruna Balasubramanian, and Anshul Gandhi. 2024. BBR vs. BBRv2: A Performance Evaluation. In *2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. 379–387. https://doi.org/10.1109/COMSNETS59351.2024.10427175
[20] Martin Duke and Gorry Fairhurst. 2025. Specifying New Congestion Control Algorithms. RFC 9743. (March 2025). https://doi.org/10.17487/RFC9743
[21] Sally Floyd. 1991. Connections with Multiple Congested Gateways in Packet-switched Networks Part 1: One-way Traffic. *SIGCOMM Comput. Commun. Rev.* 21, 5 (Oct. 1991), 30–47. https://doi.org/10.1145/122431.122434

[22] Sally Floyd and Kevin Fall. 1999. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking* 7, 4 (1999), 458–472. https://doi.org/10.1109/90.793002

[23] Randy L Haupt and Sue Ellen Haupt. 2004. *Practical genetic algorithms*. John Wiley & Sons.

[24] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 1–10.

[25] Safiqul Islam, Kristian Hiorth, Carsten Griwodz, and Michael Welzl. 2022. Is it really necessary to go beyond a fairness metric for next-generation congestion control?. In *Proceedings of the 2022 Applied Networking Research Workshop (ANRW '22)*. Association for Computing Machinery, New York, NY, USA, Article 1, 7 pages. https://doi.org/10.1145/3547115.3547192

[26] Romain Jacob, Marco Zimmerling, Carlo Alberto Boano, Laurent Vanbever, and Lothar Thiele. 2021. [Tool] Designing Replicable Networking Experiments With Triscale. *Journal of Systems Research* 1, 1 (Sept. 2021). https://doi.org/10.5070/SR31155408

[27] Piotr Jurkiewicz, Grzegorz Rzym, and Piotr Borylo. 2018. Flow Length and Size Distributions in Campus Internet Traffic. arXiv 1809.03486v2. (Sept. 2018).

[28] F P Kelly, A K Maulloo, and D K H Tan. 1998. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society* 49, 3 (01 Mar 1998), 237–252. https://doi.org/10.1057/palgrave.jors.2600523

[29] Ike Kunze, Jan RÃijth, and Oliver Hohlfeld. 2020. Congestion Control in the WildâĂŤInvestigating Content Provider Fairness. *IEEE Transactions on Network and Service Management* 17, 2 (2020), 1224–1238. https://doi.org/10.1109/TNSM.2019.2962607

[30] Jiahui Li, Han Qi, Ruyi Yao, Jialin Wei, Ruoshi Sun, Zixuan Chen, Sen Liu, and Yang Xu. 2025. CClinguist: An Expert-Free Framework for Future-Compatible Congestion Control Algorithm Identification. In *Proceedings of the ACM SIGCOMM 2025 Conference (SIGCOMM '25)*. Association for Computing Machinery, New York, NY, USA, 279âĂŞ295. https://doi.org/10.1145/3718958.3750485

[31] Ayush Mishra and Ben Leong. 2023. Containing the Cambrian Explosion in QUIC Congestion Control. In *Proceedings of the 2023 ACM on Internet Measurement Conference (IMC '23)*. Association for Computing Machinery, New York, NY, USA, 526–539. https://doi.org/10.1145/3618257.3624811

[32] Ayush Mishra, Lakshay Rastogi, Raj Joshi, and Ben Leong. 2024. Keeping an Eye on Congestion Control in the Wild with Nebby. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 136âĂŞ150. https://doi.org/10.1145/3651890.3672223

[33] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. 2019. The Great Internet TCP Congestion Control Census. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 3, Article 45 (dec 2019), 24 pages. https://doi.org/10.1145/3366693

[34] Ayush Mishra, Wee Han Tiu, and Ben Leong. 2022. Are We Heading towards a BBR-Dominant Internet?. In *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22)*. Association for Computing Machinery, New York, NY, USA, 538–550. https://doi.org/10.1145/3517745.3561429

[35] D. Nace and M. Pioro. 2008. Max-min Fairness and its Applications to Routing and Load-Balancing in Communication Networks: a Tutorial. *IEEE Communications Surveys & Tutorials* 10, 4 (Fourth 2008), 5–17. https://doi.org/10.1109/SURV.2008.080403

[36] Lorenzo Pappone, Alessio Sacco, and Flavio Esposito. 2025. Mutant: learning congestion control from existing protocols via online reinforcement learning. In *Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '25)*. USENIX Association, USA, Article 80, 16 pages.

[37] Parsa Pazhooheshy, Soheil Abbasloo, and Yashar Ganjali. 2023. Harnessing ML For Network Protocol Assessment: A Congestion Control Use Case. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets '23)*. Association for Computing Machinery, New York, NY, USA, 213–219. https://doi.org/10.1145/3626111.3628182

[38] Parsa Pazhooheshy, Soheil Abbasloo, and Yashar Ganjali. 2025. Mahak: An Automated and Efficient Assessment Framework for Internet Control Algorithms. In *2025 IEEE 33rd International Conference on Network Protocols (ICNP)*. IEEE, 1–13.

[39] Adithya Abraham Philip, Rukshani Athapathu, Ranysha Ware, Fabian Francis Mkocheko, Alexis Schlomer, Mengrou Shou, Zili Meng, Srinivasan Seshan, and Justine Sherry. 2024. Prudentia: Findings of an Internet Fairness Watchdog. In *Proceedings of the ACM SIGCOMM 2024 Conference (SIGCOMM '24)*. Association for Computing Machinery, Sydney, NSW, Australia, 12.

[40] Devdeep Ray and Srinivasan Seshan. 2022. CC-Fuzz: Genetic Algorithm-Based Fuzzing for Stress Testing Congestion Control Algorithms. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks (HotNets '22)*. Association for Computing Machinery, New York, NY, USA, 31âĂŞ37. https://doi.org/10.1145/3563766.3564088

[41] Fatih Berkay Sarpkaya, Fraida Fund, and Shivendra Panwar. 2025. To Adopt or Not to Adopt L4S-Compatible Congestion Control? Understanding Performance in a Partial L4S Deployment. In *Passive and Active Measurement: 26th International Conference, PAM 2025, Virtual Event, March 10âĂŞ12, 2025, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 217âĂŞ246. https://doi.org/10.1007/978-3-031-85960-1_10

[42] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. 2018. Towards a Deeper Understanding of TCP BBR Congestion Control. In *IFIP Networking 2018*. Zurich, Switzerland.

[43] Wei Sun, Lisong Xu, Sebastian Elbaum, and Di Zhao. 2019. Model-Agnostic and Efficient Exploration of Numerical State Space of Real-World TCP Congestion Control Implementations. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 719–734. https://www.usenix.org/conference/nsdi19/presentation/sun

[44] Hajime Tazaki, Frédéric Urbani, Emilio Mancini, Mathieu Lacage, Daniel Camara, Thierry Turletti, and Walid Dabbous. 2013. Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments. In *The 9th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. Santa Barbara, United States. https://inria.hal.science/hal-00880870

[45] Pratiksha Thaker, Matei Zaharia, and Tatsunori Hashimoto. 2021. Don't Hate the Player, Hate the Game: Safety and Utility in Multi-Agent Congestion Control. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21)*. Association for Computing Machinery, New York, NY, USA, 140âĂŞ146. https://doi.org/10.1145/3484266.3487392

[46] Charles Truong, Laurent Oudre, and Nicolas Vayatis. 2020. Selective review of offline change point detection methods. *Signal Processing* 167 (2020), 107299. https://doi.org/10.1016/j.sigpro.2019.107299

[47] David Tuber. 2023. Measuring network quality to better understand the end-user experience. (2023). https://blog.cloudflare.com/aim-database-for-internet-quality

[48] Belma Turkovic, Fernando A Kuipers, and Steve Uhlig. 2019. Fifty Shades of Congestion Control: A Performance and Interactions Evaluation. *arXiv preprint arXiv:1903.03852* (2019).

[49] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. 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 (HotNets '19)*. Association for Computing Machinery, New York, NY, USA, 17âĂŞ24. https://doi.org/10.1145/3365609.3365855

[50] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Modeling BBR's Interactions with Loss-Based Congestion Control. In *Proceedings of the Internet Measurement Conference (IMC '19)*. ACM, New York, NY, USA, 137–143. https://doi.org/10.1145/3355369.3355604

[51] Ranysha Ware, Adithya Abraham Philip, Nicholas Hungria, Yash Kothari, Justine Sherry, and Srinivasan Seshan. 2024. CCAnalyzer: An Efficient and Nearly-Passive Congestion Control Classifier. In *Proceedings of the ACM SIGCOMM 2024 Conference (SIGCOMM '24)*. Association for Computing Machinery, Sydney, NSW, Australia, 12. https://doi.org/10.1145/3651890.3672255

[52] Francis Y Yan, Jestin Ma, Greg Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. *Measurement at http://pantheon. stanford. edu/result/1622* (2018).

[53] Chen-Yu Yen, Soheil Abbasloo, and H. Jonathan Chao. 2023. Computers Can Learn from the Heuristic Designs and Master Internet Congestion Control. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 255âĂŞ274. https://doi.org/10.1145/3603269.3604838

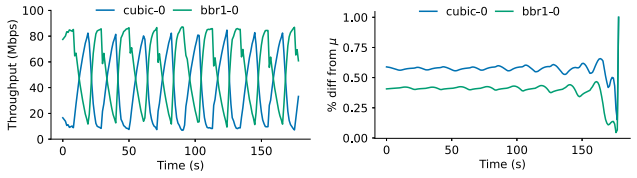
[54] Adrian Zapletal and Fernando Kuipers. 2023. Slowdown as a Metric for Congestion Control Fairness. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks (HotNets '23)*. Association for Computing Machinery, New York, NY, USA, 205–212. https://doi.org/10.1145/3626111.3628185

[55] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. 2017. An Axiomatic Approach to Congestion Control. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks, Palo Alto, CA, USA, HotNets 2017, November 30 - December 01, 2017*. 115–121. https://doi.org/10.1145/3152434.3152445

[56] Danesh Zeynali, Emilia N. Weyulu, Seifeddine Fathalli, Balakrishnan Chandrasekaran, and Anja Feldmann. 2024. Promises and Potential of BBRv3. In *Passive and Active Measurement: 25th International Conference, PAM 2024, Virtual Event, March 11âĂŞ13, 2024, Proceedings, Part II*. Springer-Verlag, Berlin, Heidelberg, 249–272. https://doi.org/10.1007/978-3-031-56252-5_12

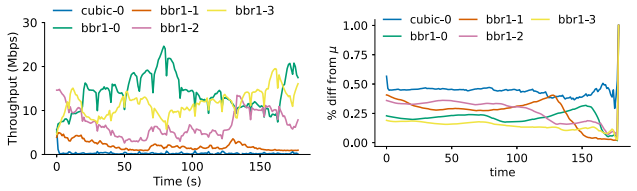
[57] Yin Zhang, Lee Breslau, Vern Paxson, and Scott Shenker. 2002. On the Characteristics and Origins of Internet Flow Rates. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '02)*. ACM, New York, NY, USA, 309–322. https://doi.org/10.1145/633025.633055

Appendices are supporting material that has not been peer-reviewed.



(a) Throughput trace (b) % difference from average

Figure 19: 100bw-10rtt-3.0bdp: 1 Cubic flow competing with 1 BBR flow where there are large, but consistent oscillations.



(a) Throughput trace (b) % difference from average

Figure 20: 50bw-40rtt-0.75bdp: 1 Cubic flow competing with 4 BBR flows where there is no convergence.

A Convergence algorithm

A.1 Related Work

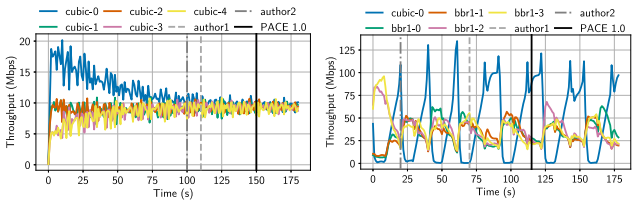
For example, the PCC Vivace [18] work assumes that flows will eventually reach a stable state in which all connections reach a near-to-fair bandwidth outcome. Hence, the work defines convergence time as the time it takes for all the flows to be within 25% of equal fair share. However, empirically, not all algorithms in all settings are guaranteed to converge to a near-fair state, and indeed, these scenarios of severe unfairness are those that we are *most* interested in uncovering! Fig. 1b illustrates one such scenario where outcomes are far from fair, and hence the algorithm above would fail to find a converged state to measure.

Both the PCC Vivace [18] and another definition provided by the 'Axiomatizing Congestion Control' [55] (referred to as 'Axiomatizing' going forward)[7] also make the faulty assumption that bandwidth allocations remain within some range of a 'fixed' or stable point, when in reality, some CCAs' 'converged' state consists of dramatic medium or long-term oscillations [48]. Fig. 19a shows an example from our experiments where we see these large oscillations. Notably, while these oscillations are large, they are consistent in their magnitude and frequency. It does make sense to say that the behavior has converged – so long as we take measurements over a long enough timescale to capture multiple iterations of the oscillatory behavior.

A.2 Convergence Algorithm Example

Fig. 22 illustrates an example of PACE's iterations for finding the point of convergence for the flow labelled cubic-0 in Fig. 21b with a window size of 55, so each point is the standard deviation for that

[7]The definition provided by the Axiomatizing work is as follows: *"We say that a congestion-control protocol P is α-convergent, for $\alpha \in [0, 1]$, if there is a configuration of window sizes $(x_1^*, \ldots, x_n^*) \in [0, M]^n$ and time step T such that for any $t > T$ and sender $i \in N$, $\alpha x_i^* \leq x_i^{(t)} \leq (2-\alpha)x_i^*$ (e.g., $\alpha = 0.9$ means that from some point onwards the window sizes are within 10% from a fixed point.)"*



(a) Cubic vs. Cubic, 50bw-160rtt-3.0bdp (b) Cubic vs. BBRv1, 200bw-10rtt-3.0bdp,cubic-bbr1

Figure 21: Examples of PACE convergence algorithm.

point in the rolling-window graph ($s_t = \sigma(f_{t+1} \ldots f_{t+win_size})$ up until $t + win_size = n$ where n is the duration of the flow).

A.3 Algorithm Description

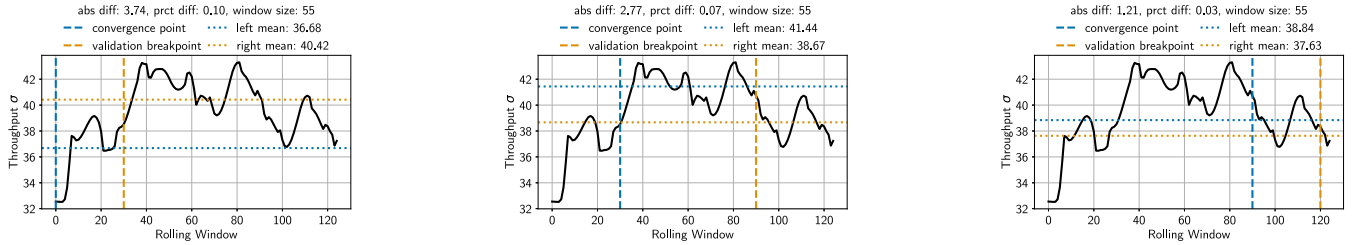
Algorithm 1 Convergence

Input:
f - flow's throughput trace
$window_size$ - size of rolling window for std dev
pct_thresh - threshold for percent difference between left and right means of std. deviation
abs_thresh - threshold for absolute difference between left and right means of std. deviation
Output:
t - time after convergence has occurred

```
procedure RecursiveBreakpoint
    Generate rolling windows of window_size
    Calc std dev of each window to generate series s
    abs_diff = inf
    pct_diff = inf
    breakpoint = 0
    while abs_diff > abs_thresh or pct_diff > pct_thresh do
        s = s[breakpoint :]
        breakpoint = call ruptures on s to find breakpoint
        if can no longer segment s then
            return inf
        end if
        l_mean = mean(s[: breakpoint])
        r_mean = mean(s[breakpoint :])
        abs_diff = abs(l_mean − r_mean)
        pct_diff = abs(l_mean − r_mean)/r_mean
    end while
    return breakpoint
end procedure
```

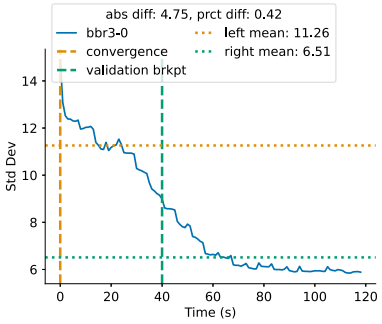
An example of how the convergence algorithm works over iterations is highlighted in Fig. 23. These are illustrations of each of the iterations of the algorithm for finding the point of convergence for the flow labelled bbr3-0 in Fig. 5.

First, the algorithm computes the standard deviation over time over a rolling window. In the first iteration the algorithm starts by setting the convergence time to time 0. Then it calls `ruptures` to find a change point between our current possible convergence time (time 0) and the end of the standard deviation trace. This change point is labelled the "validation breakpoint" in Fig. 23. The

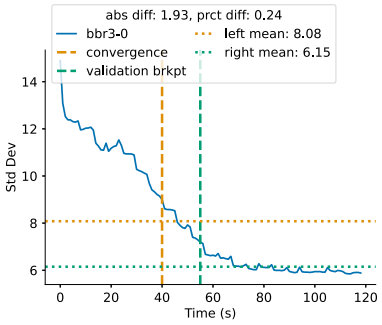


(a) First iteration **(b) Second iteration** **(c) Third iteration**

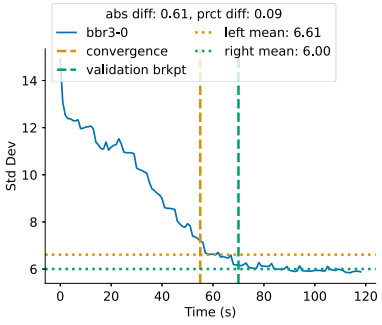
Figure 22: Convergence algorithm output over iterations before deciding the flow has converged for the cubic-0 flow in Fig. 21b.



(a) First iteration



(b) Second iteration



(c) Third iteration

Figure 23: Convergence algorithm output over iterations before deciding the flow has converged for the bbr3-0 flow in Fig. 5.

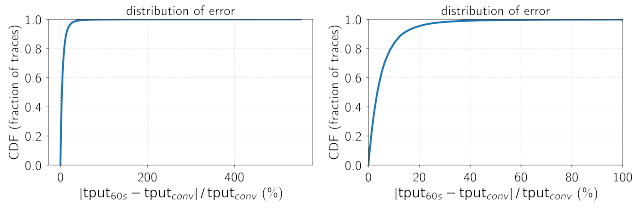
algorithm then computes the "left mean" of the standard deviation of the trace from where the convergence point is to the validation breakpoint, and the "right mean" of the standard deviation trace from the validation breakpoint to the end of the trace. In the first iteration example in Fig. 23a, the validation breakpoint is at 40 seconds, and the left mean from 0s-40s of the standard deviation is 11.26 while the right mean from 40s-120s is 6.51. To determine if the convergence point is the point our algorithm decides convergence has already occurred, we compare the absolute difference between the left mean (4.75) and percent difference (0.42) to a threshold. If at least one of the thresholds is met, the algorithm says the flow has already converged at this point. In this case neither of the thresholds are met, so the algorithm now goes to the next iteration. We repeat this process two more times as shown in Fig. 23b and Fig. 23c until either threshold is met and the convergence time of 70s is returned.

A.4 Parameter Selection

As shown in Algorithm 1 there are several parameters that we need to tune including: the rolling window size for the standard deviation trace and the thresholds for the absolute and percent difference between the standard deviation left mean and right mean. In order to choose our parameter thresholds, we hand-labeled 100 experiments for the convergence point and calculated the average throughput after these traces. We then used our algorithm to label traces using various *abs_thresh* and *pct_thresh* thresholds. Then, out of the settings that resulted in over 90% agreement on which flows converged, we picked the settings that showed the smallest mean difference in percentage from the average threshold selected by our manual labeling. These settings ended up being an *abs_thresh* and *pct_thresh* of 0.2 and 0.2 respectively. To pick the *window_size*, we simply fixed the *abs_thresh* and *pct_thresh* and varied the window size from 30-100 increasing in increments of 10 and picked the result that had the smallest mean difference from the average threshold selected by our manual labeling. This resulted in an optimal window size of 60.

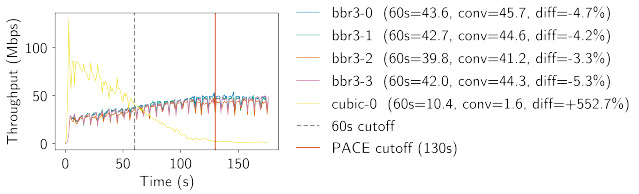
A.5 PACE Motivation

We show the distribution of throughput differences when calculating the average throughput starting from the PACE and 60 second cutoff in Figure 24. We provide a truncated and untruncated version for readability. We evaluate 31,984 traces from Cubic vs. BBRv1 and Cubic vs. BBRv3 experiments. We find that across these traces, 17% have a difference of at least 10%, 5% have a difference of at least 20%, and 0.5% have a difference of at least 50%. While the 60-second cutoff yields a small percent difference for most traces,

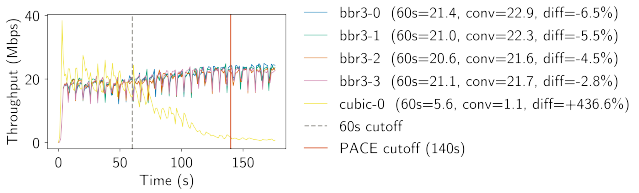


(a) Not truncated **(b) X-axis truncated to 100%**

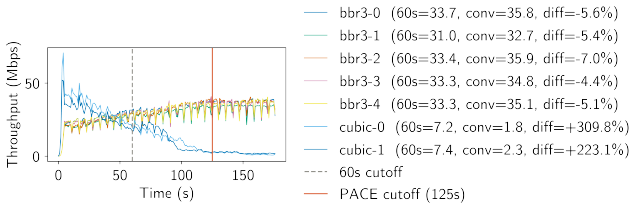
Figure 24: Throughput difference when taking average calculated using PACE vs. 60s cutoff to end.



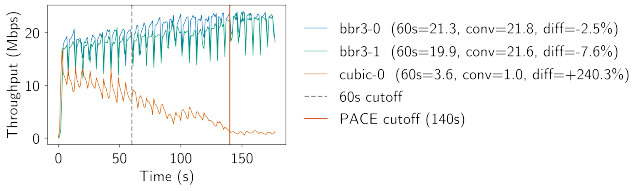
(a) 200bw-320rtt-0.4bdp: cubic-0 flow with 552.7% diff



(b) 100bw-320rtt-0.75bdp: cubic-0 flow with 436.6% diff



(c) 200bw-320rtt-0.4bdp: cubic-0 flow with 309.8% diff

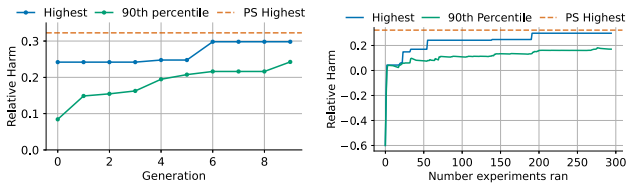


(d) 50bw-320rtt-0.75bdp: cubic-0 flow with 240.3% diff

Figure 25: Top 4 traces with highest difference in throughput when taking average from PACE and 60s cutoff.

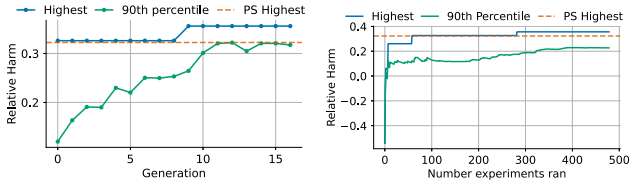
we find that for many traces that converge after 60 seconds, the differences can be quite large, reaching as much as 552.7% (Figure 25a). We show traces with the top 4 highest percent differences in Figure 25. Notice that for these traces, the large discrepancy comes from the fact that the flows do not stabilize until well after the 60-second cutoff.

B Sensitivity Analysis of HarmGen



(a) Improvement per generation **(b) Improvement per experiment**

Figure 26: Each generation what the highest harm value HarmGen has found so far improves as well as the 90th percentile compared to the highest harm value found by the param sweep (PS Highest)



(a) Improvement per generation **(b) Improvement per experiment**

Figure 27: Impact of experiment budget of 500 experiments

B.1 Impact of mutation probability

The mutation probability determines how many random genes HarmGen will be introduced in each generation. To explore the impact of different mutation probabilities, we vary the mutation probability from 10% to 50%. Fig. 28 shows the relative harm for the Top 100 scenarios found using each of these probabilities compared to the parameter sweep and random search. We see that a mutation probability of 30% is able to find higher harm scenarios than other probabilities.

B.2 Impact of population size and experiment budget

HarmGen can find high-harm scenarios with a budget of only 300 experiments. In Fig. 26 we highlight how the quality of the solutions found by the GA improves over generations and as it runs more experiments. We compare the highest harm value as well as the 90th percentile found by HarmGen after each generation in Fig. 26a, to the highest harm value found by the parameter sweep (PS Highest). After 6 generations and running only 200 experiments, HarmGen can find harm values almost as high as the parameter sweep which was 3500 experiments. With HarmGen, CCA developers can find these worst-case scenarios without having to run many experiments.

Does increasing the budget of experiments improve HarmGen's results? To see the impact of the experiment budget on the results, we run HarmGen with a budget of 500 experiments. Fig. 27 shows that with an increased budget we can find scenarios with even worse harm than those found by the parameter sweep after 8 generations. The impact of population size and experiment budget are tightly connected. We need a large enough population size to ensure diversity in the offspring, but if we have too large of a population size and not a large enough budget then we may not have enough

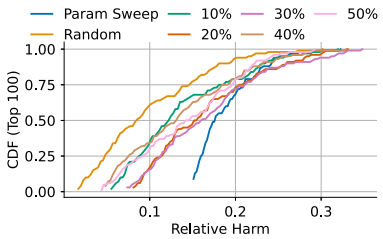


Figure 28: Impact of mutation probability on top 100

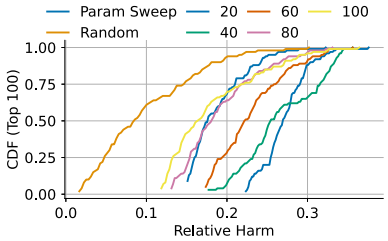


Figure 29: Impact of population size on top 100

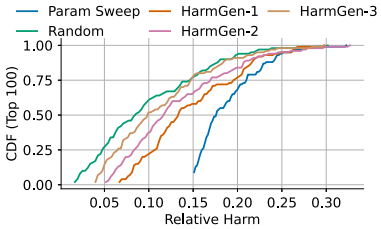
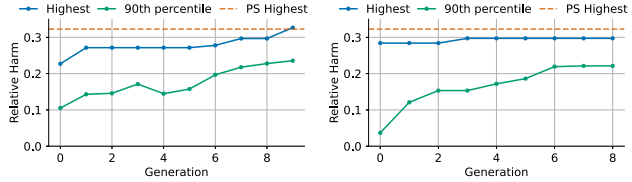


Figure 30: Impact of repeats on top 100



(a) Improvement per generation (run 2) **(b) Improvement per generation (run 3)**

Figure 31: Each generation what the highest harm value HarmGen has found so far improves as well as the 90th percentile for repeated runs. We see similar results as Fig. 26a.

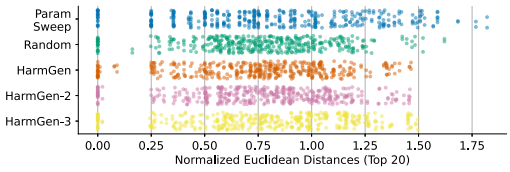


Figure 32: Comparison of the Euclidean distances between the top 20 harmful scenario's vectors for repeated runs. We see similar results across runs.

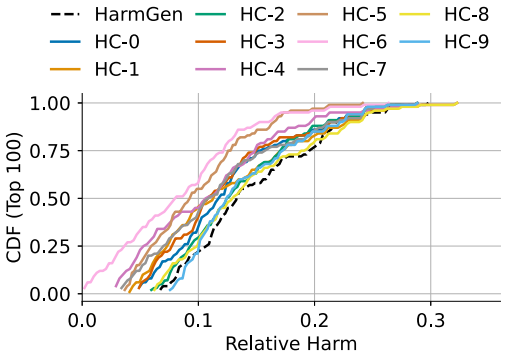


Figure 33: Impact of repeats on top 100 scenarios from hill climbing algorithm

generations for an effective search. For example, if the population size is 100 and the experiment budget size is 300, then there are only 3 generations for the search.

Another metric to consider is also the cost of running more experiments. As mentioned in our description of the testbed, we want to run as many experiments in parallel as our testbed can support. We are only running at most 20 experiments in parallel if the population size is 20 instead of possibly 100 when the population size is 100. The experiment budget in Fig. 29 is 500 experiments with varying population sizes. It seems a population size of 20 or 40 experiments works best, but these searches take much longer when the population size is 100. A population size of 20 took our testbed twice as long (7 hours vs. 14 hours) to run as a population size of 100. We find a population size of 60 is a good balance between finding high-harm scenarios and efficiency.

B.3 Impact of repeats

We repeat running HarmGen two more times to see if we get similar results when re-running the GA. Fig. 30 compares the Top 100 harm found across all three runs of the HarmGen to the parameter sweep and random search. While they vary slightly in the quality of top solutions, in all cases the results are better than random search. In the additional two repeats, 100% and 95% of the Top 100 are more harmful than random search. These same repeats find scenarios that are on average 31.6% and 19.8% more harmful than the values discovered by random search. In addition, we still get the same improvements in the quality of harmful scenarios over generations for the repeated runs of HarmGen (Fig. 31) as well as a variety of those scenarios (Fig. 32). For these repeats, the median Euclidean distances are 0.8 and 0.83, which is even better than the parameter (0.79) sweep and random search (0.77).

The top scenarios found by each run of HarmGen are different. This could be considered a flaw of HarmGen but is actually an advantage. This indicates that there are local maxima in the harm function and with each run of HarmGen it can find those settings. If HarmGen converged to the exact same worst-case settings every run, this would indicate that there is not enough randomness being introduced over generations. In practice, CCA developers could use HarmGen in an iterative process, run HarmGen to find high-harm scenarios, investigate those and possibly fix them, and then run HarmGen again and so on.

C Investigating BBRv3 vs. Cubic Results

In §7 we conducted a case study to demonstrate that we can use HarmGen for the task of comparing how evolutions in BBR may have improved or worsened interactions with Cubic. While the goal of this work was not to solve BBRv3's harmful interactions with Cubic, we do discuss in this section what the data we collected from these experiments shows about how these interactions are

causing BBRv3 to be more harmful than BBRv1. Aside from just capturing PCAPs at the sender and receiver to compute throughput and harm, we also capture queue occupancy data, including every time a packet is enqueued, dequeued, and dropped by our testbed, and socket statistics at the sender so we can see the internal variables of the CCA during the duration of our experiments. Based on conversations with Google's BBR team, we first seek to confirm if we see what they indicated was the issue in the cases where BBRv3 is more harmful to Cubic than BBRv1. One of the core changes from BBRv3 to BBRv1 is in the way BBRv3 probes for bandwidth. BBRv3 adds additional states to `ProbeBW`, with changes in `pacing_gain`, which is an internal variable that sets how many additional packets BBR will try to put into the network based on its current BW estimate. We notice that there are fewer periods of reductions in `pacing_gain` from BBRv1 to BBRv3, which we believe is one of the reasons that BBRv3 is more harmful to Cubic. However, this needs further investigation to confirm. For a setting where BBRv3 does worse, we see the average time between packet losses drops significantly.

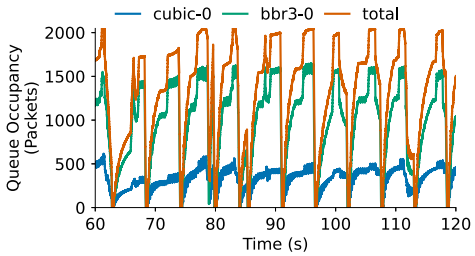


Figure 34: 111bw-173rtt-1.27bdp: BBRv3 more harmful than BBRv1: Cubic vs. BBRv3 queue occupancy

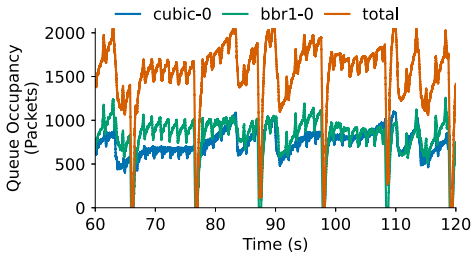
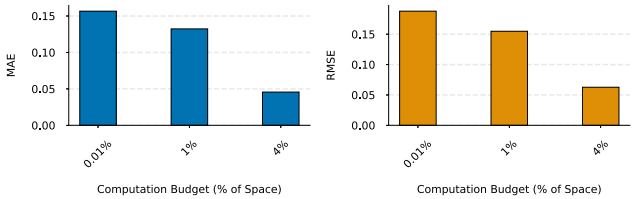


Figure 35: 111bw-173rtt-1.27bpd: BBRv3 is more harmful than BBRv1: Cubic vs. BBRv1 queue occupancy

D Mahak Prediction Error Across Computational Budgets

This section shows how Mahak's prediction error changes as the computational budget grows, measured for Cubic vs. BBRv1 long-running flows using our relative harm metric. We vary the budget across 0.01%, 1%, and 4% of the total search space and report both MAE and RMSE at each point.

As shown in Fig. 36, both MAE and RMSE decrease substantially as the budget increases from 0.01% to 4%. The error values we observe at the 4% budget are comparable to those reported in the original Mahak evaluation [37], indicating that, in the relatively simple workload of long flows competing after convergence, substituting relative harm for long flows in place of the original single-CCA performance metric does not introduce additional estimation difficulty for the underlying Gaussian Process Regressor.



(a) MAE **(b) RMSE**

Figure 36: Mahak's MAE/RMSE on Cubic vs. BBRv1 long-running flows across varying computational budget.

E Cubic vs. BBRv1 and Cubic vs. BBRv3 w/ Mahak

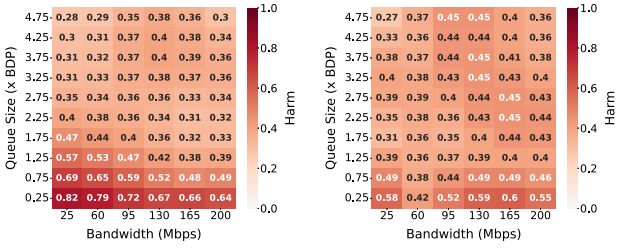
Across all parameter dimensions, BBRv3 consistently reduces harm to Cubic compared to BBRv1, but the improvement is non-uniform.

For queue size (Fig. 37), BBRv1 harm is strongly correlated with smaller queues, reaching up to 0.82 at 0.25×BDP, while BBRv3 partially mitigates this sensitivity, though harm remains elevated (up to 0.60) at small queue sizes.

For the number of competing α (BBR) flows (Fig. 38), under BBRv1, harm generally decreases as more BBR flows enter the network, suggesting that additional BBRv1 flows compete with each other and collectively exert less pressure on Cubic flows; BBRv3, by contrast, exhibits a more complex, non-monotonic pattern across α flow counts, indicating that BBRv3's inter-flow dynamics with Cubic are less predictable than BBRv1's.

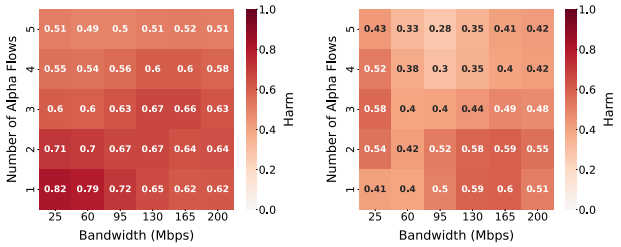
For the number of β (Cubic) flows (Fig. 39), harm generally increases as more Cubic flows compete, under both BBRv1 and BBRv3, suggesting that a larger Cubic cohort does not collectively defend its share but rather amplifies the harm caused by competing BBR flows.

Overall, these heatmaps confirm that while BBRv3 represents a meaningful step forward, no single network parameter reliably drives harm to zero, and worst-case settings persist across all dimensions tested.



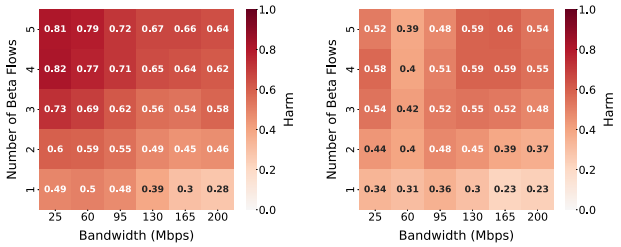
(a) Cubic vs. BBRv1 **(b) Cubic vs. BBRv3**

Figure 37: Mahak's prediction on the search space. (Queue Size)



(a) Cubic vs. BBRv1 (b) Cubic vs. BBRv3

Figure 38: Mahak's prediction on the search space. (Number of Alpha Flows)



(a) Cubic vs. BBRv1 (b) Cubic vs. BBRv3

Figure 39: Mahak's prediction on the search space. (Number of Beta Flows)

F Harm vs. JFI w/ Mahak

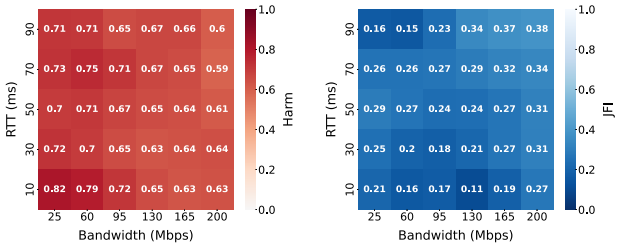
The side-by-side Mahak predictions using harm versus JFI (Figs. 40–43) further validate the argument in §3 that JFI is a poor metric for evaluating inter-CCA fairness.

While the harm heatmaps reveal consistent, interpretable structure—harm increases at low RTT, small queues, few BBR flows and more Cubic flows—the JFI heatmaps exhibit irregular, non-monotonic patterns that are difficult to interpret and do not align with the directional trends visible in the harm maps. Critically, JFI conflates scenarios where Cubic is harmed by BBR with scenarios where BBR underperforms relative to Cubic, assigning similar values to qualitatively opposite outcomes. For instance, regions of the parameter space that have high harm (BBR starving Cubic) appear indistinguishable in JFI from regions of moderate, symmetric sharing.

This confirms that JFI lacks the directionality necessary to guide CCA developers toward actionable fixes: a developer optimizing for JFI would not know who is being harmed or in which direction to adjust their algorithm. Harm, by contrast, provides a signed, normalized signal that directly reflects the degree to which a novel CCA degrades the performance of incumbent flows, making it a more reliable objective for automated search tools like Mahak and HarmGen.

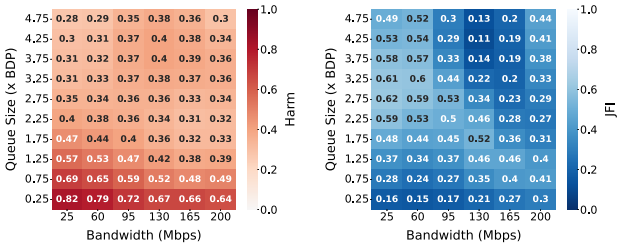
G A Worst-Case Harm Scenario for TCP Prague vs. Cubic

Fig. 44 illustrates one of the highest-harm scenarios found by HarmGen for long Prague flows competing with long Cubic flows, at a setting of 200 Mbps bandwidth, 91 ms RTT, and a queue size of 1xBDP. In this scenario, three Prague flows obtain more throughput than four competing Cubic flows, resulting in a relative harm of 0.26.



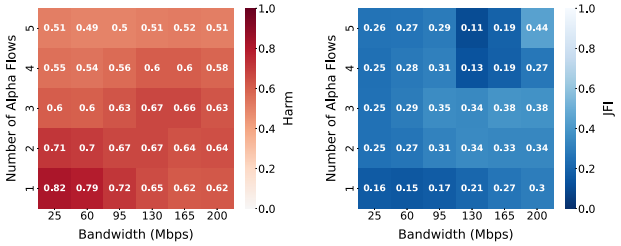
(a) Harm (b) JFI

Figure 40: Mahak's prediction on the search space (RTT).



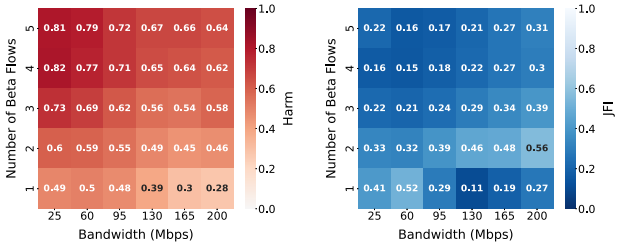
(a) Harm (b) JFI

Figure 41: Mahak's prediction on the search space. (Queue Size)



(a) Harm (b) JFI

Figure 42: Mahak's prediction on the search space. (Number of Alpha Flows)



(a) Harm (b) JFI

Figure 43: Mahak's prediction on the search space. (Number of Beta Flows)

While this indicates some unfairness, two important observations temper concern. First, the magnitude of harm is substantially lower than what is observed when Cubic competes with BBR (maximum harm of 0.81), suggesting that the L4S dual-queue AQM provides meaningful isolation that limits the damage Prague can inflict on classic flows. Second, the observed unfairness is not dramatically worse than the harm Cubic causes to itself in equivalent scenarios, implying that deploying Prague alongside Cubic does not introduce a qualitatively new fairness problem beyond what already exists in the status quo.

Nevertheless, the fact that Prague can still obtain a disproportionate share of bandwidth in high-bandwidth, high-RTT environments—even with flow isolation in place—highlights that partial L4S deployments warrant continued monitoring and that the AQM coupling mechanism does not fully eliminate inter-CCA unfairness under all conditions.

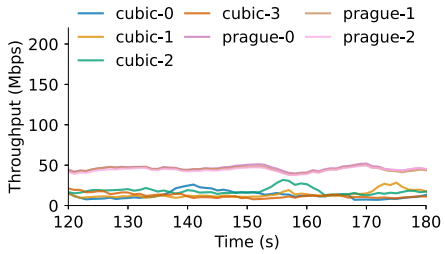


Figure 44: 200bw-91rtt-1.0bdp: 4 Cubic flows competing with 3 Prague flows after convergence.