Starvation in End-to-End Congestion Control

Venkat Arun, Mohammad Alizadeh, Hari Balakrishnan
{venkatar, alizadeh, hari}@csail.mit.edu
MIT Computer Science and Artificial Intelligence Lab, Cambridge, MA

ABSTRACT

To overcome weaknesses in traditional loss-based congestion control algorithms (CCAs), researchers have developed and deployed several delay-bounding CCAs that achieve high utilization without bloating delays (e.g., Vegas, FAST, BBR, PCC, Copa, etc.). When run on a path with a fixed bottleneck rate, these CCAs converge to a small delay range in equilibrium. This paper proves a surprising result: although designed to achieve reasonable inter-flow fairness, current methods to develop delay-bounding CCAs cannot always avoid starvation, an extreme form of unfairness. Starvation may occur when such a CCA runs on paths where non-congestive network delay variations due to real-world factors such as ACK aggregation and end-host scheduling exceed double the delay range that the CCA converges to in equilibrium. We provide experimental evidence for this result for BBR, PCC Vivace, and Copa with a link emulator. We discuss the implications of this result and posit that to guarantee no starvation an efficient delay-bounding CCA should design for a certain amount of non-congestive jitter and ensure that its equilibrium delay oscillations are at least one-half of this jitter.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Protocol correctness**;

KEYWORDS

Congestion Control, Delay-Convergence, Starvation

ACM Reference Format:
Venkat Arun, Mohammad Alizadeh, Hari Balakrishnan {venkatar, alizadeh, hari}@csail.mit.edu *MIT Computer Science and Artificial Intelligence Lab, Cambridge, MA.* 2022. Starvation in End-to-End Congestion Control. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3544216.3544223

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08.
https://doi.org/10.1145/3544216.3544223

1 INTRODUCTION

With the rise of interactive and real-time applications, the widespread recognition of bufferbloat as a significant problem [17], and increasing user expectations for high quality-of-experience, the networking community has developed a variety of *delay-bounding* congestion control algorithms (CCAs) for applications to use on the Internet. Unlike CCAs such as Reno/NewReno [22, 24], Cubic [20], and Compound [41], delay-bounding CCAs do not keep increasing their congestion window (cwnd) until they experience packet loss or receive acknowledgments (ACKs) with explicit congestion notification (ECN) bits [14]. Instead they use the measured round-trip time (RTT) and other factors (e.g., rate estimates) to set cwnd. They aim to achieve high utilization without bloating delays, and are also more efficient in the face of some packet loss.

The development of delay-bounding CCAs for the Internet began with Vegas in 1994 [6], followed by FAST [44], but largely stagnated because of the inability of these schemes to obtain good throughput when competing with buffer-filling CCAs such as Reno/NewReno and especially Cubic. Over the past ten years, however, delay-bounding CCAs have experienced a resurgence with several proposals that overcome these issues, including Sprout [46], Remy [45], BBR [8, 10], PCC [12, 13], Copa [3], and Verus [48]. Some of these schemes are widely deployed, most notably BBR, which is now the CCA used by many popular Internet sites.

In this paper we study how multiple flows running a given delay-bounding CCA share bandwidth. We start by noting that many CCAs share a common property. On ideal network paths with a constant bottleneck rate and propagation delay, they converge to a small delay range and oscillate within that range. The mean queueing delay range experienced at equilibrium is either constant (no matter what the bottleneck rate) or a decreasing function of the bottleneck rate (e.g., the CCA maintains a certain number of enqueued packets).

We show that this convergence property has an important consequence. Because most CCAs attempt to work across many orders of magnitude of rates, they *must* map a large rate range into a small delay range.[1] Thus, even small changes in estimated queueing delay would induce enormous changes

[1] Unless they are also willing to let the delay bound grow with the rate, which is not interesting in practice.

in the inferred rate. This observation suggested to us that perhaps bandwidth may not be shared equitably unless delays were perfectly measurable.

We realized, however, that even perfect delay measurements do not suffice unless there is a way to precisely measure only *congestive* queueing delays. The reason is that non-congestive (i.e., non-bottleneck-queueing) contributions to network delay are rarely constant in practice, and hard to separate from congestion. Due to effects such as ACK aggregation, delayed ACKs, end-host operating system or thread scheduling, token bucket filters, hardware offload timing variations, etc., packets experience jitter on network paths unrelated to bottleneck queueing.

The designers of delay-bounding CCAs have long recognized these issues and use various techniques to attempt to distinguish queueing from non-queueing (non-congestive) RTT variations. These include averages (Vegas, FAST, BBR) [6, 8, 44], minimums (LEDBAT, Copa) [3, 38], and maximums (Verus) [48] of RTT; maximums of rate (BBR) [8]; and repeating experiments (PCC) [12]. The problem, however, is inherently difficult due to the wide variety of delay patterns observed on real-world paths. A recent paper on CCA verification (CCAC) reported that many of these CCAs fail in surprising ways even in simple scenarios [2].

Given this motivation, we focused on understanding the effects of non-congestive jitter on delay-bounding CCAs, expecting some degree of unfairness and hoping to quantify it as a function of the delay jitter. What we found, instead, surprised us: efficient delay-bounding CCAs that converge in steady state to a bounded delay range are not merely unfair, but *cannot avoid starvation*. We prove that when two flows using the same CCA share a bottleneck link, if the non-congestive delay variations exceed double the difference between the maximum and minimum queueing delay at equilibrium, then there are patterns of non-congestive delay where one flow will get arbitrarily low throughput compared to the other. Our theorem shows that CCAs have to choose at most two out of three properties: high throughput, convergence to a small and bounded delay range, and no starvation. We prove this result using a simple but realistic network model where the network path can delay any packet in an arbitrary way within a small range.

It is important to note that starvation is a strong form of unfairness, going well beyond traditional notions of RTT unfairness or even one flow getting a constant factor higher throughput than the other. We prove that there is *no finite* $s > 1$ where the faster flow will always get less than s times the throughput of the slower one. To demonstrate that these results are not contrived or hypothetical, we use insights from the proof to show empirical scenarios with BBR, Copa, and PCC where starvation occurs — simple topologies with equal propagation RTTs where the ratio of throughput between two flows is $10:1$.[2]

This result sounds pessimal for delay-bounding CCAs, so the question is whether we are doomed to choose between bounding delays and avoiding starvation. We discuss how we might be able to achieve both desirable goals by being explicit about non-congestive delays in CCA design, ensuring that the CCA's delay variations in equilibrium are at least half as as large as the non-congestive jitter expected along a path. If that is not the case, then our results prove that starvation is inevitable.

2 DELAYS AND DELAY-CONVERGENCE

This section discusses the sources of delays, focusing on the difference between congestive and non-congestive sources of delay variation. It formally defines the notion of delay-convergence discussed earlier. Table 1 provides a glossary of symbols used in the paper.

2.1 Delay Contributors

The RTT experienced by a packet has three components:

(1) *congestive (bottleneck) delay*, which is the sum of the queueing delay incurred by packets waiting to be sent on the bottleneck link and the transmission time over the bottleneck link;
(2) *minimum packet propagation RTT (or delay)*, denoted R_m, defined as the time it takes for a single packet to traverse the non-bottleneck portions of the path and for the sender to receive an ACK (this includes both the speed-of-light delay of the path and the packet transmission delays on each non-bottleneck link);
(3) *non-congestive delay*, which we define as *jitter* due to network elements (perhaps also at the bottleneck) that may temporarily hold packets or ACKs but are not by themselves persistent rate bottlenecks.

Sources of non-congestive delay include ACK aggregation, delayed ACKs, end-host or in-network scheduling overheads, token bucket filters, hardware offload timing variations, and delays at the MAC and physical layers. In practice, non-congestive delay can range from a few milliseconds caused by operating system thread scheduling [29] to tens of milliseconds due to link-layer (e.g., Wi-Fi) ACK aggregation [18]. Each of these factors jitters the RTT in unpredictable ways.

Delay-bounding CCAs seek to control the queueing delays at the bottleneck link. The problems we address arise because end-to-end CCAs do not (and perhaps cannot) disambiguate between two sources of variability: queueing and the non-congestive delay jitter.

[2]This ratio would be higher but for limitations of our link emulator.

C	Link rate	R_m	Min propagation RTT
D	Model's delay bound	cwnd	Congestion window
cwnd	Congestion window	s	Bound on unfairness
$d_{\max}(C), d_{\min}(C)$	Max/min delay for CCA after convergence		
$\delta(C)$	$d_{\max}(C) - d_{\min}(C)$	$\delta^{\max}$	Upper bound on $\delta(C)$
λ	$d^{\max}$, $\delta^{\max}$ apply for $C > \lambda$	$d^{\max}$	Upper bound on $d_{\max}(C)$

Table 1: Glossary of symbols.

2.2 Delay-Convergence

We observe that despite their many operational differences, most (if not all) delay-bounding CCAs developed to date share a common property, which we call *delay-convergence*. They seek to converge to a fixed sending rate and queueing delay, making only small (if any) oscillations about that point. This design pattern offers two benefits. First, stable sending rates provide stable performance for the application. Second, many schemes, either implicitly or explicitly, map sending rates to corresponding (inferred) queueing delays, which makes it easier to reason about their behavior. As a result, many end-to-end delay-controlling CCAs employ this strategy including Vegas, FAST, Sprout, BBR, PCC Vivace, Copa, PCC Proteus, and Verus.

We define a delay-convergent CCA based on how it behaves when a single flow runs on an *ideal path*. An ideal path has a constant bottleneck link rate C, minimum packet propagation RTT R_m, and a bottleneck queue large enough to never overflow. Naturally, such a large enough queue size only exists for CCAs that bound their maximum queue.

Definition 1. *A CCA $\mathcal{A}$ is **delay-convergent** if two conditions hold when it is run on an ideal path with a given R_m:*

1. *There is a time T after which the RTT experienced is always in a bounded interval $[d_{\min}(C), d_{\max}(C)]$, where C is the bottleneck rate. Define $\delta(C) \triangleq d_{\max}(C) - d_{\min}(C)$.*
2. *Both $d_{\max}(\cdot)$ and $\delta(\cdot)$ are bounded for C not approaching zero, i.e., there exist a link rate $\lambda > 0$ and finite bounds $d^{\max}$ and $\delta^{\max}$ such that $d_{\max}(C) < d^{\max}$ and $\delta(C) < \delta^{\max}$ for all $C > \lambda$.*[3]

Figure 1 depicts this definition, showing the time evolution of the RTT for a hypothetical CCA. As we will see, CCAs that ensure a smaller $\delta^{\max}$ — and hence are "more convergent" — are more susceptible to starvation. In particular, we prove that starvation can occur if the delay *ambiguity* caused by non-congestive jitter delay is $> 2\delta^{\max}$. For many CCAs, $\delta^{\max}$ is small because they strive to keep delay variations small compared to R_m. Hence even a little ambiguity can cause starvation.

For instance, $\delta(C)$ is 0 for Vegas, FAST, and BBR in cwnd-limited mode; $\frac{4\alpha}{C}$ for Copa, where α is the packet size (< 0.5 ms when $C > 96$ Mbit/s and $\alpha = 1500$ bytes); $R_m/4$ for BBR in pacing mode; and $R_m/20$ for PCC Vivace (see Figure 3). We show that all these protocols suffer from starvation even in simple two-flow scenarios with small non-congestive jitter.

[3]A finite $d^{\max}$ guarantees that a CCA is delay-convergent. We still discuss $\delta^{\max}$ separately because it controls how susceptible a CCA is to starvation.

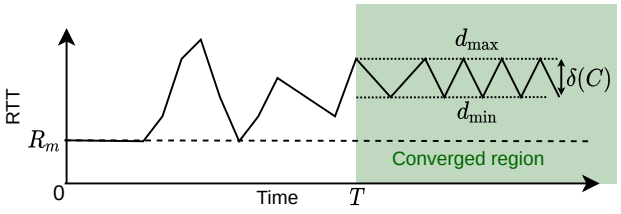


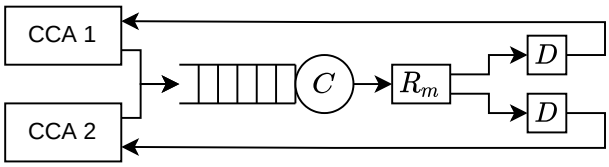
Figure 1: Ideal-path behavior of a hypothetical delay-convergent CCA.

For a fixed R_m, a delay-convergent CCA maps each link rate C to a delay range in steady state. Figure 2 shows how this range may vary as C changes. Typically $d_{\max}(\cdot)$ and $d_{\min}(\cdot)$ are decreasing functions of C because CCAs typically increase their rate as their estimate of congestive delay decreases.

Figure 3 shows the rate-delay graphs for some real-world delay-convergent CCAs. For all these CCAs, $\delta(C)$ is small (or grows smaller with C).

3 NETWORK MODEL

Our analysis and proof use the network model shown below:



Flows share a single large FIFO queue from which packets are dequeued at a constant rate of C bits/s[4] and experience a constant minimum packet propagation RTT R_m. Packets then pass through a component that adds a *non-deterministic, bounded* delay that represents the non-congestive delay on a network path. This component can delay packets by any value between 0 and D seconds without reordering packets. This non-congestive delay is flow-specific because the non-bottleneck parts of the network path may be different for different flows.

Non-congestive delays create ambiguity about the amount of queueing delay at the bottleneck. For example, when a CCA measures an RTT increase of Δd,[5] it only knows that the increase due to congestion is between $\Delta d - D$ (which might be

[4]We assume that the bottleneck link rate C is constant; when it varies as on wireless links, designing a CCA only becomes harder.

[5]CCAs are typically only concerned with changes in delay because the constant propagation delay R_m conveys no information about congestion.

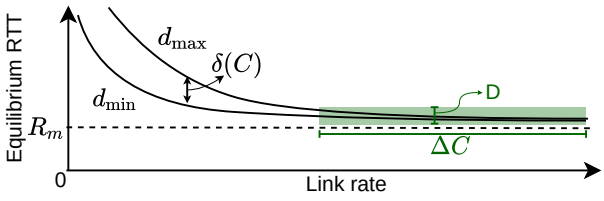


Figure 2: The rate-delay graph for a hypothetical delay-convergent algorithm. The C of the ideal path varies while its R_m is fixed.

negative) and $\Delta d+D$. This ambiguity creates confusion at the sender. If the delay observed is truly caused by congestion, the CCA should reduce its rate, but if it does so and the delay turns out to be non-congestive, it may under-utilize the network. When flows sharing a link estimate the queueing delay differently, they may send at unequal rates, which can cause starvation. A larger D implies higher ambiguity, and as $D \to 0$ the model approaches the ideal path that induces no ambiguity.

It is important that non-congestive delays in the model are non-deterministic but not random. This model ensures that standard filtering techniques cannot always distinguish between queueing and non-congestive delays. This modeling choice is motivated by the observation that while many CCAs attempt various kinds of filtering, they fail on some paths. For example, averaging [6, 13, 38] works only when non-congestive delay is random with mean 0. This is usually not the case; because network components add positive delay, the mean is usually positive. Using minimums of delay [3, 38] and maximums of rate [8] have also been tried, but have been shown to be ineffective [2]. Although we cannot rule out the existence of a sophisticated statistic that identifies non-congestive delays accurately (perhaps using machine learning), the problem is inherently difficult due to the wide variety of delay patterns observed on real-world paths. Thus we believe that a CCA should be stress-tested against our model for robustness.

Our model extends the CCAC model [2] to multiple flows, but is weaker[6]; for example, we do not model token bucket filters whereas CCAC's model does. Since our model is weaker, our theorems are stronger; any impossibility result that holds for our model also holds for the CCAC model.

4 STARVATION IS INEVITABLE FOR DELAY-CONVERGENT ALGORITHMS

Our main result is that if the delay range in equilibrium, $\delta^{\max}$, induced by the CCA is smaller than one-half of the measurement ambiguity of the network path, D, then there are conditions under which an efficient CCA cannot simultaneously bound delays and avoid starvation. This section describes the key ideas and steps involved in proving this result. We start with a motivating example and definitions before providing the formal statement and proof.

4.1 Example

Before formalizing our result, we describe an example that gives insight about how non-congestive delay can lead to starvation. Consider a CCA such as Vegas or FAST that seeks to maintain α packets in the queue per flow in equilibrium. α is a parameter of the algorithm (*e.g.*, $\alpha = 4$ packets). Over an ideal path, once the CCA hits this target, its rate stabilizes and the queue length never changes. Hence $\delta^{max} = 0$ and the RTT is $R_m + \alpha/C$ as shown in Figure 3.

The issue is that, to achieve this equilibrium, the CCA must measure the queueing delay with high precision. For example, with $\alpha = 4$ and a packet size of 1500 bytes, $\alpha/C = 0.5$ ms for $C = 96$ Mbit/s and 0.05 ms for $C = 960$ Mbit/s. Thus a difference of only 0.45 ms in the estimated queueing delay can cause the CCA to vary its sending rate by 10×! Therefore, if the delay measurement ambiguity exceeds this amount, it can easily cause severe unfairness. Delay jitter of tens of milliseconds occur on Wi-Fi [18] and cellular [47] links.[7]

This problem is not limited to Vegas or FAST or even CCAs with decreasing rate-delay functions as shown in Figure 2. Any delay-convergent CCA that seeks to bound delay variation below the level of jitter (delay ambiguity) of the network will suffer from the same problem. The fundamental issue is that very different link rates are consistent with similar delay measurements. When different flows experience different non-congestive delays, they can infer very different link rates, causing unfairness, and as we show, starvation.

4.2 Definitions

We assume that all flows in the network start at time ≥ 0. We define the *throughput* of a flow at time t to be the number of bytes acknowledged between time 0 to t divided by t. We are interested in starvation, an extreme form of unfairness. We propose the following definition.

Definition 2. *Consider two flows f_1 and f_2 starting from arbitrary initial conditions (e.g., one of the flows could have run for a long time and the other just starting). The network is* ***s-fair*** *if there always exists a finite time t such that for all time beyond t, the ratio of the throughput achieved by the faster flow to the slower one is smaller than s.*

Definition 3. ***Starvation*** *is said to occur if the network is not s-fair for any finite s.*

[6] In the single flow case, the CCAC paper shows that it includes all network behaviors of our model.

[7] See also slide 5 of https://datatracker.ietf.org/meeting/101/materials/slides-101-iccrg-an-update-on-bbr-work-at-google-00

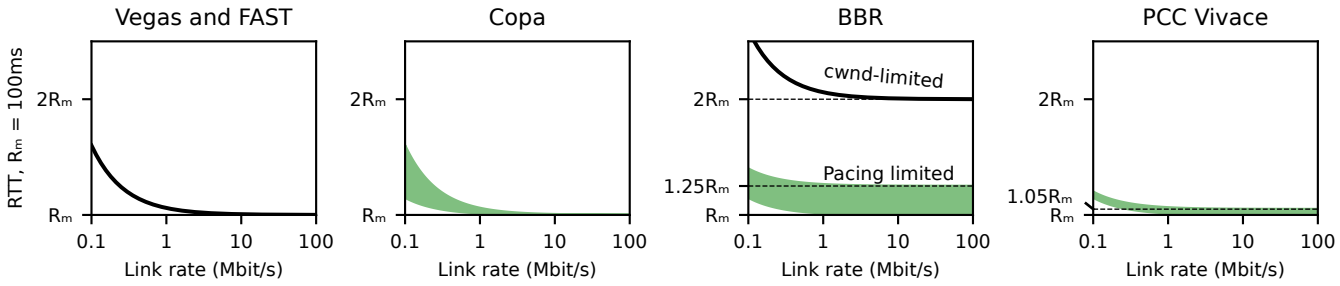


Figure 3: How various delay-bounding CCAs map delay to sending rates. The shaded region shows d_{max} and d_{min} for each CCA. The region's width is $\delta(C)$. BBR has two modes (shown). For Vegas, FAST, and BBR (cwnd-limited), $d_{max} = d_{min}$, hence it is a line, not a region. For these CCAs, $\delta(C) = 0$. Section 5 describes how these mappings arise from the dynamics of these algorithms. Delay increases as $C \to 0$ for all CCAs since a transmission delay of $1/C$ is unavoidable.

This definition allows for massive unfairness without causing starvation. For example, if in steady state the two flows achieve a throughput ratio of a million to one, we would still say that there is no starvation. Our analysis asks if *any* finite ratio is achievable and proves that it is impossible for delay-convergent CCAs.

To prove that starvation occurs, we need to show that there *exist* starting states and network behavior after which the flows never improve their bandwidth allocation *no matter how long they run*. This is surprising because we expect our CCAs to eventually converge to a good allocation no matter the initial allocation. Flows start with unequal allocations, for instance, when one starts after the other. We also show empirically that for Copa, BBR and PCC it is quite easy to cause starvation even when the flows start at the same time.

One might think that this starvation claim requires CCAs to run at high efficiency; for example, perhaps a focus on 100% utilization leads to aggressive behavior causing starvation. We find that our result applies to CCAs that always utilize at least some constant fraction of the link capacity, which can be quite small (say 1%), and far from extreme efficiency. We only need to eliminate "silly" CCAs such as "set cwnd = 10 always", which avoid starvation but are inefficient and impractical, as well as application-limited flows.

Definition 4. *A CCA is* **f-efficient** *if, when run on an ideal path with bottleneck link rate C and minimum RTT R_m, it eventually gets a throughput of at least fC; i.e., for any t, there exists a $t' > t$ such that the number of delivered bytes in the period 0 to t' is at least fCt'.*

We define f-efficiency in this way because we need to make statements about *all* delay-convergent CCAs, even absurd ones. For instance one can contrive a CCA for which the limit of the throughput as $t \to \infty$ does not exist. Practical CCAs are usually better behaved. We believe this definition adequately captures steady-state throughput since throughput must be at least fC infinitely many times.

4.3 Starvation Theorem

The following theorem states our result about the inevitability of starvation for delay-convergent CCAs.

Theorem 1. *For any deterministic, f-efficient, delay-convergent CCA $\mathcal{A}$, any propagation delay R_m, any throughput ratio $s \geq 1$, and any $D > 2\delta^{\max}$, there exists a network scenario with two flows (specified via two per-flow initial states and trajectories of non-congestive delays), such that one flow gets a throughput x_1 and the other flow gets a throughput $x_2 \geq s \cdot x_1$.*

The outline for the proof is as follows. Appendix A fills in the mathematical details that we omit here.

Step 1. Recall that the bound $\delta^{\max}$ has a corresponding λ; $\delta(C) < \delta^{\max}$ and $d^{\max}(C) < \delta^{\max}$ for all $C > \lambda$. We identify two bottleneck link rates $C_1, C_2 \geq \lambda$, such that C_2 is much larger than C_1 (at least a factor s/f larger) but the CCA $\mathcal{A}$, when run independently on links with these two rates, converges to delays in two ranges that are close to each other. Here, "close" means that the delay ranges achieved at rates C_1 and C_2 lie within an interval of size $\delta^{max} + \epsilon$. We will prove that for any $\epsilon > 0$, we can always find such a C_1 and C_2 for a delay-convergent CCA (we will pick ϵ later).

Our claim follows from a pigeonhole principle argument illustrated in Figure 4. Recall that the delays for any link rate $> \lambda$ must fall in the interval $[R_m, d^{max}]$. Now there are only a finite number of non-overlapping intervals of size ϵ that can fit in $[R_m, d^{max}]$ (at most $\lceil (d^{max} - R_m)/\epsilon \rceil$ of them). But consider, for example, the infinite sequence of link rates $(\lambda_0, \lambda_1, \ldots)$, defined as $\lambda_i = \lambda \cdot (s/f)^i$. We have: $\lambda_i \geq \lambda$ for all i and $\lambda_j \geq (s/f) \cdot \lambda_i$ for all $j > i$. Since this is an infinite sequence of link rates, we can find a pair $\lambda_j \geq (s/f) \cdot \lambda_i$ such that $d_{max}(\lambda_i)$ and $d_{max}(\lambda_j)$ fall within the same interval of size ϵ, i.e. $|d_{\max}(\lambda_1) - d_{\max}(\lambda_2)| < \epsilon$. Let $C_1 = \lambda_i$ and $C_2 = \lambda_j$.

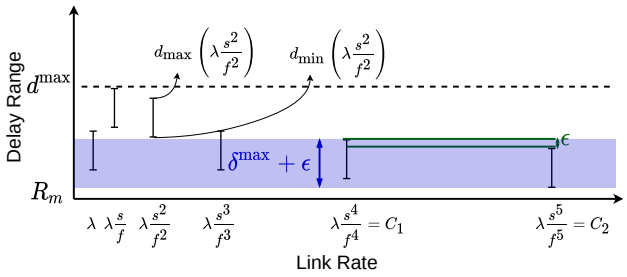


Figure 4: Pigeonhole principle argument used in the proof.

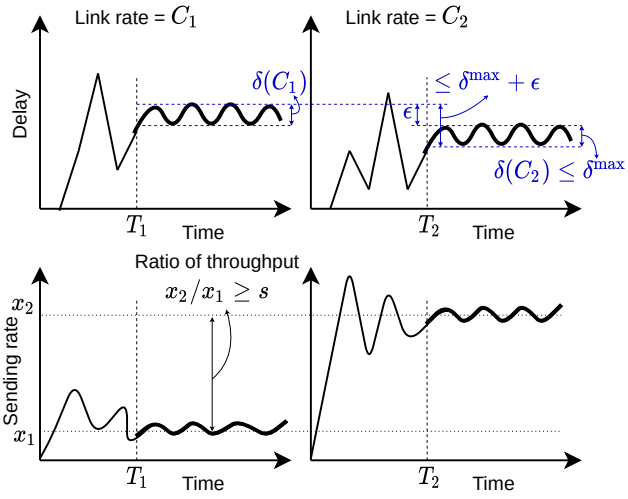


Figure 5: Convergence trajectory of a hypothetical delay-convergent CCA.

The claim follows because the delay range for any link rate has size at most δ^{max}.

Step 2. Consider the trajectories of sending rate and delay when running a flow using CCA $\mathcal{A}$ independently on ideal paths with link rates C_1 and C_2. Recall that an ideal path has zero non-congestive delay. Figure 5 shows an example of a hypothetical delay-convergent algorithm that converges after times T_1 and T_2 in the two cases. The CCA converges to similar but distinct delay ranges on the two links. However, it converges to very different sending rates in the two cases. Let x_1 and x_2 denote the long-term throughput achieved on links of capacity C_1 and C_2 respectively. Clearly $x_1 \leq C_1$, and since CCA $\mathcal{A}$ is f-efficient, $x_2 \geq fC_2$. It follows that $x_2 \geq fC_2 \geq f \cdot (s/f)C_1 = s \cdot C_1 \geq s \cdot x_1$, where we have used the fact that $C_2 \geq (s/f)C_1$.

Step 3. To recap, so far we have identified two link rates C_1, C_2 such that the CCA converges to delays in a similar range but its sending rates are far apart on these links. In the final step, we construct a 2-flow scenario on a *shared* link with rate $C_1 + C_2$ and propagation delay R_m, such that the two flows follow exactly the same trajectories we found in Step 2. Therefore, in this scenario, the two flows converge to throughputs x_1 and x_2 that satisfy our starvation criteria: $x_2 \geq s \cdot x_1$.

Our starting observation is that for a deterministic CCA,[8] the sending rate at any time t is a function of the delays observed up to time t and the initial state of the algorithm. Therefore, if we can set the non-congestive delays in the 2-flow scenario such that each flow observes a *total* delay that is identical to one of the delay trajectories from Step 2, then the two flows' sending rates will follow the rate trajectories from Step 2 as well. We will refer to controlling the non-congestive delay on a flow's path to achieve a specific delay trajectory as *emulating* that delay trajectory. The question is can we emulate the delay trajectories from Step 2 by adding up to D seconds of non-congestive delay to each flow's path in the 2-flow scenario?

To complete the construction of the 2-flow scenario, we must specify the initial state of the two flows' CCAs, the initial state of the shared link's queue, and the trajectories of the non-congestive delay for the two flows at all times. Let $d_1(t)$ and $d_2(t)$ be the delay trajectories and $r_1(t)$ and $r_2(t)$ be the rate trajectories achieved for links C_1 and C_2 respectively in Step 2. Assume that the flows converged to their eventual delay ranges at times T_1 and T_2, as shown in the figure 5. Define $\bar{d}_1(t) = d_1(t + T_1), \bar{r}_1(t) = r_1(t + T_1), \bar{d}_2(t) = d_2(t + T_2), \bar{r}_2(t) = r_2(t + T_2)$ as time-shifted versions of the delay and rate trajectories such that the time origin is set to the time of convergence. These trajectories correspond to the bold segments shown in the figure.

We initialize the internal state of the two flows to the states of the corresponding flow in Step 2 at times T_1 and T_2. Our goal now is to emulate the delays $\bar{d}_1(t)$ and $\bar{d}_2(t)$ for all $t \geq 0$ by choosing the non-congestive delay for the two flows appropriately. Let $\eta_1^\star(t)$ and $\eta_2^\star(t)$ be the non-congestive delays for flows 1 and 2 respectively, and let $d^\star(t)$ be the sum of the propagation delay R_m and queueing delay in the 2-flow scenario.[9] Note that $d^\star(t)$ is common to both flows. To achieve emulation, we need to ensure that: $d^\star(t)+\eta_1^\star(t) = \bar{d}_1(t)$ and $d^\star(t) + \eta_2^\star(t) = \bar{d}_2(t)$ for all $t \geq 0$.

We control $\eta_1^\star(t)$ and $\eta_2^\star(t)$, but what is $d^\star(t)$? To get a handle on $d^\star(t)$, let's assume for the moment that our delay emulation is successful and the two flows send precisely at the rates $\bar{r}_1(t)$ and $\bar{r}_2(t)$. Then, we can compute the queueing delay in the 2-flow scenario exactly: it is simply the delay of a queue with net arrival rate of $\bar{r}_1(t) + \bar{r}_2(t)$ and net drain

[8] While CCAs such as BBR and PCC employ randomness, it does not materially affect the result (see §5).

[9] We use a superscript $\star$ for all signals in the 2-flow scenario.

rate of $C_1 + C_2$. In the proof (see Appendix A for details), we use this observation to show that $d^\star(t)$ is given by:

$$d^\star(t) = \underbrace{\frac{C_1\bar{d}_1(t) + C_2\bar{d}_1(t)}{C_1 + C_2}}_{\text{Time-varying}} - \underbrace{(\delta^{\max} + \epsilon)}_{\text{Constant}}.$$

$d^\star(t)$ has two components: (i) a time-varying part that is a weighted average of the delay trajectories $\bar{d}_1(t)$ and $\bar{d}_2(t)$ from Step 2; (ii) a constant part that depends on the initial queueing delay chosen in the 2-flow scenario. This initial delay is $d^\star(0)$, and we are free to set it to any value $\geq R_m$. In the proof, we choose this value to obtain the expression above for $d^\star(t)$.

For delay emulation to succeed, we must be able to satisfy $d^\star(t) + \eta_i^\star(t) = \bar{d}_i(t)$ for some $\eta_i^\star(t) \in [0, D]$ for all $t \geq 0$ and $i \in \{1, 2\}$. This can be done if and only if $d^\star(t)$ satisfies two properties:

(1) $d^\star(t) \leq \min\{\bar{d}_1(t), \bar{d}_2(t)\}$, i.e. $d^\star(t)$ must be a lower bound on $\bar{d}_1(t)$ and $\bar{d}_2(t)$. This guarantees that the non-congestive delay is non-negative.
(2) $\max\{\bar{d}_1(t), \bar{d}_2(t)\} \leq d^\star(t) + D$, i.e. $d^\star(t) + D$ must be an upper bound on $\bar{d}_1(t)$ and $\bar{d}_2(t)$. This guarantees that the non-congestive delay is $\leq D$.

The last step of the proof shows that $d^\star(\cdot)$ defined above satisfies both properties. We defer the details to Appendix A, but the reason this works is that the delay values for $\bar{d}_1(t)$ and $\bar{d}_2(t)$ are never too far from each other, and hence never too far from $d^\star(\cdot)$. Figure 6 shows how $d^\star(\cdot)$ is situated relative to $\bar{d}_1(\cdot)$ and $\bar{d}_2(\cdot)$ for our running example.

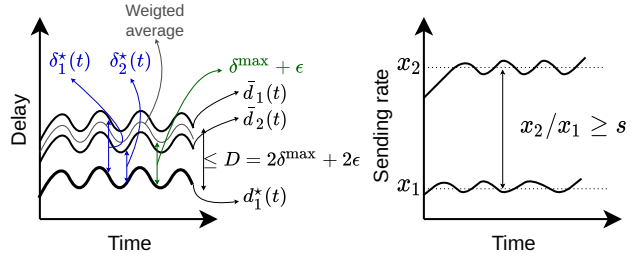


Figure 6: How $d^\star(\cdot)$ is situated relative to $\bar{d}_1(\cdot)$ and $\bar{d}_2(\cdot)$ in the example.

5 REAL-WORLD ALGORITHMS

Extreme unfairness tantamount to starvation can occur when multiple flows share a bottleneck link but the rest of the path they traverse is different. This section shows that starvation is not merely theoretical, but can be observed in real-world delay-convergent CCAs even in simple settings. The scenarios we show are inspired by our proof.

We discuss several CCAs here, explaining why certain CCAs are delay-convergent and giving their oscillation range (i.e., $d_{\min}(C)$ and $d_{\max}(C)$). Then we describe the trace produced by Theorem 1 to cause one of the flows to starve and discuss how it can arise in realistic network path. Finally we experimentally demonstrate the occurrence of starvation using real implementations of several CCAs. The only non-delay-convergent CCAs we are aware of are loss-based. We discuss them as well.

We have a simple criterion for deciding whether a network scenario is realistic. First, it should possible for a composition of real network elements to produce the behavior. Second, the behavior should not be a sequence of coincidences. That is, the probability of the behavior happening should not decrease with the duration of that behavior; if flows were to be on such a network, one of the flows must be likely to starve. As we we will see, the paths we come up with are common on the Internet.

5.1 Vegas, FAST, and Copa

These CCAs all have the same equilibrium, though their dynamics differ. They all try to maintain a constant number (α) of packets in the queue. Even with a single flow, these algorithms can send at a rate that is arbitrarily smaller than the link rate. This happens when they overestimate their queueing delay and slow down. They can overestimate their queueing delay on paths with non-congestive delays or underestimate their minimum RTT, R_m.

Copa attempts to mitigate these problems by computing queueing delay as *standing RTT - min RTT*, instead of *latest RTT - min RTT* where *standing RTT* is the minimum RTT observed over a short period of time (*min RTT* is the minimum over a long period). Unfortunately, this method is not robust to persistent non-congestive delay. CCAC [2] found a way to use multiple network elements to fool this statistic even when there is no persistent non-congestive delay in a single network element.

The following simple scenario drives Copa to starvation. Run a Copa flow on a 120 Mbit/s link with R_m of 60 ms. Send one packet with a 59 ms RTT to cause it to under-estimate its minimum RTT. Here Copa achieved a throughput of 8 Mbit/s, which was caused by a 1 ms error in measuring the delay of one packet. We ran this experiment with the Mahimahi emulator [32].

This single-flow phenomenon also occurs with two or more flows, e.g., when the delay jitter happens only for one flow. We repeat the above experiment with two flows where only one flow gets the 59 ms packet. In this case, one flow gets 8.8 Mbit/s while the other gets 95 Mbit/s. Vegas and FAST can also be compromised in similar ways.

5.2 BBR

BBR has two modes of operation. The first mode is the one described in the original paper [8]. Here, BBR's sending rate is limited by its pacing rate, which is set to (pacing_gain)·(bandwidth_estimate). The pacing_gain is usually 1, allowing BBR to send at its estimated bandwidth. Every 8 RTTs, pacing_gain is increased to 1.25 to probe and see if more bandwidth is available. In the RTT following this increase, pacing_gain is reduced to 0.75 to drain any queue created during the gain. The bandwidth_estimate variable is set to the maximum bandwidth measured over the last 10 RTTs, where bandwidth is measured by dividing the number of acknowledged bytes over 1 RTT intervals.

If more bandwidth were available during the probe phase (or at any other time), bandwidth_estimate would have increased. The way BBR seeks to achieve fairness is by having different flows probe at different random times, as described in a fairness analysis document [42]. In the pacing mode, $d_{\min}(C) = R_m$ and $d_{\max}(C) = 1.25R_m$. If $D > \delta^{\max} = 0.25R_m$, our network model can prevent one of the flows from recognizing that additional bandwidth is available during the probe phase, causing it to send at a rate that is arbitrarily small compared to its fair share. This situation is identical to the one described in the CCAC paper [2] and happens in the presence of a network element, such as a cellular base station, whose bandwidth allocation lags behind the flow's demand. Our proof constructs exactly this behavior. By contrast, if D is smaller, BBR can experience starvation in the cwnd-limited mode as described below.

Because the bandwidth estimate uses a max filter, BBR tends to over-estimate the link rate when ACKs do not arrive smoothly, since there will be some RTT during which we get greater than average rate. As a result, the queue can grow indefinitely. To prevent such bufferbloat, BBR uses cwnd to cap the number of in-flight packets. Hence when BBR has overestimated the bandwidth, it is in the cwnd-limited mode [21, 43]. In this mode, cwnd controls the behavior and dynamics of the pacing rate and its increase/decrease during bandwidth probing are not material to the sender's transmissions.

Starvation in cwnd-limited mode. Here, cwnd is set to 2·(bandwidth_estimate)·(R_m_estimate) + α. The α term is called quanta in the BBR document [9], and is intended to "allow enough quanta in flight on the sending and receiving hosts to reach high throughput even in environments using offload mechanisms". This term was removed in a later version, but another additive term extra_acked was added in its stead [10]. We believe that the α term performs a critical function in addition to the intended one; it enables fairness in the cwnd-limited mode by forcing a unique fixed point.

We now calculate the equilibrium point for BBR on an ideal path. At equilibrium, its bandwidth estimate equals the ACK arrival rate, which equals the sending rate, cwnd/RTT. Hence we have

$$\begin{aligned}\text{cwnd} &= 2R_m \cdot (\texttt{bandwidth\_estimate}) + \alpha \\ &= 2R_m \cdot \text{cwnd}/\text{RTT} + \alpha\end{aligned}$$

Thus, equilibrium_sending_rate = $\frac{\text{cwnd}}{\text{RTT}} = \frac{\alpha}{\text{RTT}-2R_m}$ (Figure 3). At equilibrium, RTT > $2R_m$, achieving full utilization.

When there is only one flow, the sending rate is C because the link is fully utilized. This gives cwnd = $CR_m + \alpha$. We can repeat this calculation for multiple flows using the additional constraint that ACKs for the i^{th} flow arrive at the rate of $C\frac{cwnd_i}{\sum_{j=1}^{n} cwnd_j}$. Then we get $cwnd_i = 2CR_m/n + \alpha$. At this equilibrium, the RTT is $2R_m + n\alpha/C$.

This behavior is similar to Vegas, FAST, and Copa where at equilibrium the RTT is $R_m + n\alpha/C$. The only difference is that BBR maintains an extra R_m of delay. But this is an important difference; unless BBR overestimates the congestive delay by R_m, it maintains non-zero queueing delay and achieves full utilization. By contrast, even a single Vegas/FAST/Copa flow can under-utilize the link if it mis-estimates the RTT by α/C. However, *fairness* is still achieved for BBR by the $n\alpha/C$ term. If we remove the $+\alpha$ term and recalculate the equilibrium, we find that any value of $cwnd_1$ and $cwnd_2$ can be a fixed point as long as $cwnd_1 + cwnd_2 = 2R_mC$, even if $cwnd_1 = 0$ and $cwnd_2 = 2R_mC$! If one BBR flow is running and has occupied the entire link, when a new flow comes, it will not achieve its fair share. While the $+\alpha$ term fixes the problem, $n\alpha/C$ is a rather small value of delay to measure and becomes smaller as C grows. Hence the same precision is required as in Vegas, FAST and Copa. The analysis suggests that when flows with different RTTs compete, the flow with the smaller RTT can starve, as has been observed empirically [21].

Empirical evaluation. BBR is a complex protocol, spanning 900 lines of code. To confirm that our simplified theoretical model for BBR is useful, we conducted some experiments. We used Mahimahi [32] to run two BBR flows (implemented in the Linux kernel v5.13.0) with R_m of 40 ms and 80 ms over a common bottleneck link of 120 Mbit/s for 60 seconds. Since there were two BBR flows, their interaction and natural OS jitter was enough to push them into cwnd-limited mode. On paths without OS jitter, some other source of jitter may be necessary to break BBR. In this situation, one flow got an average of 8.3 Mbit/s and the other got 107 Mbit/s, an order-of-magnitude difference in sending rates.

Delay-convergence in BBR. Strictly speaking, the cwnd-limited mode does not meet our definition for delay-convergent CCAs with $\delta(C) = 0$ because (1) some jitter is necessary for this mode to be active while our definition is

over ideal links and (2) BBR periodically stops transmitting to probe for minimum RTT. During a probe, RTT falls to R_m, so $\delta(C)$ is $R_m \neq 0$. These are, however, mere technicalities and the ideas behind our starvation proof still work. First, instead of running in an ideal link, we need to run on a link with some jitter. Second, our proof works even if the CCA has oracular knowledge of R_m; alternatively, we can stop emulation when the CCA is probing for RTT and the rest of the argument holds, since BBR will ignore the data collected during probe. Further, BBR uses randomness in when it probes for bandwidth while our theorem only applies to deterministic CCAs. However, this does not make a tangible difference as demonstrated by the experiment above.

5.3 PCC Vivace

The PCC Vivace paper [13] showed that on an ideal link it converges to a fair throughput allocation that fully utilizes the link and maintains zero queueing delay. It regularly increases and decreases its rate to check if that will increase its utility function. Based on the largest constants given in the paper this will cause the queueing delay to oscillate at most between R_m and $1.05R_m$. These form $d_{\min}(C)$ and $d_{\max}(C)$ with $\delta^{\max} = R_m/20$. PCC's rate-delay curve is shown in Figure 3. Like BBR, PCC also employs randomness, but it this does not make a difference to the result. (In fact, we conjecture that Theorem 1 is true for randomized CCAs too.)

To empirically test if PCC experiences starvation, we ran two PCC Vivace flows in a Mahimahi emulator with 60 ms propagation delay and 120 Mbit/s bandwidth. For one of the flows, ACKs are received only at integer multiples of 60 ms, preventing finer delay measurement. This flow only achieved 9.9 Mbit/s while the other flow got 99.4 Mbit/s. We used Vivace's kernel module for the experiments [25]. [10]

5.4 Loss-based CCAs

CCAs like NewReno [22] or Cubic [20] are not delay-convergent. We study their fairness in two ways. First, we extended CCAC to handle multiple flows (see Appendix C) and used it to discover bad behavior when there is non-congestive delay jitter. Second, we modify our model to allow it to preferentially drop packets for one flow.

Let us take delay jitter first. Suppose two flows share a bottleneck, but one of them is well-paced while the other sends packets in bursts. This situation can occur with generic segment offloading (GSO) [26] used by the sender for CPU efficiency, ACK aggregation (say due to WiFi [18]), or delayed ACKs [5, 23]. As the queue gets nearly full, the flow that sends packets in bursts is more likely to lose packets. When this happens, this flow reduces its cwnd and the queue stops being full until a while later when again the bursty flow is more likely to lose packets. Packet bursts can reduce the utilization even when there is only a single flow on the link, but only when the bursts are large [2]. However when two or more flows are present, even a small burst can cause unfairness. This is illustrated using an ns3 [1] simulation in Figure 7. We also reproduced similar results in emulation in Mahimahi.

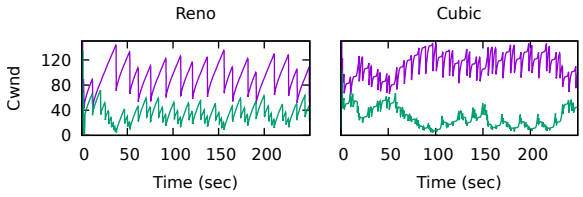


Figure 7: Congestion window evolution when two flows are run on a 6 Mbit/s, 120 ms link and 60 packets of buffer. The lower flow's receiver uses delayed ACKs of up to 4 packets while the other ACKs every packet. The CCAs used are Reno (left) and Cubic (right). The ratio of throughput obtained between the two flows is 2.7× and 3.2× for Reno and Cubic respectively.

One flow using delayed ACKs of just 4 packets can cause it to get 1/3 the throughput of the other flow. Nevertheless this is not starvation since the unfairness is bounded. In loss-based AIMD, when the faster flow reduces its cwnd (which it eventually must when it occupies nearly all of the link), it gives the slower flow time to ramp up before it starts to decrease its cwnd again. In Cubic, the faster flow will eventually overshoot the entire bandwidth-delay product (BDP) as governed by the cubic function. The slower flow can only increase its cwnd and experience losses two or three times before this happens. Hence the unfairness is bounded. We used CCAC to prove that there is no trace of length 10 RTTs where starvation is unbounded for two AIMD flows when the bottleneck has 1 BDP of buffer. Proving this result for any trace length is future work.

PCC Allegro [12]. While loss-based AIMD is not delay-convergent and is therefore immune to small delay jitter, it converges to a loss rate as a function of the BDP [27]: $\text{cwnd} \propto \frac{1}{\sqrt{\text{loss rate}}}$. If we add to the network model the ability to arbitrarily drop a small fraction of packets, we can get the CCA to under-utilize the link when the BDP is sufficiently large. This is well known and works even when only a single flow is present.

PCC's behavior is more interesting. To get around this problem, the loss-based PCC variant, PCC Allegro, has a loss threshold that it can tolerate. As long as the packet

[10] We needed a relatively large jitter of 60 ms because Mahimahi is a noisy emulator. Linux user-space scheduling adds several milliseconds of jitter to *both flows*. We need the flows to have different jitter. A cleaner, less-variable network emulation environment will produce a configuration with smaller non-congestive jitter for starvation to occur. Similar effects prevented us from observing a starvation ratio greater than 10:1 for other CCAs.

loss rate is lower than this threshold, it will fully utilize the link. In our framework, this is analogous to BBR in cwnd-limited mode always maintaining R_m seconds of queueing; as long as error in delay measurement is smaller than R_m, BBR fully utilizes the link. Just as BBR is an improvement to the Vegas family, PCC is a loss-resilient improvement to the Reno family. However, just like BBR, PCC can also experience starvation when one of the flows (but not the other) experiences even small amounts of congestion signal; for BBR it is propagation delay, for PCC it is random loss. The reason is analogous; the space of loss rates is smaller than the space of sending rates.

As empirical validation, we ran two PCC flows for 60 seconds on a 120 Mbit/s Mahimahi link with 40 ms RTT and 1 BDP buffer. One of the flows experienced a random loss rate of 2% and got only 10.3 Mbit/s while the other, which experienced no random loss, got 99.1 Mbit/s. PCC is supposed to be resilient to up to 5% loss. Indeed, when we ran two flows with 2% loss, they shared the link fairly and efficiently. When we ran just one flow with 2% loss, PCC was able to fully utilize the link capacity. Like BBR, PCC breaks only when two flows are present and experience unequal congestion signals (here, loss). We do not believe it is possible to circumvent this problem with algorithms that map loss rates (or delays) to sending rates.

6 IMPLICATIONS AND NEXT STEPS

The main lesson from this paper is that to avoid starvation delay-convergent CCAs must explicitly model and design for non-congestive delays. This affects the design space of CCAs in three key ways:

(1) If D is the bound on network jitter, the CCA must maintain at-least D seconds of delay to be f-efficient (§6.1).
(2) To avoid starvation, it is not enough to maintain a queue that is larger D, but the *variation* in delay must be larger as well (see §6.2).
(3) If we have an upper bound on the link rate, then we can achieve all three objectives without large variance in queueing delay. The delay bound achieved is a function of D and our maximum tolerable unfairness (§6.3).

The rest of this section expands on these ideas, and concludes by proving an impossibility result for delay-bounded (non-delay-convergent) CCAs.

6.1 Is an f-efficient, Delay-Convergent CCA Achievable?

Can a CCA simultaneously achieve f-efficiency and delay-convergence if we can tolerate starvation? The answer is not obvious because current schemes like BBR, Copa, Vegas, and FAST do not, as shown in the recent CCAC paper [2]. That paper found counterexamples consistent with the delay-jitter network model of this paper showing scenarios where f-efficiency is not achieved for any $f > 0$.

We are not aware of any existing CCAs that are both f-efficient and delay-convergent. That said, we have not analyzed every algorithm in depth. Perhaps the best hope is offered by BBR. The CCAC paper showed that if BBR were modified to have a higher pacing rate, CCAC could no longer find any example where BBR under-utilizes the link. We believe this happens because the higher pacing rate forces BBR to operate in the cwnd-limited mode, since the behavior is identical to when BBR over-estimates the pacing rate (see §5.4). In this mode, $d_{\max}(C) \geq 2R_m$, which is large.

Note, however, that CCAC did not *prove* that BBR is f-efficient. It merely ruled out the existence of under-utilization over short ($\leq$ 10 RTTs) sequences of network behavior. It also assumed a sender with oracular knowledge of R_m.

We conjecture, however, that it is possible to design an f-efficient, delay-convergent CCA if we ignored starvation. Perhaps the modified BBR is such an algorithm. Any such CCA must maintain a larger delay than the network jitter, or risk under-utilization. The following theorem formalizes this.

Theorem 2. *Any deterministic CCA for which there exists a link rate C and minimum RTT R_m such that $d_{\max}(C) \leq D$ can experience arbitrarily low utilization in our network model with parameter D.*

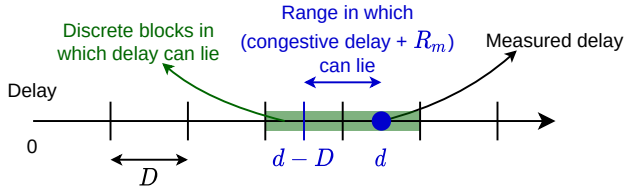
Proof. The idea is similar to our proof for Theorem 1. Let the delay experienced by the CCA on an ideal path of rate C and propagation delay R_m be $d(t)$. Let its sending rate be $r(t)$. We will construct a network with propagation delay R_m and $C' \gg C$ such that the delay experienced by the CCA is exactly $d(t)$. As a result, the CCA will transmit at exactly $r(t)$ since the CCA is deterministic. This is possible because we will choose C' to be large enough that the queuing delay, $q(t) \leq d(t)$. Now $d(t) \leq d_{\max}(C) \leq D$ where the first inequality follows from the definition of $d_{\max}(\cdot)$. Hence $0 \leq d(t) - q(t) \leq D$, which is the condition we need for emulation. Since the actual link rate, C', can be arbitrarily larger than the rate $\approx C$ at which the CCA sends, starvation occurs. □

6.2 Larger Oscillations May Avoid Starvation

Theorem 1 shows that a CCA whose ideal-path delay variation, $\delta^{\max}$, is smaller than one-half of the non-congestive delay in the network, D, cannot simultaneously be f-efficient, bound delays, and avoid starvation. Hence the only way to achieve all three properties is to design a CCA that has large delay variation on ideal paths (i.e., at equilibrium).

Why might large delay variations avoid starvation? The reason why CCAs with small variations starve is that there

isn't enough space to assign all achievable rates to distinct-enough delay ranges. This is because once a CCA has converged to a small delay range, it keeps receiving the same signal over and over. It cannot distinguish delay variations due to congestion from those due to non-congestive jitter.



An imprecise but useful mental model is to think of measurement ambiguity as discretizing measurement. When we measure an RTT of d, we know that up to D of it may be non-congestive. The material portion of d for our purposes is in the range $[\max(0, d-D), d]$. Let us divide the RTT d into discrete blocks of size D. This range tells us that the correct (`congestive delay` + R_m) must lie in one of the two blocks highlighted in the picture above.

A delay-convergent algorithm with $\delta^{\max} \leq D$ gets the same set of blocks over and over again. But one with larger variation can get different blocks/bits of information each time. This forms an infinitely large stream of bits in which to encode the correct sending rate. Different bit streams can now be assigned to different sending rates. This helps sidestep the pigeonhole argument, which forms the bedrock of our impossibility proof.

For instance, sending rate can be encoded in the *frequency* of oscillation of delay, rather than its absolute value. Given enough samples, it may be possible to measure frequency with arbitrary precision, avoiding starvation. Loss-based algorithms like AIMD do this; AIMD's sending rate is determined by the frequency at which packets drop. While our analysis does not include packet losses, it is interesting to note that AIMD has large oscillations relative to D. Hence smaller delay jitter does cause starvation (§5.4). If the oscillations (and hence the buffer size) were smaller than D, it is indeed possible to starve AIMD; one flow always sends packets in bursts that are larger than the buffer, experiencing drops, while the other flow grows its cwnd to be arbitrarily larger than the first flow. We conjecture that AIMD on delay is an interesting design space for researchers to seek starvation-free CCAs.

6.3 Avoiding Starvation in a Bounded Rate Range

A CCA that seeks to converge to a small range of delays must map sending rates that are far away to delays that are more than D apart, or risk starvation. While this is not possible for an infinitely large range of rates, it becomes possible if we know that the correct sending rate will come from a bounded range. In practice, we may know such bounds *a priori* from a knowledge of network parameters (e.g., access link rate at the sender) or by profiling applications.

Rather than worry about perfect flow-level fairness, which might not be an interesting goal in practice [7], we seek to be s-fair; i.e., bound unfairness to a maximum specified throughput ratio of $s > 1$. For a given D, and maximum tolerable delay $R^{\max}$, we define a figure-of-merit for a rate-delay curve as the ratio of the maximum rate it supports to the minimum: $\frac{\mu_+}{\mu_-}$. Rates in today's Internet can span several orders of magnitude, from $<$ 100 Kbit/s to $\approx$ 10 Gbit/s. Hence having $\frac{\mu_+}{\mu_-} \geq 10^3$ and perhaps $\approx 10^5$ is desirable.

For Vegas, FAST, and Copa, the rate-delay function is $\mu(d) = \alpha/(d - R_m)$, where μ is the sending rate [3]. The function for BBR's cwnd-limited mode is $\mu(d) = \alpha/(d - 2R_m)$. The arguments in this section are similar for both these functions, so we analyze the Vegas/FAST/Copa function here.

To achieve s-fairness for all $\mu \in [\mu_-, \mu_+]$, we want the difference in the delays seen between μ and $s\mu$ to exceed D. This gap will ensure that rates that are more than s away from each other are mapped to distinguishable delays. The difference in delays for our rate-delay function is:

$$\left(R_m + \frac{\alpha}{\mu}\right) - \left(R_m + \frac{\alpha}{s\mu}\right) > D$$
$$\Rightarrow \mu < \frac{\alpha}{D}\left(1 - \frac{1}{s}\right).$$

This gives us μ_+. μ_- is the rate corresponding to $d = R^{\max}$, so $\mu_- = \alpha/(R^{\max} - 2R_m)$. Hence, for the Vegas family,

$$\frac{\mu_+}{\mu_-} = \frac{R^{\max} - R_m}{D}\left(1 - \frac{1}{s}\right) = O\left(\frac{R^{\max}}{D}\right). \quad (1)$$

We propose an alternate rate-delay function that does much better:

$$\mu(d) = \mu_- s^{\frac{R^{\max} - d}{D}} \quad (2)$$

When $d = R^{\max}$, $\mu = \mu_-$ as desired. μ_+ occurs when $d = R_m + D$. This is the minimum RTT required to ensure full utilization (Theorem 2). Hence, $\mu_+ = \mu_- s^{(R^{\max} - (R_m + D))/D}$. Thus,

$$\frac{\mu_+}{\mu_-} = s^{\frac{R^{\max} - R_m - D}{D}} = O\left(s^{\frac{R^{\max}}{D}}\right).$$

This range is exponentially larger than the Vegas family and can span several orders of magnitude of link rates, as desired. For instance, for $D = 10$, $s = 2$ and $R^{\max} = 100$ ms we can support a range of $2^{10} \approx 10^3$. With $s = 4$, that increases to $2^{20} \approx 10^6$.

A real-world CCA may make the following modifications to this rate-delay curve. (1) This function never increases its rate beyond μ_+. This is simply solved by using a Vegas-like

function for $d < D$ that goes to infinity. Such a CCA will scale to arbitrarily large link rates, but risk starvation when rate exceeds μ_+. (2) If the CCAs have a method to estimate R_m, they can set $R^{\max}$ as a function of R_m: e.g. $R^{\max} = R_m + 100$ ms. Note that both Equation 2 and the Vegas family can send at rates lower than μ_-, but will increase delay beyond $R^{\max}$. For rates $< \mu_-$, delays increase more slowly for the Vegas family than for Equation 2.

Algorithm 1 A delay-convergent CCA that uses $\mu_- s^{\frac{R^{\max}-(d-R_m)}{D}}$. The following is run every R_m. Here, d is the latest measured RTT, μ is the current sending rate and $a, 0 < b < 1, \mu_-, R^{\max}$ are parameters of the algorithm.

```
if μ < μ_- s^((R^max - (d - R_m))/D) then
    μ ← μ + a
else
    μ ← bμ
end if
```

An Algorithm. Algorithm 1 shows a CCA that uses Equation 2. This algorithm is incomplete on several fronts. It does not feature a mechanism to discover R_m or handle short buffers by slowing down in the presence of loss. It increases its rate additively, and does not feature the faster increase times of modern algorithms. It does not have a cwnd cap to be resilient to sudden drops in link capacity [4]. We show this merely to illustrate an idea, not propose a deployable CCA.

To verify this algorithm, we used CCAC to produce traces where the algorithm is either inefficient or more than s-unfair. CCAC was unable to produce such traces, giving us some confidence that the key ideas work. CCAC helped us fine-tune some details of the algorithm such as (a) use AIMD instead of the AIAD used by Vegas and Copa because the fairness properties of AIMD are critical in the presence of measurement ambiguity and (b) change the rate by the same amount every RTT independent of the number of ACKs received. Note that this does not constitute a complete proof, since CCAC only searched over finite traces. We have not yet performed the steady-state analysis described in the CCAC paper.

Estimating R_m is a challenge for any delay-convergent CCA, which may require fundamental new insights from the community to overcome. Estimating R_m is hard because it requires all flows to coordinate and empty the queue at the same time. Copa's mechanism works well in the absence of delay ambiguity, but not otherwise. BBR's mechanism works when there is a single flow, but its RTT probe does not always succeed in coordinating across multiple flows.

6.4 Explicit Signaling

Some routers set ECN bits [14] in packets when they detect congestion. Unlike delay and loss, which can occur for reasons other than congestion, ECN is an unambiguous signal of congestion. Hence ECN may help CCAs avoid starvation. As evidence, consider our analysis of NewReno in Section 5.4. CCAC indicated that when non-congestive delay jitter is present, but losses only occur due to buffer overflow, AIMD does not starve. When non-congestive loss is present, AIMD, Cubic and PCC Allegro all suffer starvation. If the router set ECN bits when the queue exceeds a threshold, and a CCA reacted to that and not to small amounts of loss, then it may avoid starvation. AQM mechanisms are more sophisticated than the simple threshold-based heuristic discussed above [16, 33, 35]. We conjecture that such AQM mechanisms, coupled with CCAs that ignore small amounts of loss, can prevent starvation.

6.5 An Absolute Upper Bound

We have also proved that *no* deterministic CCA can be simultaneously f-efficient, delay-bounding (but not delay-convergent, i.e., the delay can have large oscillations over ideal links), and starvation-free. This theorem uses a stronger network model than in Section 3; here the adversary can also vary the link rate arbitrarily. We call this the "strong" model. Since there are no bounds on the link rates, this adversary is very powerful. Perhaps too powerful, for it can create unrealistic networks. Thus we still believe that it may be possible to achieve all three properties on *practical* networks. Nevertheless, it serves as a useful upper bound on what is possible. The proof technique is interesting in that we have found it instructive to study the network paths it constructs. It often constructs paths similar to ones constructed by Theorem 1, i.e., consistent with our simpler network model.

Theorem 3. *Any deterministic, f-efficient, delay-bounding CCA will starve in the strong model for any value of the propagation delay R_m.*

The proof is given in Appendix B. This theorem does not need to control the initial conditions or require both CCAs to be the same.

7 RELATED WORK

Congestion control has been subject to extensive theoretical analysis using both deterministic and statistical models of the network [11, 27, 39]. Almost all CCAs, either implicitly or explicitly, converge to some fixed point. This fixed point may be expressed in terms of the loss rate, delay, or sending rate, or ACK arrival rate. Many papers study these fixed points. Some assume there is a single bottleneck [3, 6, 8, 12]

while others generalize to networks with multiple bottlenecks [37, 44]. The network utility maximization (NUM) framework shows that CCAs can often optimize a global utility function [30, 34]. There has also been work in understanding how different CCAs can coexist, either with a goal to get them to share bandwidth equitably [3, 15, 19, 20, 43] or to deliberately cause unfairness to give different priorities to different flows [28, 31, 38].

Prior work has analyzed the ability of delay and ECN to create unique fixed points that flows can converge to and their stability once converged [40, 51]. These papers prove, for instance, that a CCA that converges to a time-invariant delay that does not change with the number of senders cannot be fair. Another body of work examines which CCA properties can be achieved simultaneously [49, 50]. Prior work has analyzed CCAs on non-ideal paths with packet loss, but to our knowledge this is the first paper with a theoretical analysis of paths with non-congestive delay jitter.

Empirical studies have noted unfairness that is higher than expected [36], including BBR's extreme RTT unfairness [21]. By contrast to the recent CCAC paper [2], we analyze and prove results about multiple flows using different proof techniques and a simpler network model (i.e., the network model here is capable of doing less than CCAC's, so our impossibility result holds in the CCAC model too). We became interested in proving the impossibility result of this paper when we failed to find any CCA that CCAC's automated verifier could not break.

8 CONCLUSION

This paper identified a surprising problem with a design pattern used by most (if not all) delay-bounding CCAs. We proved that all delay-convergent CCAs are susceptible to starvation and characterized the conditions under which starvation is inevitable.

We offer three key conclusions for CCA designers, who should model (or better estimate) non-congestive delays explicitly in delay-convergent CCAs. First, to utilize the link efficiently, a CCA must maintain a queue that is larger than the non-congestive delay on the path; second, this alone is not enough to avoid starvation, but in addition the variation in the queueing delay in steady state must also be greater than one-half of the delay jitter; and third, if we have a prior upper bound on sending rate, we may be able to avoid starvation while also reducing the queueing delay variation.

It is also possible that purely end-to-end CCAs might always suffer from the issues we found, and in-network support such as active queue management, explicit congestion signaling, or stronger isolation is required. At any rate, we think new ideas are needed here to sidestep our starvation result. Section 6 presented ideas that may be useful in this task.

This work does not raise any ethical issues.

ACKNOWLEDGMENTS

We thank Anup Agarwal, Rahul Bothra, Miguel Ferriera, Prateesh Goyal, Manya Ghobadi, Zili Meng, Radhika Mittal, Akshay Narayan, Sudarsanan Rajasekaran, Ahmed Saeed, Keith Winstein, the SIGCOMM 2022 reviewers, and Steve Uhlig (the paper's shepherd) for their feedback on this work. The NASA University Leadership Initiative (grant #80NSSC20M0163) provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not any NASA entity. This work was also partially funded by NSF award #1751009.

REFERENCES

[1] Accessed 2021. The ns-3 simulator. https://nsnam.org/. (Accessed 2021).
[2] Venkat Arun, Mina Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. 2021. Toward Formally Verifying Congestion Control Behavior. In *SIGCOMM*.
[3] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *NSDI*.
[4] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. 2001. Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms. In *SIGCOMM*.
[5] R Braden. 1989. Requirements for Internet Hosts – Communication Layers. *IETF* (1989). RFC 1122, Section 4.2.3.2.
[6] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. 1994. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM*.
[7] Bob Briscoe. 2007. Flow Rate Fairness: Dismantling a Religion. *ACM SIGCOMM Computer Communication Review* 37, 2 (2007), 63–74.
[8] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-based congestion control. In *ACM Queue*. 58–66.
[9] Neal Cardwell, Yuchung Cheng, S. Hassas Yeganeth, Ian Swett, and Van Jacobson. 2017. BBR Congestion Control, version 1. IETF Internet Draft. (2017). https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-00#section-4.2.3.2 Section 4.2.3.2.
[10] Neal Cardwell, Yuchung Cheng, S. Hassas Yeganeth, Ian Swett, and Van Jacobson. 2021. BBR Congestion Control, version 2. IETF Internet Draft. (2021). https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-01#section-4.6.4.2 Section 4.6.4.2.
[11] D-M. Chiu and R. Jain. 1989. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems* 17, 1–14.
[12] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. 2015. PCC: Re-architecting Congestion Control for Consistent High Performance. In *NSDI*.
[13] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-learning Congestion Control. In *NSDI*.
[14] Sally Floyd. 1994. TCP and Explicit Congestion Notification. *ACM SIGCOMM CCR* 24, 5 (1994), 8–23.

[15] S. Floyd, M. Handley, J. Padhye, and J. Widmer. 2000. Equation-Based Congestion Control for Unicast Applications. In *SIGCOMM*.
[16] Sally Floyd and Van Jacobson. 1993. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Trans. on Networking* 1, 4 (1993), 397–413.
[17] Jim Gettys. 2011. Bufferbloat: Dark Buffers in the Internet. *IEEE Internet Computing* 15, 3 (2011), 96–96.
[18] Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. 2020. ABC: A Simple Explicit Congestion Controller for Wireless Networks. In *NSDI*.
[19] Prateesh Goyal, Akshay Narayan, Frank Cangialosi, Srinivas Narayana, Mohammad Alizadeh, and Hari Balakrishnan. 2022. Elasticity detection: A building block for internet congestion control. In *SIGCOMM*.
[20] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review* 42, 5 (July 2008), 64–74.
[21] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental evaluation of BBR congestion control. In *ICNP*.
[22] Janey C Hoe. 1996. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *ACM SIGCOMM 1996*. 270–280.
[23] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport. *IETF* (2021). RFC 9000.
[24] Van Jacobson. 1988. Congestion Avoidance and Control. In *SIGCOMM*.
[25] Nathan Jay, Tomer Gilad, Nogah Frankel, Tong Meng, Brighten Godfrey, Michael Schapira, Jae Won Chung, Vikram Siwach, and Jamal Hadi Salim. 2018. A PCC-Vivace Kernel Module for Congestion Control. (2018).
[26] Linux 2021. Linux Networking Documentation/Segmentation offloads. (2021).
[27] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. 1997. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM SIGCOMM CCR* 27, 3, 67–82.
[28] Tong Meng, Neta Rozen Schiff, P Brighten Godfrey, and Michael Schapira. 2020. PCC Proteus: Scavenger Transport and Beyond. In *SIGCOMM*.
[29] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. In *SIGCOMM*.
[30] Kanthi Nagaraj, Dinesh Bharadia, Hongzi Mao, Sandeep Chinchali, Mohammad Alizadeh, and Sachin Katti. 2016. Numfabric: Fast and Flexible Bandwidth Allocation in Datacenters. In *SIGCOMM*.
[31] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. 2019. End-to-end transport for video QoE fairness. In *SIGCOMM*.
[32] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX ATC*.
[33] Kathleen Nichols, Van Jacobson, Andrew McGregor, and Jana Iyengar. 2018. *Controlled Delay Active Queue Management*. Technical Report.
[34] Daniel Pérez Palomar and Mung Chiang. 2006. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications* 24, 8 (2006), 1439–1451.
[35] Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Suryanarayana Prabhu, Vijay Subramanian, Fred Baker, and Bill VerSteeg. 2013. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In *14th International Conf. on High Performance Switching and Routing (HPSR)*.
[36] Adithya Abraham Philip, Ranysha Ware, Rukshani Athapathu, Justine Sherry, and Vyas Sekar. 2021. Revisiting TCP congestion control throughput models & fairness properties at scale. In *IMC*.
[37] Jordi Ros-Giralt, Noah Amsel, Sruthi Yellamraju, James Ezick, Richard Lethin, Yuang Jiang, Aosong Feng, Leandros Tassiulas, Zhenguo Wu, Min Yeh Teh, et al. 2021. A Quantitative Theory of Bottleneck Structures for Data Networks. *IEEE Transactions on Networking (under review)* (2021).
[38] Sea Shalunov, Greg Hazel, Janardhan Iyengar, Mirja Kuehlewind, et al. 2012. Low extra delay background transport (LEDBAT). In *RFC 6817*.
[39] R. Srikant. 2004. *The Mathematics of Internet Congestion Control*. Springer Science & Business Media.
[40] Mohit P Tahiliani, Vishal Misra, and KK Ramakrishnan. 2019. A Principled Look at the Utility of Feedback in Congestion Control. In *Workshop on Buffer Sizing*.
[41] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. 2006. A Compound TCP Approach for High-speed and Long Distance Networks. In *INFOCOM*.
[42] The Google BBR Team. 2018. BBR bandwidth based convergence. https://github.com/google/bbr, commit 87d8587c50, Documentation/bbr_bandwidth_based_convergence.pdf. (2018).
[43] 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 *HotNets*. 17–24.
[44] D.X. Wei, C. Jin, S.H. Low, and S. Hegde. 2006. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Trans. on Networking* 14, 6 (2006), 1246–1259.
[45] Keith Winstein and Hari Balakrishnan. 2013. TCP ex Machina: Computer-Generated Congestion Control. In *SIGCOMM*.
[46] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *NSDI*. 459–471.
[47] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. 2018. Pantheon: the training ground for Internet congestion-control research. In *USENIX ATC*.
[48] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In *SIGCOMM*.
[49] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. 2017. An Axiomatic Approach to Congestion Control. In *HotNets*.
[50] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. 2019. Axiomatizing Congestion Control. *ACM POMACS* 3, 2 (2019), 1–33.
[51] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. 2016. ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY. In *CoNEXT*.

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

A PROOF OF STARVATION OF DELAY-CONVERGENT ALGORITHMS

We will now fill in the details of the proof sketch discussed in section 4. The theorem is restated here for convenience:

THEOREM 1 For any deterministic, f-efficient, delay-convergent CCA $\mathcal{A}$, any propagation delay R_m, any throughput ratio $s \geq 1$, and any $D > 2\delta^{\max}$, there exists a network scenario with two flows (specified via two per-flow initial states and trajectories of non-congestive delays), such that one flow gets a throughput x_1 and the other flow gets a throughput $x_2 \geq s \cdot x_1$.

Proof. Steps 1 and 2 of the proof in Section 4 are complete. That section omitted details from step 3, which we now fill.

Recall that in step 2, we created two *different* ideal links where the flows running by themselves will achieve throughputs that are more than a factor s different. The key is that the delay the flows experience both lie within a range of size $\delta^{\max} + \epsilon$. Now we will run both these flows on the *same* FIFO queue as in our model. We will pick the starting states for the two flows to be the same as the state after they have converged, that is their state at times T_1 and T_2 respectively. Next we pick the initial queue length and vary the non-deterministic per-flow delay such that the delay they experience is the same as they experienced in the one-flow case. Since the CCA is deterministic, their sending rates will be identical. It remains for us to show we can indeed recreate the same delay.

Let $\epsilon = D/2 - \delta^{\max} > 0$. There are two cases based on how $\min(d_{\min}(C_1), d_{\min}(C_2))$ compares with $R_m + \delta^{\max} + \epsilon$.

Case 1: $\min(d_{\min}(C_1), d_{\min}(C_2)) > R_m + \delta^{\max} + \epsilon$. In this case we will run both flows on a common link with propagation delay R_m and bottleneck link rate $C_1 + C_2$. When flow $i \in \{1, 2\}$ is running on an ideal path by itself, the derivative of the delay it experiences is $\bar{d}_i'(t) = \frac{\mathrm{d}\bar{d}_i(t)}{\mathrm{d}t} = (\bar{r}_i(t) - C_i)/C_i$. This is because $\bar{d}_i(t) \geq d_{\min}(C_i) > R_m$ and the queue is never empty. Let us calculate the queuing delay at a link with capacity $C_1 + C_2$ when packets from the two flows arrive at rates $\bar{r}_1(t)$ and $\bar{r}_2(t)$. Let $d^\star(t)$ be the delay experienced in the combined two-flow network. If $d^\star(t) > R_m$, we have:

$$\frac{\mathrm{d}d^\star(t)}{\mathrm{d}t} = \frac{\bar{r}_1(t) + \bar{r}_2(t) - (C_1 + C_2)}{C_1 + C_2} \tag{3}$$

$$= \bar{d}_1'(t)\frac{C_1}{C_1 + C_2} + \bar{d}_2'(t)\frac{C_2}{C_1 + C_2} \tag{4}$$

Hence $\frac{\mathrm{d}d^\star(t)}{\mathrm{d}t}$ is a weighted average of $\bar{d}_1'(t)$ and $\bar{d}_2'(t)$ with weights C_1 and C_2 respectively. Since we are allowed to set the initial conditions, we set the initial queue length:

$$d^\star(0) = \bar{d}_1(0)\frac{C_1}{C_1 + C_2} + \bar{d}_2(0)\frac{C_2}{C_2 + C_2} - \delta^{\max} - \epsilon$$

Since we assumed $d_1(t), d_2(t) > \delta^{\max} + \epsilon$ in our case analysis, subtracting $\delta^{\max} + \epsilon$ still leaves us with $d^\star(0) > R_m$ and Equation (3) applies. The above equation continues to hold for all t, as $d^\star$ continues to follow (3) by induction over t.[11] Hence:

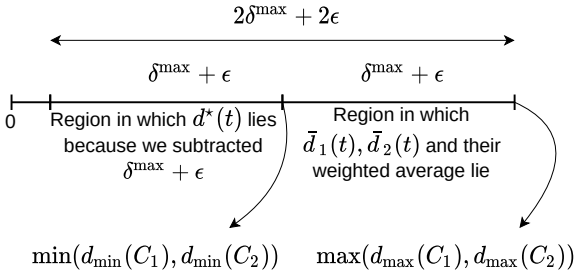
$$d^\star(t) = \bar{d}_1(t)\frac{C_1}{C_1 + C_2} + \bar{d}_2(t)\frac{C_2}{C_1 + C_2} - (\delta^{\max} + \epsilon) \tag{5}$$

[11] t is real and hence cannot support induction. However if we discretize time into infinitesimally small pieces, we can apply induction at each step.

Having derived the above expression, we have completed one of the tasks we had left unfinished in Section 4. The other task we had deferred in Section 4 was to prove that we will always be able to emulate delay. To do this, we deferred proving (1) $d^\star(t) \leq \min\{\bar{d}_1(t), \bar{d}_2(t)\}$ and (2) $\max\{\bar{d}_1(t), \bar{d}_2(t)\} \leq d^\star(t) + D$. This is equivalent to:

$$0 \leq \bar{d}_i(t) - d^\star(t) \leq D = 2\delta^{\max} + 2\epsilon$$

for $i \in \{1, 2\}$. Note that both $\bar{d}_1(t)$ and $\bar{d}_2(t)$ lie in a common region of size $\delta^{\max} + \epsilon$ for a given t. Therefore so does their weighted average, $(d^\star(t) + \delta^{\max} + \epsilon)$. The fact that they all lie in this region implies that the gap between any two of the three quantities is $\leq \delta^{\max} + \epsilon$. Thus, the $-(\delta^{\max} + \epsilon)$ term in Equation (5) brings $d^\star(t)$ below the minimum value of both $\bar{d}_i(t)$. Hence $d^\star(t) < \bar{d}_i(t)$ as required. Further, $\bar{d}_i(t) \leq d^\star(t) + D$ (note, $D = 2\delta^{\max} + 2\epsilon$). The reason is illustrated in the diagram below:



Thus we can emulate the two-flow network to make each flow think they are in ideal links of widely different capacities. Hence starvation will ensue.

Case 2: $\min(d_{\min}(C_1), d_{\min}(C_2)) \leq R_m + \delta^{\max} + \epsilon$. This is the easy case. Since both the delays lie within an interval of size $\delta^{\max} + \epsilon$, we have that $\bar{d}_1(t), \bar{d}_2(t) \leq R_m + 2\delta^{\max} + 2\epsilon = R_m + D$. This means that if the queuing delay in the two-flow bottleneck were always 0, the non-deterministic delay element alone can emulate both the delays. Hence we simply pick a link rate that is large enough that $d^\star(t) \leq \bar{d}_i(t)$ for $i \in \{1, 2\}$.

Note, in this case we can prove something stronger than starvation; the CCA isn't even f-efficient in our network model (though it is f-efficient for an ideal path with zero non-congestive delay). We can have a very large link rate, emulate $\bar{d}_1(t)$ entirely using non-congestive delay and induce the CCA to transmit at rate $\leq C_1$. Since the link rate can be arbitrarily large, this causes arbitrarily bad underutilization. This was the content of Theorem 2. □

B PROOF OF THE ABSOLUTE UPPER BOUND

We re-state and prove the theorem discussed in Section 6.5.

Theorem 3 Any deterministic, f-efficient, delay-bounding CCA will starve in the strong model for any value of the propagation delay R_m.

Proof. We will construct a sequence of single-flow network traces that will eventually let us construct a two-flow trace that causes starvation. We pick an arbitrary rate λ. Then, for our first trace, we run the CCA on an ideal link with rate λ and propagation delay R_m. Let the delay and sending rate in this case be $d_1(t)$ and $r_1(t)$ respectively. Let $D = \max_{t\in[0,\infty)} d_1(t)$ be the maximum delay experienced.[12]

Note that by varying the link rate, the adversary can create *any* queuing delay pattern it likes. This is because it can delay every packet by any amount it likes. Since it is a FIFO queue, it cannot reorder packets. Hence it cannot preferentially send packets of one flow over the other; both flows experience the same delay at the queue. Thus the theorem statement is not vacuously true.

We construct the next single-flow behavior by causing the queuing delay to be $d_2(t) = \max(0, d_1(t) - D)$. If the ratio of throughputs between the first and second case is more than s or less than $1/s$ infinitely many times (according to the theorem statement), we are done. We can run the two flows on the same FIFO queue where the link causes a queuing delay of $d_1(t) - D$. Then the non-deterministic delay element adds D seconds of delay to one flow's packets and 0 seconds of delay to another flow's packets. Since the flows see exactly the same delays as they in the single-flow case, $d_1(t)$ and $d_2(t)$, they behave exactly the same way. Hence they achieve throughputs that are more than a factor s apart.

If not, we construct a third trace where the queuing delay is $\max(0, d_2(t) - D)$. If the throughputs of the second and third trace differ by a ratio of more than s, again we are done. Else we continue on. In at most $n = \lceil Q/D \rceil$ such steps, we would have either succeeded in causing starvation or reached $d_n(t) = 0$.

We claim that because the CCA is f-efficient, when $d_n(t) = 0$, the throughput should increase to infinity. That is, for all times t and rate λ', there exists a time $t' > t$ such that the total number of bytes transmitted is greater than $t'\lambda'$. This is because, for a sufficiently large link rate, $d_n(t) = 0$. At this link rate, if the throughput achieved by the CCA is finite, the CCA can under-utilize by an arbitrary amount as the link rate increases. Here, by "finite" we mean that there exists a λ' such that for all times t, the number of bytes transmitted till t is less than $\lambda' t$. This violates our f-efficiency definition.

Now if the throughput for the n^{th} trace reaches infinity, at some point in between the ratio of must have been greater than s. □

[12] Strictly speaking, we should use the least upper bound, since the maximum may not exist.

C CCAC EXTENSION

To study starvation, we extended CCAC [2] to handle multiple flows. This turned out to be fairly straightforward. CCAC tracks, among other things, the number of bytes that have arrived at its bottleneck $A(t)$, and the number of bytes that have been served from it $S(t)$. To extend it to multiple flows, we have to maintain separate functions, one for each flow. So we have $\sum_i A_i(t) = A(t)$ and $\sum_i S_i(t) = S(t)$.

At time t, when a total of $S(t)$ bytes have been served, we need to determine how many bytes have been served *per-flow*. Ideally, we'd like to emulate a FIFO queue. Let t' be the time at which $S(t) = A(t')$. Then we'd want $S_i(t) = A_i(t')$, because that is when those packets of that flow must have entered the queue. Doing this directly requires us to intersect two lines, the equation for which involves the multiplication of two SMT variables. As a general rule, SMT solvers tend to not be very good at solving for non-linear constraints. We found that this rule applies to CCAC as well.

To get around this, we used the same relaxation method proposed in the CCAC paper. CCAC discretizes time relatively coarsely. It then ensures that the set of behaviors admitted in the discrete model is a super-set of the behaviors admitted in the continuous model. This way, any theorems proved in the discrete model also hold in the continuous model. However the discrete model may contain behaviors that the continuous model does not. Hence one must be careful to not make the discrete set *too* large.

We found the following approach to strike a good balance between ease of SMT modelling and not deviating too far from the continuous model. We merely ensured that if the queuing delay at time t is $\mathrm{d}t$ per CCAC's definition, then $S_i(t) > A_i(t - \mathrm{d}t)$.