MAKING VIDEO TRAFFIC A FRIENDLIER INTERNET NEIGHBOR

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Bruce Spang
June 2023

This dissertation is online at: https://purl.stanford.edu/tt015dr9566

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Nick McKeown, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Ramesh Johari

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Keith Winstein

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format.

Abstract

Streaming video traffic from services like Netflix and YouTube accounts for the vast majority internet traffic today—recent estimates put the fraction as high as 60-75% of all bytes sent over the internet. Given video traffic is such a large fraction of the internet, this thesis makes video traffic friendlier to neighboring applications that share the same networks.

The thesis begins with Sammy, a system that smooths out video traffic so that the throughput of a video session is close to the minimum the video session needs for good performance. By judiciously picking throughput based on the needs of the video application, Sammy is able to substantially smooth out video traffic. In internet-scale experiments, Sammy reduces throughput of video traffic by more than half and dramatically reduce congestion, all while slightly improving video quality of experience over today's top production algorithms.

The thesis next considers how new algorithms that affect congestion on the internet (like Sammy) are evaluated in networking research today. The gold standard is to run large-scale A/B tests, to understand at how an algorithm actually performs in practice. I show in experiments run at scale that typical A/B tests can lead to biased results, even to the point of switching the direction of results—an algorithm that appears worse in an A/B test might actually improve performance when deployed, and vice versa. I discuss the implications of this and suggest how researchers can deal with this bias.

Finally, I revisit the long-standing problem of sizing buffers in routers. Prior work suggested that buffers can be reduced by a factor of square root of the number of flows using the router, offering dramatic buffer reductions in networks carrying many flows. I revisit these results, removing assumptions and showing that they hold for modern congestion control algorithms. Importantly, I discuss how our approach of smoothing video traffic with Sammy challenges the assumptions of the classic buffer sizing problem, potentially allowing for a future with much smaller buffers.

Overall, this thesis invites us to reconsider how we deal with network congestion at scale. Rather than focusing on congestion control algorithms that maximize throughput, we should prioritize the ways we actually use the internet—applications like video streaming and web browsing and gaming and so on. If we do so, we can reduce congestion and improve the performance of all applications that share the internet.

Acknowledgments

Firstly, thank you, the reader, for reading my thesis! I poured so much into this work over the past six years and it's very gratifying that you're spending the time to read any of it. I love talking about this work, so if you have questions or just want to chat, please get in touch at bruce@brucespang.com.

This thesis would not have happened without a huge amount of support from many, many people over the past six years. My primary advisor has been Nick McKeown, who continually pushed my research to make it as impactful as possible, and has an incredible ability to balance the nitty gritty details that make something work with the overall reason that the work matters. I was also advised by Ramesh Johari, who understands things so well and always gets right to the heart of anything we talk about. Most of the thesis, especially the chapter on experiments, could not have happened without his guidance. Finally, I spent the last three years of my PhD as a graduate research fellow at Netflix and was also advised by my manager, Te-Yuan Huang. TY gave incredible advice that kept the research impactful to industry, and has a huge depth of knowledge about what makes video work well. She also made so much of the research possible by building organizational support for running experiments around these fairly academic questions with production Netflix traffic.

I also wanted to thank all my co-authors. In the McKeown group at Stanford, I was so lucky and had so much fun working with Serhat Arslan and Sundararajan Renganathan. At Netflix, I worked with a great team including Shravya Kunamalla, Renata Teixeira, Veronica Hannan, Brady Walsh, Tom Rusnock, and Joe Lawrence. The first few years of my PhD I worked in the theory group at Stanford, which made all the math I ran into later in the PhD feel very approachable. In the theory group I especially wanted to thank Mary Wootters, Joshua Brakensiek, Ray Li, Margalit Glasgow, Saba Eskandarian, and Li Yang Tan. I also got some incredible IT support with all the lab experiments in this research from Joseph Little at Stanford.

So much of the past six years, including this thesis, would have been so much worse without my family and friends. To my family: Aletha Spang, Laurence Spang, Susan Marsh; and to to my friends: Alan Figot, Alex Ortiz, Aurora Alvarez-Buylla, Avery Hill, Barrett Redmond, Dania Nanes, Daniel Nelli, David Armenta, Elani Gitterman, Jacob Wisser, Leandra Brickson, Logan Steinharter, Lucas Pavan, Maria Viteri, Masha Karelina, Michael Montgomery, Nathan Wolf, Sedona Murphy, Sigbrit Søchting, and TJ Wilkason. I love you all so much.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

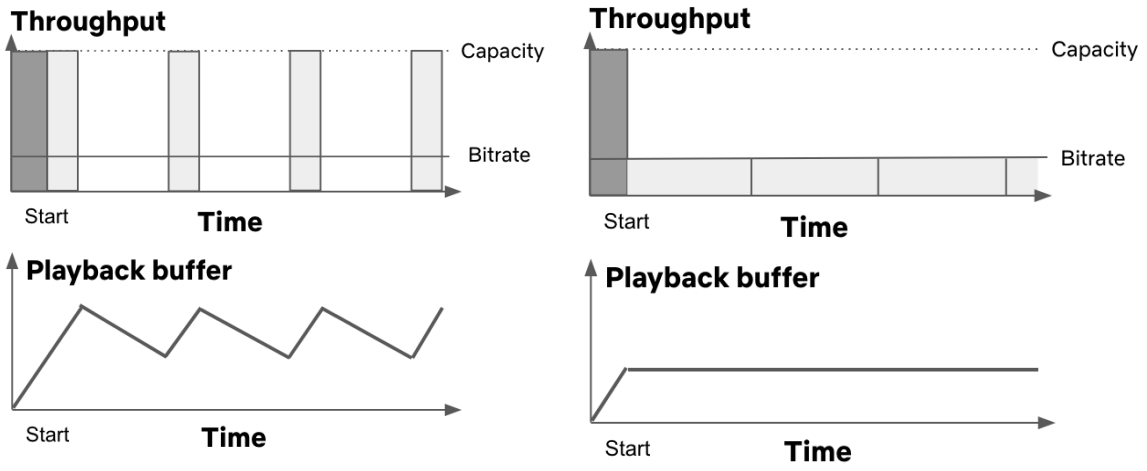
On-demand streaming video from services like Netflix and YouTube currently comprises 50-75% of internet traffic [152]. With a single application having such a large volume of traffic, as a community we should closely examine how it uses the network and ensure that it is as good a neighbor as possible. If we do so, we will improve the internet for all applications that share it.

Video traffic today has the on-off, bursty traffic pattern shown in Figure 1.1a: every few seconds, video switches between an on period of sending data as quickly as possible and an off period of silence. This well-known phenomenon [147] arises from a mismatch between the *bitrate* (the number of bits watched over the time the video plays) and the *throughput* (the number of bits downloaded over the time the video is downloading).

Congestion control algorithms [141] choose the throughput for video traffic, aiming to maximize throughput without congesting the network. The bitrate for video traffic is chosen by Adaptive Bitrate (ABR) algorithms. Video is split up into "chunks" of a few seconds each, and each chunk is encoded at a number of different bitrates: from a higher quality, larger version, to a lower quality, smaller version. An ABR algorithm selects the bitrate of each chunk to ensure high-quality, uninterrupted playback.

Whenever throughput of a chunk is higher than the bitrate of a chunk, the video downloads faster than it is played back. To allow smooth video playback despite this mismatch, the chunk is temporarily stored on the player, in a playback buffer to be played later. During on periods when throughput is higher than bitrate, the playback buffer grows. Once the buffer is full, video traffic pauses downloading and creates the off periods shown in Figure 1.1a.

Over the past decade, video traffic has been getting more bursty: on periods are getting shorter, and off periods are getting longer. A decade ago, home access speeds and video bitrates were both on the order of a few megabits per second [42, 108], and so throughput and bitrate were similar and video traffic spent most of its time in on periods [147]. Since then, median home network speeds worldwide have improved to tens or hundreds of megabits per second [167, 43]. Video encoding has



(a) Video traffic today.

(b) Smoother, same QoE.

Figure 1.1: A few seconds of a typical streaming video session. Today, video has on periods (light grey) where it sends data as fast as the network supports (a). But as shown in (b), we can smooth throughput and reduce congestion without impacting video QoE.

also improved: today a 1080p video requires only a few megabits per second [108] and a 4k video requires typically less than twenty megabits per second [129]. Videos download faster, playback buffers fill up faster, and video traffic has shorter, faster on periods.

In this thesis, I will argue that despite this trend, *video traffic should not be bursty.* I will show that if we smooth out video traffic (the majority use of the internet today) we can improve performance for neighboring applications sharing the same network. Moreover, video streaming services are incentivized to smooth out video traffic to maximize their *own* performance. Smoothing out video traffic is good for everybody.

This thesis is organized into three parts. I begin by going into more detail about why video traffic should be smoother and introduce an algorithm, Sammy, to smooth out video traffic. I next look at how to run experiments with algorithms like Sammy that affect congestion on the internet. Finally, I discuss sizing buffers in internet routers (a classic problem in networking) and discuss how our approach to smoothing video traffic fundamentally changes this problem.

1.1 Sammy: an algorithm to smooth video traffic

Bursty video traffic is undesirable to neighboring traffic sharing the same network. During on periods, video traffic uses congestion control algorithms like TCP Reno [97], Cubic [78], or BBR [33] to pick a throughput. The goal of these algorithms is to maximize throughput, while simultaneously avoiding congestion. Designing congestion control algorithms that work well has been a long-standing problem in networking, and there are consequences to sending as fast as possible with the algorithms we do

have, including packet loss and queueing delay, bufferbloat [69], and unfairness between TCP flows [17, 2, 191, 28, 118, 13, 158, 94, 51, 190, 30, 93, 105, 182, 183, 101].

As an alternative, consider what would happen if we were to reduce throughput *below* network capacity as shown in Figure 1.1a. We would avoid congestion completely. There would be *no* queueing delay or packet loss. A short HTTP request issued during an on period could complete faster with more available bandwidth and lower queueing delay. A longer-lived video conferencing flow would see more consistent throughput and delay. Reducing burstiness should benefit all other traffic on the internet. This is a natural idea, and smoothing video traffic has been suggested by prior work [70, 2, 4, 126, 154, 22, 122].

The major challenge in reducing the throughput for video traffic (and what has not been considered by prior work) is doing so without reducing the performance of video traffic. Video performance is known as *Quality of Experience* (QoE) and is measured by three components: *video quality* (how good the video looks), *play delay* (how long the video takes to start playing), and *rebuffers* (times when the video playback is interrupted because data is not available). Video QoE is important for the experience of the people watching the video [50, 115, 199], and therefore also important to video streaming services. We would not want to smooth video traffic—the majority use of the internet—by making its users suffer.

But video can be made less bursty without reducing QoE. As an example, consider Figure 1.1b. We have taken the exact same video streaming trace and smoothed it out. We have reduced the throughput[1] to the level of the video bitrate—well below the network capacity. From the end user's perspective, the quality of experience (QoE) of the video is identical: quality is the same (the same number of bytes are downloaded in the same amount of time), the play delay is the same (the width of the dark gray box is the same), and there are no rebuffers (the buffer never empties).

Although this example shows that video traffic can be smooth in theory, there are a number of challenges in designing an algorithm to smooth video traffic in practice. I go into detail about the challenges and ways of solving them in Chapter 2. I propose an algorithm called *Sammy*,[2] which smooths out video traffic without reducing QoE, and work with Netflix to implement and evaluate this algorithm at scale. In large-scale experiments run at Netflix, Sammy substantially smooths video traffic: at the median, lowering throughput by more than half, improving congestion, and even *slightly* improving video QoE.

Streaming video services are incentivized to use approaches like Sammy to smooth video traffic. First, neighboring traffic could easily be from the same video service. By improving performance for its neighbors, Sammy improves performance for the video service itself. Second, Sammy forces an ABR algorithm to be careful about its use of throughput estimates. This exercise gave us a slight QoE improvement, and could yield larger improvements for other ABR algorithms.

[1] Note that we have defined throughput here as the throughput during on periods

[2] As of this writing, Sammy is the current reigning world's fastest snail [145].

1.2 Experimenting at scale with algorithms like Sammy

Sammy's evaluation, like that of most new networking algorithms, heavily relies on experiments called A/B tests. In an A/B test, the experimenter randomly allocates a small fraction of traffic (say 1% or 5%) to a new algorithm, called the treatment group, and compares its performance against the control group running the old algorithm. A/B tests are widely used as the gold standard for understanding how a new algorithm will behave at scale. Almost all large tech companies routinely use A/B tests to evaluate changes before deploying them [113, 176, 120, 34, 73, 48, 104, 157, 138]. Networking research often includes the results of A/B tests, and uses them to justify new algorithms [96, 36, 161, 34, 33, 138, 117, 53, 63, 52, 120, 104, 92, 127, 193, 119].

Imagine running an A/B test with Sammy, in which there are two video sessions that share the same network: one treatment and one control. Imagine that by smoothing the treatment session, we are able to improve performance of the control neighbor—perhaps the control quality improves while treatment quality remains the same. In this example, using Sammy would be an improvement overall. But if we were to analyze the results as an A/B test, Sammy would appear to perform worse: we would compare treatment to control, and Sammy has lower quality than the newly improved control.

This confusion is caused by *interference*, when units in the treatment group interact with units in the control group. It is well known in causal inference that interference can bias experiment results [95]. In social networks, changing something for a user in the treatment group can impact the behavior of their friends in the control group and bias the results of an experiment [58]. In online marketplaces, increasing the price of items in a treatment group can increase the demand for the relatively cheaper items in the control group and bias results [88]. There are many examples of interference bias from markets, education, disease, and more [109, 44, 80, 89].

In networking experiments, packets from treatment and control groups use the same networks, traversing the same links and the same queues. There is a long line of networking research showing that algorithms compete with each other when sharing a congested network [158, 94, 189, 190, 30, 93, 182, 183, 17, 2, 191, 28, 118, 13, 51, 105]. If an algorithm like Sammy frees up bandwidth, the control traffic could take up that bandwidth and get better performance. This could make Sammy look worse than it would if the uncapped traffic were not present, even if it was improving congestion.

In Chapter 3, I show that interference exists not only in this hypothetical experiment, but also biases the results of A/B tests run at scale in congested networks. To do so, I describe a production experiment with *bitrate capping*, a technique used in response to COVID-19 to lower bitrates offered and reduce overall internet load [72, 6]. Bitrate capping reduces congestion and improves performance in this experiment, but A/B tests give the opposite result—incorrectly suggesting that bitrate capping *increases* congestion!

The goal of this chapter is to make the networking community, both academic researchers and industry practitioners, aware of the bias of A/B tests and to propose techniques to help mitigate it. This research was conducted prior to the development of Sammy, hence the use of bitrate capping

in experimentation. I encourage future work evaluating Sammy at scale, or other algorithms that smooth video traffic. More generally, I encourage the community to apply these techniques broadly and evaluate networking algorithms with alternate experiments, and continued measurement and development of new techniques to mitigate bias.

1.3 Buffer sizing theory, and experiments for video traffic

In the last chapters of the thesis, I revisit a classic problem in networking: sizing router buffers. When packets arrive at internet routers, they are put into a buffer to be sent later. When packets arrive faster than the link capacity (the speed they are sent), the buffer grows. When they arrive slower, the buffer shrinks. The buffer allows the router to smooth out variability in arrival rates: when arrival rates are higher than capacity, the buffer allows the router to avoid dropping packets. When arrival rates are below capacity, the buffer allows the router to maintain link utilization.

There has been a long-standing question in networking of how big these buffers should be. Sizing the router buffer correctly is important, and the tradeoff is typically thought of as follows: if the router buffer is too small, congestion control algorithms can under-utilize the link. But if a buffer is too large, packets are delayed and the router is more complex and costly to manufacture.

There are two widely used rules of thumb for sizing router buffers to ensure 100% link utilization with TCP Reno: **Case 1: When a network carries a single TCP Reno flow**. Van Jacobson observed in 1990 [98] that a bottleneck link carrying a *single* TCP Reno flow requires a router buffer of size $B \geq \text{BDP}$, the bandwidth-delay product, in order to keep the link fully utilized.
Case 2: When a network carries multiple TCP Reno flows. Appenzeller, Keslassy, and McKeown argued in 2004 [8] that a bottleneck link carrying n long-lived TCP Reno flows requires a buffer of size $B \geq \text{BDP}/\sqrt{n}$ in order to keep the link highly utilized.

In Chapter 4, I revisit these rules of thumb and show similar rules that apply to modern congestion control algorithms, accounting for TCP changes like Rate-Halving [86, 160] and PRR [52], and modern congestion control algorithms such as Cubic [78], BBR [33] and BBRv2 [35]. Furthermore, we precisely characterize link utilization as a function of buffer size, and show how smaller buffers can still maintain high (though not 100%) link utilization.

One of the themes of this thesis is that video QoE does not translate directly between classic networking goals like packet loss or 100% link utilization, and so by focusing on QoE we can make progress on classic networking problems. To this end, I describe experiments with buffer sizing run at Netflix in Chapter 5, where I adjust the size of router buffers and measure the impact on video QoE. Properly sizing a buffer is both crucial for improving network and video quality of experience, and also quite difficult to do. We show experiments with both too-small and too-large buffers, both of which can negatively impact QoE.

The problem of buffer sizing is to make buffers as small as possible in order to ensure high

utilization for a congested link. But note that this is the opposite goal we have with Sammy, where we would like to ensure a link is *not congested* and utilized *as little as possible* while still ensuring high QoE. This difference fundamentally changes the buffer sizing problem for algorithms like Sammy that smooth video traffic. We go into more detail about this in Chapter 6, and discuss how sizing buffers for video traffic is still very open for future work.

1.4 Conclusions

Sammy significantly reduces congestion in experiments (reducing RTTs by 20% and retransmits by 50%), but is very different from typical approaches to congestion control. In fact, one could argue that Sammy is not a traditional congestion control algorithm at all. As we conclude in Chapter 6, this calls for a reevaluation of the way we approach congestion control in networking research.

Traditionally, congestion control research focuses on maximizing throughput. This leads to a zero-sum game where one algorithm getting more throughput causes its neighbors to get less, harming its neighbors. This thesis advocates for a different perspective on congestion control: prioritizing real-life internet usage (such as video streaming) while fostering a friendly environment for neighboring traffic. This fresh approach creates a number of opportunities for future research.

As discussed in Chapter 6, potential next steps include making video traffic even friendlier, reexamining congestion control and buffer sizing for video traffic, and implementing experiments with Sammy on a larger scale. I hope that Sammy is the beginning of new sort of research on internet congestion which uses application-level logic to create friendlier internet traffic, and that future work will help explore the potential of this approach. Together, we can create a better internet for everyone.

Chapter 2

Making video traffic a friendly internet neighbor

2.1 Introduction

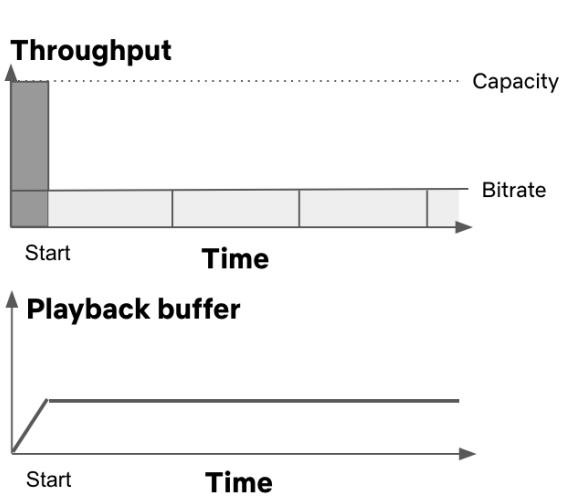
As discussed in Chapter 1, smoothing video traffic can reduce congestion and make it a friendlier internet neighbor. The major challenge is in doing so without making video traffic perform worse. There are two different aspects of this challenge.

First, there is a fundamental limit to how much traffic can be smoothed without impacting QoE. For example, take the smoothed video session shown in Figure 2.1a. We could smooth this even more by reducing the throughput before playback starts to match the rest of the session as in Figure 2.1b. But this would increase play delay. No buffer will be built up, so the playback will rebuffer if throughput varies.

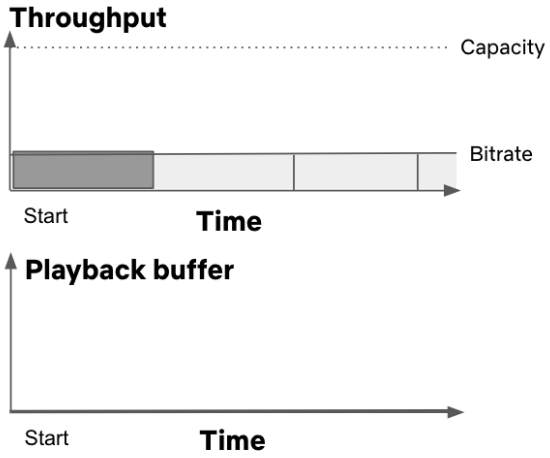
Second, ABR algorithms have historically had a core assumption that measurements of chunk throughput give them accurate estimates of the available bandwidth of the network [194, 193, 5, 101, 154, 168, 172, 91, 169]. Reducing chunk throughput breaks this assumption, and could cause an ABR algorithm to select lower qualities—artificially lowering QoE.

In this chapter, I present a novel solution to this challenge and make video traffic smoother. Remarkably, our approach is uniformly beneficial: substantially smoothing video traffic while slightly *improving* QoE relative to existing, finely-tuned production video streaming systems.

To smooth video traffic, I allow ABR algorithms to directly limit TCP's packet-by-packet sending rate using a new technique called *application-informed pacing*. An ABR algorithm might ask for a chunk of video to be delivered at no more than one packet per millisecond, and TCP uses TCP Pacing [86, 2, 191, 33, 132, 151, 74] to send packets no faster than once every millisecond. This allows the ABR algorithm to smooth out throughput across the full range of timescales: from the



(a) Smoother video traffic, same QoE



(b) Video traffic that has been smoothed to the point of reducing QoE (startplay delay is higher)

Figure 2.1: A few seconds of a typical streaming video session. Today, video has on periods (light grey) where it sends data as fast as the network supports (a). But as shown in (b), we can smooth throughput and reduce congestion without impacting video QoE.

level of a few packets, to an entire chunk, to an entire video session. Application-informed pacing is described in more detail in Section 2.3.2.

I next propose *Sammy*,[1] an algorithm that selects both bitrates and pacing rates to achieve high video QoE and improve smoothness. Sammy is described in Section 2.4. Our key insight, described in Section 2.3.1, is that while ABR algorithms have historically *used* measurements of available bandwidth to make their decisions, they do not *need* accurate estimates to achieve good QoE.

I implement Sammy at Netflix and evaluate it with large scale, production experiments in Section 2.5. Sammy substantially smooths video traffic: at the median, lowering chunk throughput by 51%. This improves congestion metrics: improving retransmissions by 52%, and RTTs by 22%. Sammy slightly improves video QoE relative to production values: improving initial video quality by 0.2% and overall quality by 0.03%, while maintaining rebuffers, and play delay.

I present illustrative lab experiments in Section 2.6 in which Sammy improves the performance of neighboring traffic: increasing its throughput and reducing queueing delay. Sammy improves delay for a neighboring UDP flow by 48%, improves throughput for a TCP flow by 25%, improves response times for HTTP traffic by 18%, and improves play delay for another video session by 5%.

Streaming video services are incentivized to deploy Sammy. First, neighboring traffic could easily be from the same video service. By improving performance for its neighbors, Sammy improves performance for the video service itself. Second, Sammy forces an ABR algorithm to be careful about its use of throughput estimates. This exercise gave us a slight QoE improvement, and could

[1] As of this writing, Sammy is the current reigning world's fastest snail [145].

yield larger improvements for other ABR algorithms.

This work is a first step towards smoothing video traffic. I conclude in Section 2.7 by highlighting that there is still more work to be done. As a community, we have an opportunity to further smooth traffic, video and beyond. After all, a smoother internet benefits everyone.

2.2 Background and Related Work

The burstiness of video traffic arises from the standard architecture of modern video streaming services. One of the core design principles of the internet is layering [41], separating applications from underlying transport protocols. The internet community has kept a minimalist inter-layer interface, giving applications little ability to express their service goals. As a result, transport protocols optimize for transport-level goals: maximizing throughput, avoiding congestion, and splitting throughput fairly. Adaptive bitrate (ABR) algorithms select bitrates to ensure that viewers get the best quality of experience (QoE) possible, given whatever throughput is picked by TCP.

In this section, we will give an overview of existing work and describe why this architecture makes it difficult to achieve our goal of smoothing video traffic while achieving high QoE. In this section we focus on the most relevant papers. For a more complete overview, there are a number of wider surveys [116, 153, 143].

2.2.1 ABR algorithms

When a network has limited bandwidth, there is a tradeoff between the three major video QoE metrics: quality, startplay delay, and rebuffers. A video playback can have both high quality and no rebuffers if it incurs a high play delay by downloading the entire video before it starts. It can start quickly in a slow network by either picking a low quality or rebuffering after playback starts. The role of an ABR algorithm is to manage this tradeoff between the different QoE metrics, and ensure that the user gets the best possible QoE.

Videos are split up into chunks of a few seconds each. Each chunk is encoded in a ladder of different bitrates: from a small, low-quality version to a larger, high-quality version. The ABR algorithm chooses a rung from this ladder for each chunk. The transport layer then splits that chunk into packets and sends each packet to the video client. When chunks are downloaded, they are added to a playback buffer in the video client. Even if the network is unavailable, the client can continue playing as long as there are chunks in the buffer.

When it takes too long to download a chunk, the buffer can shrink and it may be impossible to maintain high video quality without rebuffers. To deal with this, ABR algorithms can pick lower bitrates to grow the buffer and avoid rebuffers. There are two main types of ABR algorithms in use today:

Throughput-based: An ABR algorithm that takes explicit throughput measurements from the network, and uses them to select bitrates [194, 193, 5, 101, 154, 168, 172]. Typical algorithms produce some estimate based on chunk throughput, and then use it to optimize the various QoE metrics. ABR algorithms can also use throughput in other ways, for example, Oboe [5] switches between several parameter settings based on throughput measurements. It is generally understood [194, 193] that throughput-based algorithms will perform better the more accurately they are able to predict throughput of upcoming chunks.

Buffer-based: An ABR algorithm may select a bitrate based only on the buffer level [91, 169]. When the buffer is low, the algorithm will pick the lowest bitrate. When the buffer is high, it will pick the highest bitrate. Over time, these algorithms converge to an average bitrate close to the average chunk throughput. In effect, the buffer size encodes the past available bandwidth measurements from the network. In practice, buffer-based algorithms can also include a throughput-based component during startup [168].

Existing ABR algorithms rely on the available bandwidth measurements produced by TCP. By decreasing chunk throughput, we could make existing ABR algorithms perform worse. We discuss how we address this in Section 2.3.2.

Until now, ABR algorithms have focused on maximizing the quality of experience (QoE) for a video streaming client given whatever throughput is chosen by TCP. ABR algorithms do not make choices about throughput. In contrast, our goal is to design an algorithm that smooths video traffic and achieves high QoE while improving the internet for neighboring traffic.

2.2.2 TCP

Once an ABR algorithm has selected a chunk, it is the job of TCP congestion control algorithms to decide how fast to send the packets of that chunk into the network. Congestion control algorithms balance competing goals: achieving high throughput, avoiding congestion, and fairly splitting network resources among users [141, Sec. 3.2].

There is a long line of research on congestion control, and we refer the reader to existing surveys for details [143]. It is challenging (if not impossible [195, 200]) to simultaneously achieve all the goals of congestion control, and there are examples of congestion control algorithm struggling with packet loss and queueing delay, bufferbloat [69], and unfairness [17, 2, 191, 28, 118, 13, 158, 94, 51, 190, 30, 93, 105, 182, 183, 101]. By focusing on the needs of video QoE, Sammy gives up the goal of achieving high chunk throughput and so is able to improve congestion and leave more bandwidth available for other users of the network.

Our work uses the classic idea of TCP Pacing: a mechanism for adding delay between successive packet sends to reduce the size of bursts, and reduce packet drops and queueing delay [86, 2, 191, 33, 132, 151, 74]. Pacing gives packets a constant interarrival time, which theoretically minimizes queueing delay in general settings [79]. In practice, pacing reduces congestion [2, 191, 132]. Typically

Paper	Main Goal	Smoothing Mechanism	Eval.	Preserves QoE?	Smooths below TCP?
Our work	Smoothness	ABR-informed Pacing	Prod.	✓	✓
TCP Pacing [2]	Packet bursts	Pacing	Prod.	✓	✗
Trickle baseline [70]	Wasted buffer	Token Bucket	?	?	✓
Trickle [70]	Packet bursts	CWND limit	Prod.	?	✓
[154]	Wasted buffer	Delays TTFB	Lab	✗	✗
SABRE #1 [126]	Bufferbloat	RWND limit	Lab	✗	✓
SABRE #2 [126]	Bufferbloat	RWND limit	Lab	✗	✓
[4]	Video fairness	Token Bucket	Lab	✗	✓
[22]	Video fairness	Token Bucket	Lab	✗	✗

Table 2.1: Related work on smoothing video traffic either does not preserve QoE or does not reduce throughput below TCP-selected rates.

the time between successive packets is set to a value larger than the cwnd/RTT [2, 180], which reduces burstiness at the packet-level, but does not reduce chunk throughput. BBR [33] is a congestion control algorithm that directly adjusts the pace rate, but it aims to pace close to the bottleneck capacity while Sammy aims to pace significantly lower.

There has been prior work on improving fairness among competing video clients at the TCP layer. PCC Proteus [131] is a "scavenger" TCP protocol, i.e., it gives up throughput when competing with a non-scavenger congestion control algorithm. The authors describe a hybrid mode, in which video traffic switches from non-scavenger to scavenger mode when the throughput exceeds a threshold and show that this improves performance when multiple video clients compete. Minerva [134] improves fairness between competing video sessions by sharing proportionally based on a measure of perceptual quality. Sammy does not focus on improving fairness or competing in a certain way, but rather on making video traffic as smooth as possible for the benefit of its neighbors.

2.2.3 Reducing Burstiness for Video Traffic

There has been prior work on reducing the burstiness of video traffic. The novelty of our work is that by designing a joint ABR and rate limiting algorithm, we are able to reduce chunk throughput below available bandwidth *while achieving comparable QoE to today's top ABR algorithms.*

Related work on reducing video burstiness is summarized in Table 2.1. There are a number of differences to our work. The baseline algorithm described in Trickle [70] reduces chunk throughput based on the video encoding rate with the goal of reducing wasted buffers when videos end early. The impact of this algorithm on QoE, smoothness, or neighboring traffic is not evaluated in the paper but since the Trickle baseline reduces buffer sizes it likely has some impact on QoE. In contrast, our work explores how to design an ABR algorithm to improve smoothness while achieving high QoE. Work on pacing like TCP Pacing [2] and Trickle (relative to their baseline) [70] focuses on reducing per-packet burstiness while maintaining TCP-selected throughput. There is work [4, 126, 154] which reduces

throughput below TCP's selected rate using mechanisms other than pacing, and this work does not preserve video QoE (typically because it treats ABR algorithms as a black box). Finally, there is some related work [22, 122] which delays the time to first byte (TTFB) of the HTTP response. This reduces buffer sizes, but does not improve smoothness over TCP.

There has also been prior measurement work, observing that some video traffic reduces burstiness in practice using some of the mechanisms described above [147, 9].

The challenge of reducing burstiness with existing ABR algorithms

All prior work on reducing burstiness for video traffic falls short of achieving high video QoE because they treat ABR algorithms as a black box.

Today's ABR algorithms are designed to use either explicit measurements of available bandwidth (as in the case of throughput-based ABR algorithms) or implicit ones collected through the accumulation of a buffer (as with a buffer-based algorithm). If we reduce burstiness by reducing throughput, this changes available bandwidth and can easily cause ABR algorithms to select lower bitrates and reduce QoE.

As an example of how reducing chunk throughput can cause QoE issues, imagine a simple ABR algorithm which measures the minimum throughput x over the last few chunks and picks the highest video bitrate $< cx$ for some constant c.[2] Say we set $c = 0.5$, and picked a pacing rate of 1.5x the video bitrate. This would cause a downward spiral [90]: we would start with video bitrate B, pick a pacing rate of $1.5 \cdot B$, and measure a throughput of $x = 1.5 \cdot B$. We would then pick the highest video bitrate lower than $0.5 \cdot (1.5 \cdot B) = 0.75 \cdot B$, and would switch down. We would continue switching down until we reached the lowest bitrate.

As another example, imagine all chunks are one second long, and we pick a pace rate equal to the chunk bitrate. The chunk will download in exactly the same amount of time as it takes to play, one second. As a result, the playback buffer will not increase and remain at a near-zero level. If chunk throughput varied at all, there would be a rebuffer. For buffer-based algorithms, this also means that the buffer will not grow large enough to select a high bitrate chunk, reducing video quality.

2.3 Design discussion

In this section, we will describe the key components of Sammy: the key idea behind designing an ABR algorithm for pacing, and the application-informed pacing mechanism Sammy uses to limit TCP's throughput. In Section 2.4, we will use these components when introducing Sammy.

[2] This is the default dash.js algorithm when the buffer is low [168].

2.3.1 Designing ABR algorithms for pacing

Application-informed pacing reduces throughput below the available bandwidth of the network. With pacing, an ABR algorithm might *never* learn the available bandwidth of a network. As discussed in Section 2.2, existing ABR algorithms rely on measurements of available bandwidth. So how can an ABR algorithm optimize QoE with pacing?

The main idea behind our approach is that while ABR algorithms have historically *used* estimates of available bandwidth to make their decisions, they do not *need* accurate available bandwidth estimates to achieve good QoE. An ABR algorithm only needs to know whether the network can support the highest bitrate, or if it needs to reduce video quality to improve QoE.

For example, imagine streaming a video with a top bitrate of 10 Mbps. Once we have built up a small playback buffer, the ABR algorithm only needs to know whether the network throughput is high enough to sustain the top bitrate without rebuffering. If the available bandwidth is 100 Mbps or 1000 Mbps, a good algorithm would still pick the top bitrate.

Instead of relying on accurate estimates of available bandwidth, Sammy can use a pacing-informed ABR algorithm that instead solves a decision problem: *is the available bandwidth of the network enough to pick a bitrate, or not?* As we will discuss in Section 2.4.2, many commonly used ABR algorithms implicitly rely on such a decision problem and do not need accurate estimates of available bandwidth.

An alternate approach would be to estimate available bandwidth despite pacing, for instance using packet pair techniques [106, 111], or not pacing some portion of requests. We did not pursue this, in favor of an approach that avoids exact throughput estimation in the first place, as described above.

2.3.2 Limiting TCP throughput with application-informed pacing

Streaming video traffic is bursty because TCP sends as fast as the available network bandwidth, which is often higher than needed for good QoE. Our approach is to have the ABR algorithm reduce TCP's sending rate with *application-informed pacing*: a new technique that allows applications to set an upper limit on TCP's sending rate. By carefully limiting TCP's bursts, an ABR algorithm can reduce chunk throughput below the available network bandwidth while achieving high QoE.

In application-informed pacing, the ABR algorithm selects a pace rate and sends this rate to the server via an HTTP header. The server uses TCP Pacing as described in Section 2.2 and [86, 2, 191] to limit the sending rate at the server side. To achieve a desired rate of R packets per second, the server delays sending packets to ensure that there is a delay of at least $1/R$ seconds between the starts of successive packets.

Application-informed pacing is TCP-fair to existing internet traffic. TCP congestion control can still limit the sending rate by reducing the congestion window or pace rate, so the resulting throughput is *at most* the requested pace rate.

Deployability. Application-informed pacing is readily deployable. TCP Pacing is already part of the Linux kernel [54], is used in production at Google [33, 132, 151], and is available in certain NICs [151]. In Linux, an HTTP server can implement application-informed pacing by setting the `SO_MAX_PACING_RATE` socket option [56] to an application-provided value.

There is CDN support for application-informed pacing. Akamai supports CMCD, a video standard that allows clients to limit server-side throughput using the `rtp` parameter [12, 3]. Fastly allows setting TCP pace rates based on the value of an HTTP header [62].

Application-informed pacing is an example of cross-layer design, and there are lots of other ways to limit TCP's throughput. A system could use client receive windows to limit throughput [126], could limit a server's congestion window [70], or could use a server-side token bucket to reduce rates [4]. These techniques might be more bursty than application-informed pacing, but might be more easily deployable in certain settings.

2.4 Sammy

We will now describe Sammy, our joint ABR and pace rate selection algorithm. Sammy has separate algorithms for the initial phase (before playback starts), and the playing phase. This separation comes as a natural consequence of the different QoE goals of each phase. During the initial phase, one important goal is to start playback quickly. Once the playback starts, the QoE goal shifts towards avoiding rebuffers with high quality.

2.4.1 Algorithm for the Initial Phase

The initial phase of a video is the time period between when a user initiates playback and the playback actually starts. During this phase, ABR algorithms download chunks and build up a small buffer before beginning playback. Sammy has four competing QoE goals in this phase:

1. **High initial quality:** Sammy should pick high bitrates for the first few chunks of video playback.
2. **Few rebuffers:** Sammy should build up a sufficiently large buffer before playback starts to avoid rebuffers.
3. **Low play delay:** Sammy should begin playback as quickly as possible.
4. **Smoothness:** Sammy should smooth out traffic by picking low pace rates.

In the initial phase, we will not aim to improve smoothness. Play delay, the latency between a user clicks to play and the first frame is displayed, is sensitive to decreases in chunk throughput. Holding initial quality and starting buffer sizes the same and reducing throughput means we download the same number of bytes in a longer period of time and play delay will increase. Not pacing during the

initial phase has a minor impact on our overall goal because the initial phase is a small fraction of traffic: it is short relative to video duration, typically a few seconds over a session that is tens of minutes long. We will not pace and focus purely on the remaining three QoE goals.

That said, Sammy does need a pacing-aware algorithm for the initial phase. At the beginning of the initial phase, ABR algorithms do not yet have estimates of throughput from the current session but need to select a bitrate. To do so, prior work uses throughput measurements from the playing phase of previous sessions [100, 172]. With pacing, we significantly decrease throughput in the playing phase, making these inaccurate estimates of initial throughput. An alternate approach is to use information from the beginning of the TCP connection to estimate throughput [193], but our client implementations do not provide access to this information.

In Sammy, we record historical throughput measurements from *only* the initial phases of previous sessions on the same device and use them to select the initial bitrate. This approach has the advantage that we can decrease throughput as much as possible during the playing state without impacting throughput estimates in the initial state.

Having produced an initial throughput estimate which is unaffected by pacing, Sammy uses Netflix's existing bitrate selection algorithm for the initial phase.

2.4.2 Algorithm for the Playing phase

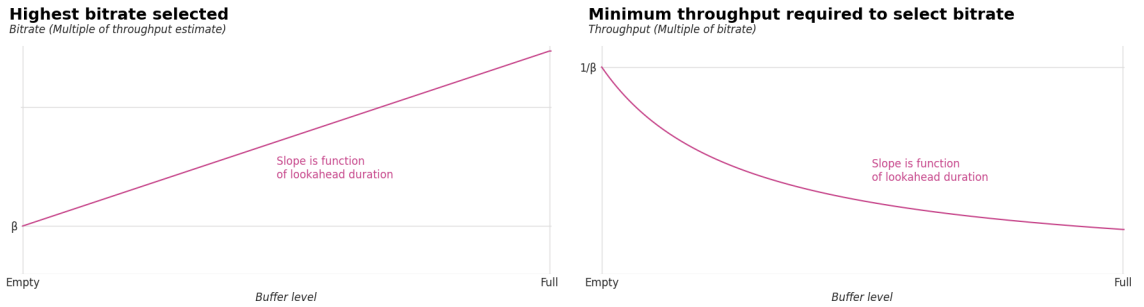
During the playing phase, Sammy selects both a bitrate and a pace rate to balance three competing QoE goals:

1. **High quality:** Sammy should pick high video bitrates.
2. **Few rebuffers:** Sammy should avoid playback interruptions by keeping the buffer above zero.
3. **Smoothness:** Sammy should smooth out traffic by picking low pace rates.

Picking a lower pace rate can affect all three goals: it improves smoothness, reduces throughput estimates (potentially impacting video quality), and causes buffers to grow more slowly (potentially impacting rebuffers).

Our overall strategy for the playing phase will be to pick an ABR algorithm that allows us to reduce throughput estimates without impacting bitrate selection. We analyze the algorithm to find the minimum required throughput to avoid impacting bitrate selection. We then use a buffer-based algorithm [91] to pick high pace rates when the buffer is low (growing the buffer more quickly) and lower pace rates when the buffer is high (growing the buffer more slowly), while ensuring that the pace rates stay above the minimum required throughput from our analysis.

Sammy's ABR algorithm: As discussed in Section 2.3.1, instead of relying on an ABR algorithm which *accurately estimates* available bandwidth, Sammy will use an ABR algorithm based on a decision problem: *is the available bandwidth high enough to pick a bitrate, or not?* It will then



(a) A typical throughput-based ABR algorithm picks higher bitrates as the buffer grows.

(b) A typical throughput-based ABR algorithm has a minimum throughput needed to select a chunk.

Figure 2.2: By analyzing how an ABR algorithm picks bitrates as a function of chunk throughput estimates (a), we can find a lower bound on pace rates to avoid impacting QoE (b).

pick a pace rate which ensures the ABR algorithm can correctly answer this question for the top bitrate.

Fortunately, many existing ABR algorithms implicitly solve such a decision problem. As an example, we will consider a typical throughput-based algorithm: the HYB algorithm [5], modified to use lookahead (i.e. take upcoming chunk sizes into consideration). This analysis also applies to MPC algorithms [194] with appropriately chosen utility functions. The HYB algorithm computes a throughput estimate from recent throughput measurements, and multiplies this estimate by a parameter $\beta \in [0, 1]$ to offset prediction errors. It then uses a standard buffer update equation [90] to predict how the buffer evolves over the lookahead duration. It picks the highest bitrate which keeps the buffer above zero.

To better understand the behavior of this algorithm, in Section 2.4.3 we analyze how the playback buffer evolves over time. Let D_T be the lookahead duration of the upcoming T chunks. We show that for a throughput x, bitrate r, and starting buffer size B_0, the buffer evolves according to

$$B_T = B_0 + D_T - D_T \frac{r}{\beta x}.$$

HYB picks the highest bitrate which keeps $B_T > 0$, which gives us the following constraint on the bitrate r:

$$r \leq \beta x \left(1 + \frac{B_0}{D_T}\right).$$

This function is shown in Figure 2.2a: as both buffers and throughput grows, HYB will pick higher bitrates. Implicitly, HYB uses a buffer-based approach to select bitrates [91].

As a corollary, this gives us a minimum throughput required to pick a bitrate r.

$$x \geq r\beta^{-1} \left(1 + \frac{B_0}{D_T}\right)^{-1}. \tag{2.1}$$

We graph this function in Figure 2.2b; when the buffer is empty, HYB needs an estimate of throughput equal to the bitrate divided by β. When the buffer is lower, HYB can select a bitrate with a lower throughput.

Equation 2.1 is the function HYB uses implicitly to decide whether or not throughput is high enough to select a bitrate. In order to avoid impacting bitrate selection, we must pick a pace rate higher than this value. When the buffer is empty, we must pick a pace rate of at least $1/\beta$. When the buffer is larger, we can pick a lower pace rate without impacting bitrate selection.

Sammy uses Netflix's production ABR algorithm, which is an MPC-style algorithm. We won't go into the details of this production algorithm, because it is proprietary and because our focus is to demonstrate how existing ABR algorithms can work with pacing.

Sammy's pace-rate selection. When the buffer is empty, Sammy paces at a multiple of the top bitrate. When the buffer is full, Sammy paces at a different, smaller multiple of the top bitrate. We set parameters so that the pace rate is above the minimum throughput required to pick the top bitrate given by Equation (2.1) and Figure 2.2. We can choose higher parameter values than this to tune the tradeoff between rebuffers and pace rates. Once Sammy selects a pace rate, it communicates this rate to the transport layer using application-informed pacing, as discussed in Section 2.3.2.

2.4.3 Relationship between chunk throughput, bitrates, and buffer sizes

In previous sections, we relied on a relationship between chunk throughput, bitrates, and buffer sizes. In this section, we will formalize this relationship.

There are a number of chunk selection opportunities, which occur at steps $t \in \{1, \ldots, T\}$. The chunk at time t has a duration d_t and size s_t which is selected by the ABR algorithm.

One step: Each time we select a chunk at time t, the buffer will evolve in some way. Let Δ_t be the time it takes to add chunk t to the buffer. For simplicity we will assume the buffer never becomes full and never becomes empty, but we could instead keep track of the amount of full and empty time after each chunk downloads.

The buffer evolution is given by the following standard equation [90]:

$$B_{t+1} = B_t + d_t - \Delta_t. \tag{2.2}$$

We will define the bitrate of chunk t as

$$r_t = \frac{s_t}{d_t}. \tag{2.3}$$

We will define x_t, the throughput of chunk t (e.g. in units of bits per second), as

$$x_t = \frac{s_t}{\Delta_t} = \frac{r_t d_t}{\Delta_t}. \tag{2.4}$$

Note that with these definitions,

$$B_{t+1} = B_t + d_t - d_t \frac{r_t}{x_t}. \tag{2.5}$$

Multiple steps: In addition to a single buffer step, we will also be interested in how the buffer evolves over T steps. Define the total duration D_T as

$$D_T = \sum_{t=1}^{T} d_t.$$

When playback starts, the buffer starts at some size B_0. If we expand (2.2), we have the following buffer size at time $T+1$. We could interpret this as being the buffer right after we finish downloading the last chunk.

$$B_{T+1} = B_0 + D_T - \sum_{t=1}^{T} \Delta_t. \tag{2.6}$$

We will define S_T to be the total size of chunks we download by time T

$$S_T = \sum_{t=1}^{T} s_t. \tag{2.7}$$

Define the time-average bitrate by

$$\bar{r} = \frac{\sum_{t=1}^{T} d_t r_t}{D_T} = \frac{S_T}{D_T}. \tag{2.8}$$

And finally we will define the time-average throughput as:

$$\bar{x} = \frac{\sum_{t=1}^{T} \Delta_t x_t}{\sum_{t=1}^{T} \Delta_t} = \frac{S_T}{\sum_{t=1}^{T} \Delta_t}. \tag{2.9}$$

With these definitions, the behavior of the buffer over T steps is the same as the behavior over one step, averaged. This is formalized by the following theorem, which should be compared to the single step update equation in (2.5).

Theorem 1. *In the above setting,*

$$B_{T+1} = B_0 + D_T - D_T \frac{\bar{r}}{\bar{x}}.$$

Proof. Given (2.6), all we need to show is that $D_T \frac{\bar{r}}{\bar{x}} = \sum_{t=1}^{T} \Delta_t$. Substituting the definition of $\bar{r}$, we have

$$D_T \frac{\bar{r}}{\bar{x}} = D_T \frac{S_T/D_T}{\bar{x}} = \frac{S_T}{\bar{x}}.$$

By the definition of $\bar{x}$, we have

$$D_T \frac{\bar{r}}{\bar{x}} = \frac{S_T}{S_T / \sum_{t=1}^{T} \Delta_t} = \sum_{t=1}^{T} \Delta_t.$$

□

Discussion

The main use of this theorem in our paper is to understand which bitrates our algorithm will pick, by understanding how a simulated buffer evolves as a function of bitrate and throughput. But this theorem is a much more general statement about how the playback buffer evolves. Note that the only critical assumption we have made is that (2.2) holds. Our definition of bitrates r_t and throughput x_t ensures that (2.4) follows from (2.2).

In this section, we will point out some of the consequences of the Theorem for ABR algorithms.

Cannot exceed average throughput without buffer help Intuition tells us average bitrate cannot exceed average throughput. Theorem 1 gives us a simple formalization.

Say that the buffer does not decrease, so $B_0 \leq B_{T+1}$. Then

$$1 - \frac{B_{T+1} - B_0}{D_T} \leq 1.$$

By Theorem 1, $\bar{r} \leq \bar{x}$. That is, the bitrate cannot exceed average throughput.

However if we reduce the size of the buffer, we can exceed average throughput. Suppose $B_0 \geq B_{T+1}$. In this case,

$$1 - \frac{B_{T+1} - B_0}{D_T} \geq 1.$$

By Theorem 1, $\bar{r} \geq \bar{x}$.

Building up a buffer comes at the expense of bitrate

Suppose we have built up a 5 minute buffer by the time we select the last chunk ($B_0 = 0$, $B_{T+1} = 300$), then rearranging Theorem 1 gives:

$$\bar{r} = \bar{x} \left(1 - \frac{5}{D_T}\right).$$

Over a twenty minute session, this says that $\bar{r} = 0.75x$. Restating, if an ABR algorithm builds up a 5 minute buffer over a 20 minute session then it will get a bitrate which is 75% of average throughput.

Sample paths do not effect bitrate

All the terms in Theorem 1 are averages, the difference between ending and starting buffer, and the duration. From the perspective of the average bitrate we can achieve, it doesn't matter if the throughput is stable or wildly variable. The path of the buffer is also not that important—we can

Type	Metric	% Chg.	95% CI
Congestion	Chunk Throughput	-50.5	[-51.2, -49.7]
	% Retransmits	-52.2	[-54.7, -49.9]
	RTT	-21.5	[-24.9, -21.5]
QoE	Initial VMAF	0.22	[0.2, 0.3]
	VMAF	0.03	[0.0, 0.1]
	Play Delay	–	[-1.4, 0.1]
	Rebuffers (/ hr)	–	[-10.2, 11.0]
	Rebuffers (% sess)	–	[-7.4, 3.8]

Table 2.2: A/B Test results for Sammy including the percentage change to control and confidence intervals. All statistically significant metric movements are improvements over Netflix's production algorithm.

build up a large intermediate buffer by picking a lower bitrate than throughput, and then spend it and regain bitrate.

As an example, suppose we start with no buffer and build up a thirty second buffer during the first sixty seconds of playback. By Theorem 1, over the first sixty seconds $\bar{r} = 0.5\bar{x}$. Suppose we make careful choices over the rest of the session, and keep the buffer at thirty seconds after twenty minutes of playback. By Theorem 1, $\bar{r} = 0.975\bar{x}$. By controlling the size of the buffer over the course of the session, we don't suffer for our early choice to build up a large buffer. This effect is what allows buffer-based algorithms to achieve high bitrates.

2.5 Production Evaluation

We implemented and ran our algorithm in production at Netflix. To evaluate its performance, we run a series of A/B tests [193, 164] to tune our algorithm and understand the tradeoffs between video QoE and congestion-related metrics. We compared Sammy to Netflix's existing extensively tested and finely-tuned production algorithm, to emphasize how Sammy can improve smoothness while maintaining or improving QoE. Each A/B test consisted of a control group running Netflix's production algorithm, and many treatment groups. Each treatment group had different settings of Sammy's parameters. We allocated a small fraction of Netflix's traffic to each test (¡ 0.5%), and measured the values of video- and transport-level metrics.

Our results show that Sammy significantly improves smoothness and reduces congestion-related metrics while *slightly improving* video QoE.

Parameter values: We will present results for a single set of parameters for Sammy throughout the rest of the paper, and we briefly discuss other parameter values in Section 2.5.3. Specifically, Sammy paces at 11.0x the maximum bitrate when the buffer is empty, and 2.9x the maximum bitrate when the buffer is full using the algorithm described in Section 2.4.2. Sammy also tunes certain parameters of Netflix's existing ABR algorithm.

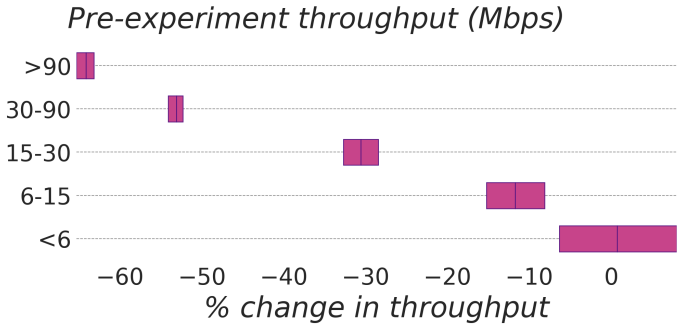


Figure 2.3: Reduction in chunk throughput (95% CI) for accounts with different pre-experiment chunk throughputs. Sammy reduces burstiness for accounts with pre-experiment throughput $>$ 6 Mbps.

2.5.1 Sammy reduces congestion

We first show that Sammy significantly improves smoothness, retransmissions, and round-trip times of Netflix traffic compared to the existing production algorithm. Table 2.2 presents the percent change between Sammy and Netflix's production algorithm, together with the 95% confidence interval.

Improving smoothness: To measure how much Sammy smooths video traffic, we focus on the average *chunk throughput* (the throughput during "on" periods). Sammy does not reduce quality, so reducing chunk throughput causes on periods to become longer, and increases the available bandwidth for neighboring traffic during on periods. We calculate the median of per-session average chunk throughput for both Netflix's production algorithm and Sammy. Sammy reduces chunk throughput by 50.5%.

Sammy's ability to reduce throughput depends on how much higher network bandwidth is relative to maximum bitrates. This raises the question about how Sammy performs in slower networks. For all accounts in the A/B test, we computed their pre-experiment throughput by looking at the 95th percentile of their chunk throughput for the week before the test began. We grouped accounts by the range of pre-experiment throughput: ¡6 Mbps, 6-15 Mbps, 15-30 Mbps, 30-90 Mbps, and ¿ 90 Mbps. We calculated average chunk throughput within each group of accounts, and compared Sammy's throughput of each group to that of the production algorithm. Figure 2.3 shows the percent change in throughput as a function of pre-experiment throughput. For accounts with pre-experiment throughput of more than 90 Mbps, Sammy reduces chunk throughput by 64%. As pre-experiment throughput decreases, Sammy reduces chunk throughput less. But Sammy does significantly reduce throughput (improving smoothness) for all pre-experiment throughputs more than 6 Mbps.

Reducing network congestion: Intuitively, reducing chunk throughput and improving smoothness should translate into improvements in congestion-related metrics, specifically lower packet retransmission rates and round-trip times (RTTs). This is supported by our A/B test results.

We calculate the fraction of retransmitted bytes over all bytes sent by TCP for each session. Sammy improves the median fraction of retransmitted bytes over all sessions by 52.2%. We measure RTTs for each packet sent by TCP and store them for each TCP connection in a t-digest [57]. We merge the t-digests for all TCP connections in a session, and estimate the median RTT for the session. We measure the median of median RTTs over all sessions. Sammy improves RTTs by 21.5%.

Given Sammy reduces chunk throughput, improves smoothness, and reduces congestion for Netflix traffic, it is plausible that neighboring traffic sharing a bottleneck link with Netflix's traffic should see improvements as well. Section 2.6 shows in a lab setting that Sammy's improvements in congestion-related metrics can translate to QoE improvements for neighboring traffic.

2.5.2 Sammy improves QoE

It is not surprising that picking lower pace rates would improve smoothness and network congestion, but more surprisingly, we show this can be done at no cost to the video user experience. In our experiments, Sammy *slightly improves* QoE. These results are summarized in Table 2.2.

Improving quality: We measure video quality by Video Multi-method Assessment Fusion (VMAF) [25], a method for estimating a viewer's perception of a video's visual quality. We calculate a time-weighted average of VMAF to get a score for each session, and measure the median score over all sessions. Sammy slightly increases overall VMAF, which is driven primarily by an increase in initial VMAF (the VMAF during the first twenty seconds of video playback). In other words, Sammy's video quality is slightly higher than with Netflix's production algorithm.

The improvements to initial VMAF (and overall VMAF) in our experiments come primarily from using *only* estimates of initial throughput during the initial phase (instead of chunk throughput from the playing phase of previous sessions), and from retuning Netflix's initial bitrate selection algorithm around these new throughput estimates.

Maintaining other metrics: Sammy has no statistically significant impact on any other aspect of QoE. There is no significant change in rebuffers: both the fraction of sessions that have at least one rebuffer, and the number of rebuffers per hour streamed. There is also no significant difference in play delay.

2.5.3 Tradeoffs and parameter settings

Sammy has a number of parameters that can be tuned, including parameters for pace rate selection and for the chosen ABR algorithm. We used Ax [142] to search the parameter space and find a Pareto improvement to all metrics of interest across multiple rounds of A/B testing.

Different parameter settings allow us to trade off between the different metrics of interest. Figure 2.5 shows the tradeoff between chunk throughput and video quality. Each point represents one treatment group in an A/B test, each with a different value of parameter settings. The x-axis

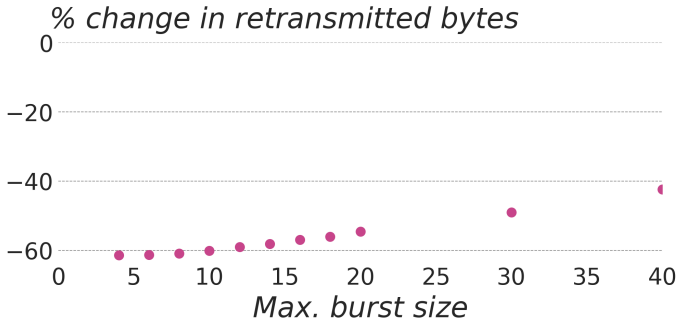


Figure 2.4: Change in retransmissions as a function of the pacing burst size in a production A/B test. Lower burst sizes improve retransmissions.

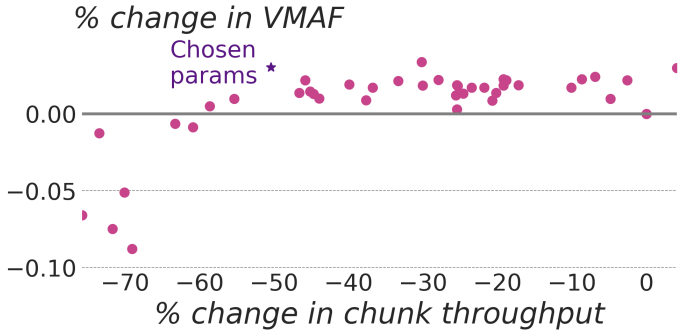


Figure 2.5: Tradeoff between video quality (VMAF) and chunk throughput for different choices of parameters.

is the % change in chunk throughput for that group, and the y-axis is the % change in VMAF. The parameters we selected reduced chunk throughput by 51% relative to control, while increasing VMAF by 0.03%. Other parameter settings give other points on this tradeoff. Eventually, decreasing throughput results in a decrease in VMAF.

2.5.4 A baseline approach reduces QoE

Sammy works hard to avoid reducing QoE with pacing, and a natural question is whether this work is necessary. Why not just pick a pace rate which is much higher than the maximum bitrate and call it a day? We ran an experiment which shows that this approach underperforms Sammy in all of our goals.

We ran an experiment with the production Netflix ABR algorithm in which we limited the pacing rate for each chunk to 4x the maximum bitrate. We made no other changes. Pacing in this way reduced chunk throughput by 53% but we observed a degradation in most of the major components of video QoE: play delay increased by 6%, and VMAF decreased by 0.2%. The play delay increase was enough to reduce the overall level of streaming, causing the experiment to be automatically stopped by safety systems.

Sammy outperforms this approach in both congestion and QoE-related metrics. Sammy achieves a comparable reduction in chunk throughput of 51% while *improving* QoE. If we were to choose parameters from Figure 2.5 which reduced QoE, Sammy would achieve a higher throughput reduction of 75% for a much lower VMAF reduction of 0.07%.

2.5.5 Effect of burst size

With pacing, there is an option of how large a burst to send at a time. To pace at 12 Mbps with 1500 byte MTU, we could send one packet every 1 ms, two packets every 2 ms, or 10 packets every 10ms. Intuitively, there is a tradeoff in picking the burst size: smaller bursts should improve congestion-related metrics, but also reduce opportunities for segmentation offload which can increase CPU usage.

Netflix's TCP implementation's default behaviour is to limit line-rate bursts to no more than 40 packets at a time. We ran an experiment where we paced at a constant 2x the maximum bitrate, and adjusted the per-packet bursts from 4 packets up to 40 packets. Figure 2.4 shows the results of this experiment.

Pacing with a burst size of 40 packets corresponds to only reducing chunk throughput, and not reducing the size of per-packet bursts. This reduces retransmissions by 40% relative to not pacing. As the burst size decreases, retransmits reduce by up to 60% relative to not pacing. But as the burst size decreases, there is no statistically difference in either chunk throughput or video QoE metrics.

This result shows why it is beneficial to use TCP Pacing instead of capping the congestion window as in prior work [70]. In our experiments, we use a burst size of 4 packets. By reducing

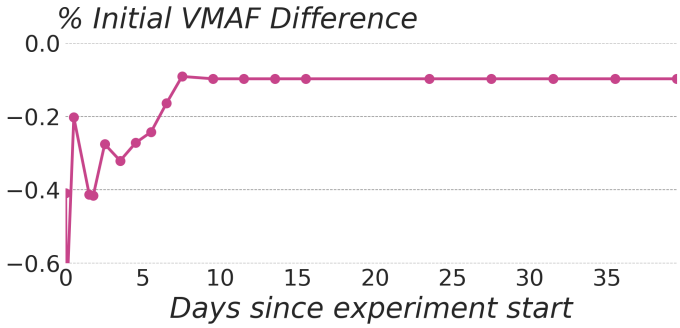


Figure 2.6: Initial quality difference over time during an A/B test. The treatment algorithm is missing historical data at the beginning of the experiment, and so performs worse over the entire experiment.

the burst size from 40 (as it would be if we capped the congestion window) to a burst size of 4, we improve retransmissions by an additional 20%.

2.5.6 Effect of historical data

As described in Section 2.4.1, Sammy and prior work use historical throughput measurements for initial bitrate selection. Doing so creates a dependency between successive sessions: the throughput at the beginning of one session impacts the bitrate selection decisions at the beginning of the next. Using historical data improves performance, but the dependency creates challenges for evaluation.

As an example, we ran an experiment simulating introducing a new historical estimate. The treatment group started with no historical measurements, while the control group had historical measurements. Both groups updated historical throughput with the same estimates, and there were no other differences between the groups. Figure 2.6 shows the percent difference in initial quality over the course of the experiment. The treatment group started with much lower initial quality and it stayed lower over the course of the experiment. It took a week for the initial quality of the treatment group to reach its closest point to the control group.

To deal with this challenge, we reset historical throughput information in both treatment and control groups in all experiments to enable an "apples-to-apples" comparison between the two.

2.6 Improving QoE of neighbors

In this section, we will present lab experiments where we directly measure how Sammy improves the QoE of neighboring traffic. The previous section shows how Sammy reduces chunk throughput and congestion-related metrics at scale. Lab experiments with a single setting are clearly not representative of most traffic on the internet, so the goal of these experiments is to illustrate how Sammy can improve the QoE of a few applications that might share its bottleneck.

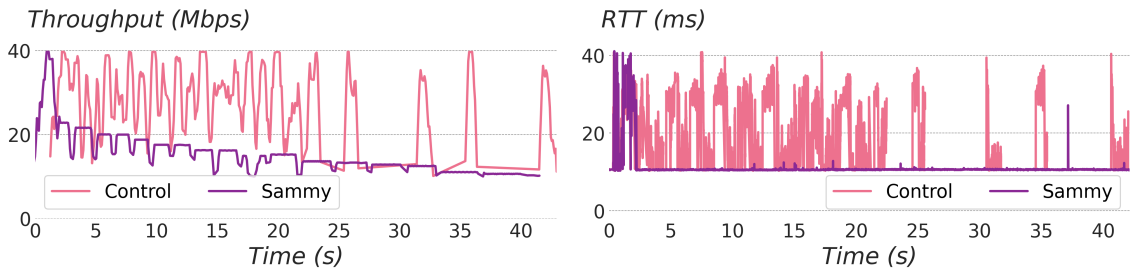
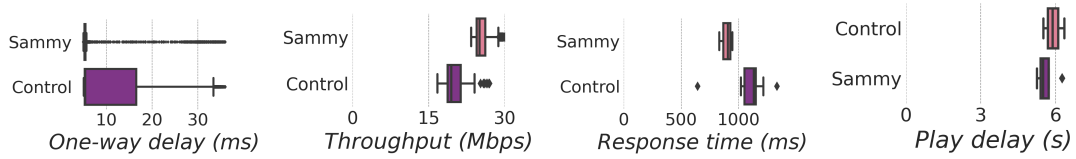


Figure 2.7: Throughput and RTT for a single Sammy flow running in a lab environment compared to Netflix's production algorithm. After 3 seconds, Sammy reduces chunk throughput enough to avoid congesting the link. Reducing chunk throughput helps it avoid on-off periods beginning for control traffic around 25 seconds.



(a) Neighboring UDP traffic **(b)** Neighboring TCP traffic **(c)** Neighboring HTTP traffic **(d)** Neighboring video traffic

Figure 2.8: In the lab, Sammy improves QoE for neighboring traffic relative to control for (a) UDP one-way delay, (b) TCP throughput, (c) HTTP response time, and (d) video play delay.

Without Sammy, the video traffic fully utilizes the link and fills up the queue, in turn impacting neighboring traffic. Sammy smooths out traffic, and chunk throughput drops to well below network capacity. This behavior avoids congesting the link, and gives neighboring traffic more bandwidth to use for itself.

Experiment setup. In all experiments, we use a 40 Mbps link with a 5ms RTT, and a queue size of 4 times the bandwidth-delay product. Sammy plays a video with a maximum bitrate of 1.05 Mbps. We run a video session using Netflix's production algorithm and a neighboring application at the same time, and then repeat the same experiment using Sammy and observe how the neighbor's QoE changes.

Sammy on its own. To understand how Sammy improves performance for neighboring traffic, we will first look at how it performs on its own. Figure 2.7 shows the throughput and RTT for a single Sammy flow, compared to a single control flow running Netflix's production ABR algorithm.

At the beginning of the session, both Sammy and control send as fast as possible during the initial phase: fully utilizing the network and filling up the queue. Playback starts after about three seconds, at which point Sammy begins pacing. The pace rate it picks is about 20 Mbps—low enough to avoid congesting the link, so RTT goes to zero. Over the rest of the session, Sammy decreases the throughput to about 15 Mbps. This rate is below its TCP-fair share rate of 20 Mbps when it

shares a link with neighboring traffic.

The change in metrics for this session is comparable to the overall change in metrics for the A/B test in Section 2.5. For this session Sammy reduces throughput by 46% (slightly less than in the A/B test) and RTTs by 42% (about twice as much as in the A/B test results).

When neighboring traffic shares this particular network with Sammy, it will experience an extra 5 Mbps of available bandwidth and no additional queueing delay. This leads to the following benefits (shown in Figure 2.8):

UDP: We first run an experiment where the neighboring traffic is a 5 Mbps paced UDP flow. The one-way delay measured for UDP packets is shown in Figure 2.8a. Sammy eliminates queueing delay for the UDP traffic, reducing the one-way delay by 48%. Without Sammy, video traffic keeps the queue full (see Figure 2.7) and the UDP traffic experiences queueing delay. Sammy sends no faster than 20 Mbps during playback, so the queue stays empty even with 5 Mbps of UDP traffic.

TCP: We next run an experiment where the neighboring traffic is a standard, congestion window limited TCP Reno connection. The TCP connection begins 10 seconds after playback starts. The TCP throughput is shown in Figure 2.8b. Without Sammy, the TCP flow gets an average of 20 Mbps (slightly higher than its TCP-fair share of throughput). Sammy increases throughput for the TCP flow by 25% which gets an average of 25 Mbps.

HTTP: The next experiment demonstrates the benefits of Sammy to neighboring HTTP requests. We repeatedly issue 3MB HTTP requests during video playback. We measure the HTTP response time shown in Figure 2.8c: the time between when the first byte of the request was issued and the last byte of the response was received. Sammy improves average response times by 18%, reducing them from 1095ms to 898ms.

Streaming video: We run another experiment to measure the impact of Sammy on a neighboring video session. We start one Sammy video session, and after a few seconds we start a video session using Netflix's production algorithm. We show the play delay for the neighboring session in Figure 2.8d. Over four trials, Sammy consistently improves the play delay of its neighbor by 5%—an average of 282ms. When a streaming service shares a bottleneck with itself, Sammy can improve a streaming service's own play delay in this way. This result gives stremaing services an incentive to deploy Sammy.

2.7 Conclusion

Our approach shows that ABR algorithms can dramatically reduce the burstiness of video traffic without reducing QoE. In our experiments run at scale at Netflix, Sammy is able to reduce the median chunk throughput by 51%, reducing retransmissions by 52% and RTTs by 22%. These improvements to network congestion came with no harm to video QoE. In fact, we observed a small improvement in video quality (both initially and overall) and no statistically significant changes in other video QoE

metrics. Because Sammy does not aim to fully utilize the link, there is more bandwidth available for neighboring traffic during Sammy's on periods. Our lab experiments illustrate how this can improve performance for neighboring traffic: Sammy reduces delay for a neighboring UDP flow by 48%, increases throughput for a neighboring TCP flow by 25%, reduces response times for neighboring HTTP traffic by 18%, and even reduces play delay for neighboring video traffic by 5%. We leave deeper investigations of the impact on neighboring traffic to future work, and would be especially interested in experiments to measure the impact at scale.

In many ways, today's video streaming architecture is a *response to* the two control loops managed by ABR and TCP. TCP learns and acquires its fair share of bandwidth on a packet-by-packet timescale; and in turn ABR algorithms adapt bitrates at chunk-by-chunk timescale. Given the steps taken in this paper, a compelling future path forward is to consider a *single control loop* to both determine the video bitrate and when to transmit each bit of the stream over the network. This algorithm could jointly optimize video QoE and transport-layer goals like congestion and fairness, and could avoid the pitfalls associated with two interacting control loops [90]. We leave that work for others. Here, we instead have the ABR algorithm limit TCP's sending rate, so as to allow more rapid deployment with current video streaming services, and keeping with standard practice of sharing the internet using TCP. One could also imagine a range of options between the two, where ABR algorithms share more and more information with the underlying transport layer. Broadly, the significant empirical results found in this paper suggest that such innovations have the potential for significant impact not only on video streaming services, but the internet at large.

We view our work as a starting point for using application-level logic to smooth out internet traffic. We have shown that video streaming does not always need the maximum throughput TCP can achieve. The layering architecture of the internet encourages other applications to use a similar strategy of allowing TCP to select the maximum throughput without application input. By using details about the behavior of other applications, we may be able to make other types of internet traffic into friendlier neighbors as well.

Chapter 3

Unbiased Experiments in Congested Networks

3.1 Introduction

Engineers routinely run A/B tests when testing new network algorithms. In an A/B test, the experimenter randomly allocates a small fraction of traffic (say 1% or 5%) to a new algorithm, called the treatment group, and compares its performance against the control group running the old algorithm. A/B tests are widely used as the gold standard for understanding how a new algorithm will behave at scale. Almost all large tech companies routinely use A/B tests to evaluate changes before deploying them [113, 176, 120, 34, 73, 48, 104, 157, 138]. Networking research often includes the results of A/B tests, and uses them to justify new algorithms [96, 36, 161, 34, 33, 138, 117, 53, 63, 52, 120, 104, 92, 127, 193, 119].

So when we recently ran experiments to test whether *bitrate capping* reduces network congestion for Netflix, we ran A/B tests. Bitrate capping was introduced in response to COVID-19; major streaming services cooperated with governments to lower bitrates offered and reduce overall internet load [72, 6]. This caused a reduction in congestion in certain networks around the globe.

We decided to dig deeper, to understand exactly how bitrate capping reduces congestion, and how doing so impacts video quality metrics. While we had data from just before and after bitrate capping was deployed (and later when it was removed), these were during periods of lockdown and stay-at-home orders when the internet was changing rapidly. We wanted to conduct a more systematic study of its effects. Naturally, we ran an A/B test where we capped a fraction of traffic to a very congested network.

In this A/B test, capping didn't appear to reduce congestion at all! In fact, it appeared to make things worse: capped traffic experienced 5% lower throughput and 5% higher delay. The A/B

test results were so marginal that if we had not had evidence showing that bitrate capping reduced congestion when widely deployed, we might have dismissed it and not explored further. How could a treatment that we *knew* reduced congestion at scale not also reduce congestion in an A/B test?

Stepping back, we realized the confusion could be caused by *interference*. Interference is when units in the treatment group interact with units in the control group. It is well known in causal inference that interference can bias experiment results [95]. In social networks, changing something for a user in the treatment group can impact the behavior of their friends in the control group and bias the results of an experiment [58]. In online marketplaces, increasing the price of items in a treatment group can increase the demand for the relatively cheaper items in the control group and bias results [88]. There are many examples of interference bias from markets, education, disease, and more [109, 44, 80, 89].

Both treatment and control groups in our test used the same network, and their packets traversed the same links and same queues. There is a long line of networking research showing that algorithms compete with each other when sharing a congested network [158, 94, 189, 190, 30, 93, 182, 183, 17, 2, 191, 28, 118, 13, 51, 105]. If capping bitrates freed up bandwidth, the uncapped control traffic could take up that bandwidth and get better performance. This could make bitrate capping look worse than it would if the uncapped traffic were not present, even if it was improving congestion. This gave us reason to believe that interference may exist, which would explain our unexpected A/B test results.

In this work we show that interference exists in experiments run in congested networks, and biases the results of A/B tests at scale. We show that bitrate capping does reduce congestion, and that the misleading A/B test result was due to interference. In order to do this, we propose and test new experiment designs which more accurately evaluate new algorithms. Our results suggest that usual A/B testing practice paints an incomplete picture of the performance of new algorithms in congested networks, and should be complemented by additional experiments.

Without interference, A/B tests give us a way to safely and accurately evaluate performance using a very small fraction of traffic. But because of interference, A/B tests on small fractions of traffic do not accurately predict performance at scale. Interference therefore creates a tradeoff between safety and accuracy: the only way to accurately measure performance is to run an algorithm on 100% of traffic, but nobody would do this with an untested algorithm! Our goal in this paper is to make the networking community, both academic researchers and industry practitioners, aware of this tradeoff and to propose techniques to help mitigate it. We encourage the community to apply these techniques broadly and evaluate networking algorithms with alternate experiments. We encourage continued measurement and the development of new techniques to mitigate bias.

We begin with an overview of experiment design in Section 3.2. We describe how A/B tests are run, and which quantities they estimate. Using a framework from the field of causal inference, we define the relevant quantities of interest for new networking algorithms.

We then run small lab experiments in Section 3.3 to give examples of how networking A/B tests can be biased. We show that experiments using multiple parallel connections, packet pacing, and different congestion control schemes all exhibit bias. If we were to evaluate these algorithms using naïve A/B tests, we would make incorrect conclusions. We might prematurely abandon a good algorithm, or deploy an algorithm that behaves worse when widely deployed than in the experiment.

Returning to our bitrate capping experiments, in Section 3.4 we describe our joint experiments with Netflix. We study the performance of bitrate capping and report on the bias we found in our initial A/B tests. While measurements show that bitrate capping significantly reduces congestion, naïve A/B tests do not reflect this behavior. Naïve A/B tests miss changes in some metrics, overestimate or underestimate the changes in others, and even get the *direction* of improvement wrong for a few. We were able to carry out this analysis due to a unique network architecture at Netflix. Using a pair of reliably congested links with well-balanced traffic, we ran different experiments on each link and compared the results.

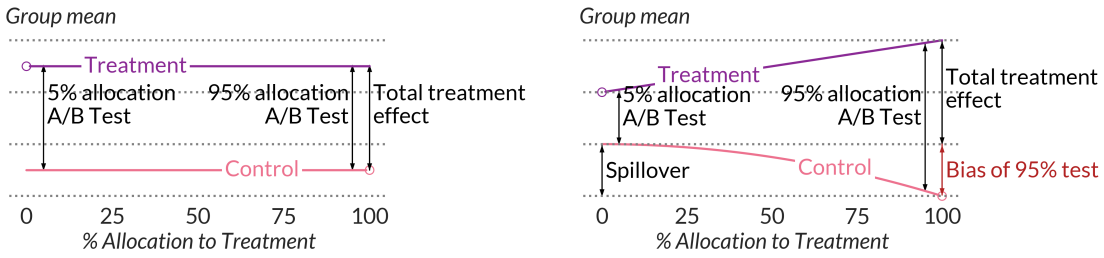
Based on our experience, in Section 3.5 we investigate possible ways experimenters can accurately evaluate new algorithms at scale. We discuss two possible paths to managing the tradeoff between safety and bias. The first is to adapt the common process of gradual deployments to measure interference. The second involves the use of small-scale, targeted *switchback* experiments to more accurately measure the effects of a new algorithm while managing safety concerns. We use the results of our paired link experiment to *simulate* what the experimenter might have obtained in these alternate approaches, and show that both substantially reduce bias.

We believe this paper is just the beginning of work on unbiased network experimentation. There is much to explore in designing more effective experiments, improving the analysis of experiments we run, and understanding the way interference behaves in networks. We wonder how many effective algorithms have been abandoned because of the way we run experiments, and what ineffective algorithms have been deployed because we were misled by A/B tests? Accordingly, we situate our work within the broader context of related research in Section 3.6 and conclude in Section 3.7.

3.2 What we want to measure

Before discussing experiments in more detail, it will be useful to give some background on how they are run, and what they can measure. In this section we provide a formal statistical foundation for A/B testing. The presentation is borrowed from causal inference [95]. The description is simplified, but gives enough conceptual scaffolding for the remainder of our work.

Treatment assignment. When we evaluate a new algorithm there are some *units* which run the algorithm. Units may be users, sessions, flows, connections, servers, etc... We let U be the set of all units. Each unit $i \in U$ is allocated to either *treatment* where it runs the new algorithm or *control* where it does not. Let A be the vector of treatment assignments to all units. We denote treatment



(a) A/B tests without congestion interference

(b) A/B tests with congestion interference

Figure 3.1: A/B tests are used to estimate the total treatment effect: how much better a treatment is than control if both were deployed globally. A/B Tests give accurate estimates of Total Treatment Effect (TTE) when there is no interference between sessions as in (a), but may be misleading when there is as in (b).

as $A_i = 1$, and the set of treated units as T. We denote control as $A_i = 0$ and the set of control units as C.

Potential outcomes. When evaluating a new algorithm, we are interested in how it improves various metrics. In the language of causal inference, these metrics are called outcomes. Let $Y_i(A)$ be the outcome of interest on unit i given the vector of treatment assignments A. $Y_i(A)$ might be the average throughput of unit i, the minimum latency, or the 99th percentile packet loss. $Y_i(A)$ can be a random variable, since we expect some variability due to randomness in algorithms and randomness in arrivals. [1]

Randomized unit assignment. In an A/B test, we randomly assign units to treatment independently with probability p or control with probability $1 - p$. In other words, each A_i is an independent Bernoulli(p) random variable. We refer to the probability p as the *treatment allocation*.

To make this point more explicit, we introduce some additional notation. Define $\mu_T(p)$ (resp., $\mu_C(p)$) to be the average outcome value over the randomness in the assignment of treatment (resp. control), when the treatment allocation is p:

$$\mu_T(p) = \mathop{\mathbb{E}}_{T \subset U} \left[\frac{\sum_{i \in T} Y_i(A)}{|T|} \right].$$

Depending on the setting and the treatment, $\mu_T(p)$ may or may not depend on the treatment allocation p. This is visually depicted in Figure 3.1. $\mu_T(p)$ is the purple treatment line, and $\mu_C(p)$ is the pink control line.

Average treatment effect. An A/B test evaluates the average treatment effect. This is how much better the treatment group performs than the control group, when a p fraction of the traffic

[1] This approach to causal inference via potential outcomes was pioneered by Neyman [170] (a 1990 translation of the original 1923 publication) and Rubin [149]; see [95] for details.

is allocated to treatment and $1-p$ to control. It is defined as:

$$\tau(p) = \mu_T(p) - \mu_C(p), \tag{3.1}$$

This is visually depicted in Figure 3.1. The treatment effect at any point on the graph is the difference between the treatment and control lines.

Total Treatment Effect. When evaluating a new algorithm, we are often interested in what *would* happen if we were to deploy it widely. This is the *Total Treatment Effect*, or TTE: the difference between the average outcome when *all* flows are in treatment and when all flows are in control. In terms of our notation above:

$$\mathsf{TTE} = \mu_T(1) - \mu_C(0).$$

This is depicted in Figure 3.1: it is the difference between the right-hand side of the treatment line (when all traffic is treated), and the left-hand side of the control line (when all traffic is allocated to control). Depending on the setting, it may or may not equal the average treatment effect.

Note that this definition of TTE is from the perspective of the experimenter, and not the internet. The experimenter may only control a small fraction of all traffic on the internet, and in this case TTE measures what happens if they switched all traffic under their control to a new algorithm. The TTE is also sometimes called the "global average treatment effect" in causal inference work (e.g., [107]), but we have avoided this name to avoid confusion around this point.

It is also reasonable to talk about TTE in specific groups of traffic. For instance, we may be interested in the TTE if we were to move all traffic globally to a new algorithm, but we may be also interested in the TTE for a single network or a group of networks. This can be incorporated into the definition by changing the set of treatment and control flows.

Spillover. In addition to how well a new algorithm performs on its own, we are often also interested in how a new algorithm impacts existing algorithms. Recently, [189] defined the notion of the "harm" of a new algorithm, which is the negative effect caused by a new algorithm competing with an existing algorithm. This networking concept is similar to the concept of *spillovers* in the causal inference literature (*e.g.* [77, 44]). Formally, we define the spillover of treatment on control as the effect of increasing the treatment fraction to p on *control* units, relative to when the treatment units were not present. In terms of our notation:

$$s(p) = \mu_C(p) - \mu_C(0).$$

Spillover is non-zero when deploying a treatment algorithm has some impact on the control algorithm. This is shown in Figure 3.1b. Note that spillover is only defined for $p < 1$. If $p = 1$, there is no control traffic and no spillover can occur.

Spillovers may or may not be undesirable. It is possible that deploying a new algorithm can improve existing traffic, and we will see examples of this later.

Estimating from A/B tests All the quantities above are expectations over the distribution of all possible treatment assignments. Any experiment has only one set of treatment assignments and can only observe one set of potential outcomes—all other potential outcomes are missing. The fundamental problem in causal inference is to reason about these missing outcomes given what we observe.

In causal inference, we use the observed outcomes to estimate the quantities above. An estimator is called *unbiased* for some quantity if its expectation is equal to that quantity.

In an A/B test we randomly allocate units to treatment or control, and measure

$$\widehat{\mu_T}(p) = \frac{\sum_{i \in T} Y_i(A)}{|T|}.$$

This process gives an unbiased estimator of $\mu_T(p)$, since $\mathbb{E}\,\widehat{\mu_T}(p) = \mu_T(p)$, and similarly for $\mu_C(p)$. By linearity of expectation,

$$\widehat{\tau}(p) = \widehat{\mu_T}(p) - \widehat{\mu_C}(p)$$

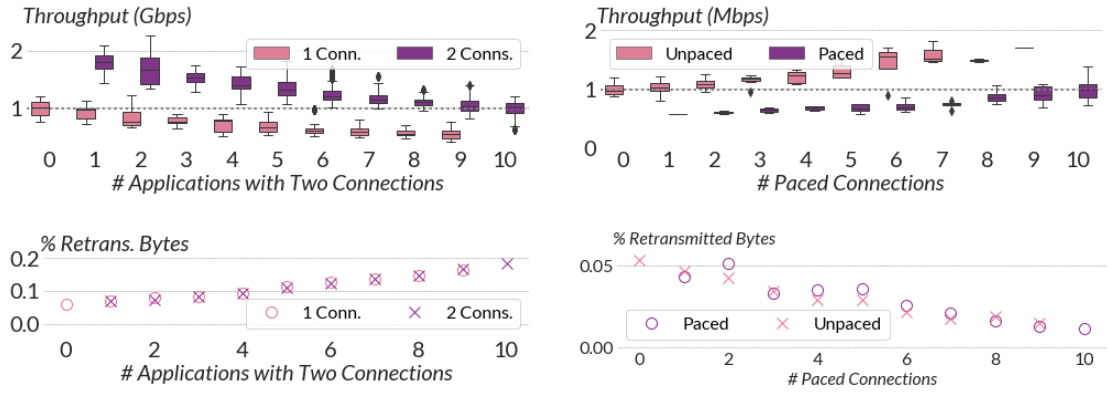
is an unbiased estimator for $\tau(p)$, and we can define similar estimators $\widehat{\mathsf{TTE}}$, and $\widehat{s}(p)$.

Congestion Interference In virtually all real-world experiments in networking today, experimenters run an A/B test. They infer that an improvement in the A/B test implies an improvement if the treatment were to be deployed. In our notation, this means that they use $\widehat{\mu_T}(p)$ and $\widehat{\mu_C}(p)$ as an unbiased estimate of the average treatment effect $\tau(p)$, and then interpret $\tau(p)$ as if it were the TTE. This is what we refer to as "naïve" A/B testing.

This process gives an unbiased estimate of TTE only in the very special case when the outcome of a unit does not depend on the fraction of other units allocated to treatment. This is part of the Stable Unit Treatment Value Assumption (SUTVA) [95], and requires that $\mathsf{TTE} = \tau(p)$ for all p, and that spillovers are zero for all p. Visually, this process assumes that algorithms behave like Figure 3.1a and not Figure 3.1b.

Any A/B test that runs over a congested network has a clear pathway for interference between units in the treatment and control groups. Any explicit or implicit change in how the treatment group uses the congested network can create a different network condition for the control groups, which may lead to different behavior. This is *especially* true if the test explicitly changes the timing of how traffic is sent, or the amount of traffic that uses the network. Because of this, we will refer to violations of SUTVA as *congestion interference*.

Note on averages Average treatment effects, spillovers, and TTE are all defined as averages. Average here refers to the distribution of units in the A/B test, and not the outcome metric. The average treatment effect could measure the average difference in average latency, but it could also



(a) Units are applications using 1 or 2 long-lived TCP connections.

(b) Units are TCP connections which either pace traffic or not.

Figure 3.2: Throughput and retransmits in experiments where 10 units share a 10 Gb/s link. Every point on the x-axis is a different A/B test. All tests suggest a large change in throughput and no change in retransmissions, but the difference between 10 treated and 10 control units (TTE) is zero for throughput and large for retransmissions.

measure the variance of average latency or 99th percentile latency. Practitioners may also be interested in *quantile* treatment effects, e.g. the difference in 99th percentile latency between treatment and control. These are regularly estimated from A/B test results [179, 1]. It is straightforward to adapt our definitions to measure quantile treatment effects, and could be done by replacing $\mu_T(p)$ and $\mu_C(p)$ with quantile estimators.

3.3 Small Lab Experiments

When interference is present, naïve A/B tests do not accurately describe the behavior of a new algorithm. They mispredict the TTE and give no estimate of spillover. To illustrate this, we set up a small test network in the lab. The lab setup gives us a global view of how a new algorithm performs at any fraction allocation, and lets us recreate Figure 3.1 for actual algorithms. With these results, we can look at the results of different A/B tests, estimate TTE, and measure spillover. These experiments do not tell us how different algorithms would behave at scale, but they provide easy-to-understand examples of how congestion interference causes bias in naïve A/B tests.

Lab Setup Our lab consists of two servers running Linux 5.5.0, each with an Intel 82599ES 10Gb/s NIC. Each NIC is connected to a port of a 6.5Tb/s Barefoot Tofino switch via 4×10Gb/s breakout cables. The switch has a 1 BDP buffer. The sender server is connected to the Tofino with two 10G cables. The interfaces are bonded and packets are equally split between them, which ensures that congestion happens at the switch (otherwise we only see congestion at the sender NIC). We set MTUs to 9000 bytes so the servers can sustain a 10Gb/s rate. We add 1ms of delay at the

sender using Linux's traffic controller `tc`, and use iperf3 to generate TCP traffic.

3.3.1 Test 1: Multiple connections

Web browsers, video streaming clients, and other applications request data over multiple TCP connections in parallel. Making simultaneous requests reduces head-of-line blocking, reduces page load time, and increases utilization [156, 162, 75, 76]. This behavior depends heavily on the particular ways an application uses TCP connections and the particular networks it traverses, and so would typically be evaluated with a large-scale A/B test.

However, using multiple TCP connections can also allow an application to outcompete its peers and achieve higher throughput, and so is often called "unfair" in the academic literature [17, 28]. This makes it an ideal example to illustrate how congestion interference can bias A/B tests.

We ran an experiment in the lab to illustrate this behavior and understand the bias it causes. We ran eleven tests in which ten applications used either one or two TCP Reno connections to transfer bulk data. We measured the average long-term throughput and retransmission rates experienced by each application.

Figure 3.2a shows the results of the lab tests. Each test has two boxplots showing the average throughput for applications using one or two connections. Applications using two connections had 100% higher throughput and identical retransmission rates than applications using one. As more applications used two connections, their average throughput decreased. When all applications used two connections, their average throughput was identical to when all applications used one. Even worse, retransmission rates were higher when all applications used two connections.

These results are because of the way TCP fairly shares throughput between connections. If n identical TCP connections share a bottleneck link of capacity C, we expect each to receive a long-term average throughput of C/n. A group of flows with two connections should get a throughput of $2C/n$, 100% larger than C/n. But fundamentally, increasing the number of connections does not increase the capacity of the link so there can be no overall improvement.

This behavior is a well-understood consequence of TCP Reno's throughput fairness. But suppose we followed common practice [96, 36, 161, 34, 33, 138, 117, 53, 63, 52, 120, 104, 92, 127, 193, 119] and ran an A/B test to measure how using two parallel connections performed. To illustrate the potential for bias, we will use the same data set interpreted in a different way.

In a naïve A/B test, we would randomly allocate some fraction of traffic to treatment and the rest to control. Treatment would use two connections and the rest would use one. We would compare the throughput and retransmissions of the treatment and control groups. *No matter what allocation we picked*, we would see that two connections have a 100% higher throughput than one, and that there was no impact on retransmission rates. The naïve interpretation is that we should always use two connections in production.

TTE and spillover give us a better idea of how two connections perform. The TTE shows that

there would be no improvement in throughput and a 200% increase in the percentage of retransmitted bytes if all traffic were switched to two connections. Spillovers allow us to measure the impact of using two connections on other applications. When nine applications use two connections, the spillovers on the one remaining application using one connection are a 25% decrease in throughput and an almost 175% increase in retransmissions.

These results demonstrate that any single A/B test would not accurately measure the impact of changing the number of connections. But we should be careful not to extrapolate too much from the lab results. Applications may benefit from being more aggressive, but using multiple connections can also increase utilization. Without more experimentation, either could be a plausible explanation for a measured increase in throughput. Fundamentally, we believe that the only way to accurately measure the performance of such a policy would be to run an experiment at scale, on real traffic. We will discuss how to run such experiments later in Section 3.5.

3.3.2 Test 2: Pacing

Pacing is a generic, widely-used mechanism for reducing packet burstiness in a network [2, 151, 132, 33]. With pacing, a host adds delay between successive packets so that it sends a smooth, evenly paced stream of data into the network.

The Linux Kernel has supported pacing for TCP since 2013 [55, 54]. It adds delay between successive packets to ensure a rate of $2 \times cwnd/RTT$ during slow start and $1.2 \times cwnd/RTT$ during congestion avoidance [180].

Prior work, using ns-2, has shown that unpaced TCP traffic outcompetes paced traffic in terms of throughput [2, 191]. They recommend pacing at a rate of $(cwnd+1)/RTT$, which is implemented by Linux. These fairness concerns suggest that spillover may be nonzero, which implies that there would be congestion interference in an A/B test.

We ran pacing A/B tests in our lab to measure whether this interference still exists and if it would impact the results of an A/B test. Figure 3.2b shows the results. Paced traffic (the treatment) obtains 50% lower throughput than unpaced traffic (the control) in any A/B test, regardless of allocation. In each A/B test, we observed essentially no reduction in retransmissions for pacing.

Applying usual A/B testing practice to these results might have led us to decide not to deploy pacing. However, if we did deploy pacing, we would be pleasantly surprised to see no impact on throughput and a large decrease in retransmissions. The A/B tests also miss that pacing is good for other traffic: the spillovers from pacing are an increase in throughput and a decrease in retransmissions.

Pacing highlights the importance of estimating TTE when experimenting with networking algorithms. It is not obvious that pacing changes the way connections compete with each other: we expected it would smooth out bursts and cause lower RTT and loss with no impact on throughput. Without careful experiment design, an experimenter could be easily misled into thinking that pacing

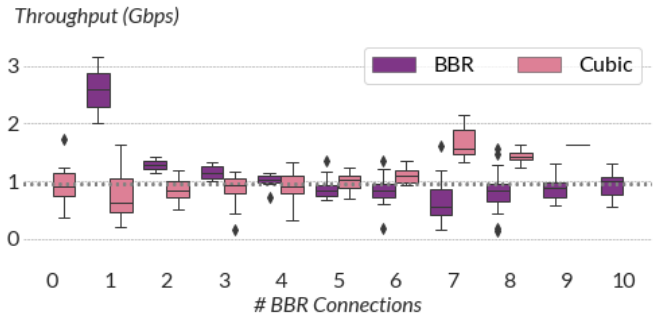


Figure 3.3: Experiments where 10 TCP connections using Cubic or BBR share a 10 Gb/s link. Throughput is the same if everyone uses either algorithm, but A/B tests suggest that both are improvements.

is not useful, or waste effort chasing a non-existent bug.

3.3.3 Test 3: Congestion Control Algorithms

There has been extensive study of the fairness of congestion control algorithms (*e.g.* [158, 189, 190, 30, 93, 28, 118, 13, 94, 51, 182, 183]). A treatment algorithm is often said to be unfair if it gets a larger share of throughput when competing against a control algorithm. In terms of our metrics, this would be if the spillover on control traffic is a decrease in throughput.

An A/B test will not accurately measure the TTE for an unfair algorithm. The treatment algorithm will take throughput away from the control, making the control perform worse than if the treatment were not present. Most widely-used congestion control algorithms are known to be unfair to at least some other algorithms in certain settings. The resulting biases undermine A/B tests on new congestion control algorithms at scale.

As an example, it's been widely reported that BBR is unfair to Cubic in certain situations [158, 94, 189, 190, 30, 93]. This unfairness suggests congestion interference, so we ran simulated A/B tests in our lab. We ran ten long-lived TCP connections, and allocated some fraction of them to BBR and the rest to Cubic. Figure 3.3 shows our results. If we were interested in deploying BBR in this setting and ran a 10% allocation, we would see a huge improvement in throughput. If instead we were interested in deploying Cubic and ran a 10% allocation, we would also see a huge improvement! But in this setting there is no difference in throughput between a global allocation to either BBR or Cubic.

3.4 Paired link experiment with bitrate capping

In response to the increased network usage during the beginning of the COVID-19 pandemic, Netflix worked with various governments to reduce load on the Internet, and rolled out a bitrate capping program which reduced video quality [64]. This program capped the video bitrate delivered to clients, while preserving the video resolution based on their subscription plans. It was observed that between March and June 2020, capping the bitrate reduced Netflix traffic in many countries by 25%, and reduced congestion for a number of ISPs.

In this section, we will describe a controlled experiment we ran to accurately measure the effects of bitrate capping. Given that bitrate capping reduced Netflix traffic by 25%, we suspected it would decrease congestion. Our preceding lab studies also led us to suspect that standard A/B tests may give biased results. So our goals with this experiment were to:

1. Measure the impact of bitrate capping on network performance and video quality of experience, by estimating TTE and spillover effects.
2. Estimate the bias of naïve A/B tests on these measurements, and
3. Evaluate whether alternate experiment designs would reduce this bias.

These are challenging goals to accomplish simultaneously. To evaluate the bias of a naïve A/B test and newer experimental designs, we need to measure what happens when all traffic is treated. But if we treat all traffic, we have nothing to compare against! We could run sequential experiments and compare their results, but this makes strong assumptions about how the system behaves over time. These would be useful assumptions to make when running alternate experiment designs, and we wanted to use this experiment to evaluate these assumptions.

In this section we describe the experiment we ran to achieve these goals. In Netflix's network, there are a pair of 100 Gb/s peering links to an ISP. The links are reliably congested during peak hours, and are statistically very similar. We treat these two links as "parallel universes," and can compare the outcomes of different experiments to investigate A/B test biases and congestion interference.

Our results are striking and sobering. Bitrate capping reduced congestion at the cost of slightly lower video quality, and improved the performance of uncapped traffic. This was almost completely undetected by naïve A/B tests which underestimated some treatment effects, failed to detect others, and, as we will see, even inferred the wrong *direction* of improvement for certain metrics.

3.4.1 Paired peering links

Netflix has a location with a pair of identical clusters, replicated for scale and redundancy. Each cluster is identically configured with a router and a number of cache servers. Each router connects to a partner ISP via a 100 Gb/s peering link. This setup is depicted in Figure 3.4.

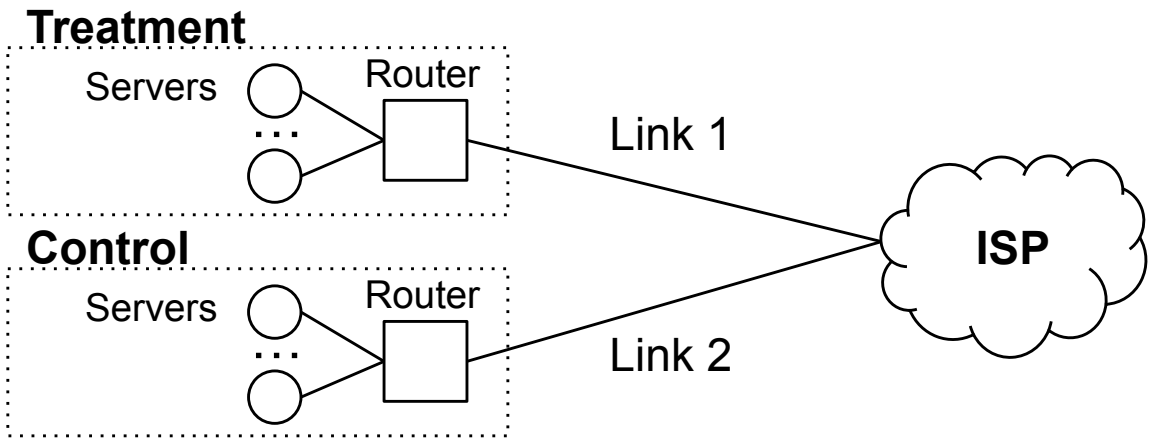


Figure 3.4: Diagram of the paired link experiment.

During peak viewing hours, demand from users connecting via this ISP increases until eventually a large standing queue builds up on both links. Latency increases, and throughput and video quality decrease. The congestion has a large impact on the quality observed by traffic, and we suspected strong congestion interference between connections sharing the same link.

A priori, we are not guaranteed that the two links will be similar to each other, since the system is optimized to serve video and not to run experiments. The content available on the two clusters is not identical, and different traffic is routed to the servers across each link. To validate statistical similarity between the two links, we collected data on both links during a week-long baseline period, comprising over five million sessions: 50.8% on link 1, and 49.2% on link 2. Netflix collects client- and server-side data on video performance. We looked at 24 important metrics including ones related to network performance (throughput, RTT, etc...) and video QoE (perceptual quality, stability, etc...). For each metric, we used the analysis approach described in Appendix 3.9 to compare links 1 and 2. We will discuss the most relevant subset of these metrics.

We obtained the following results, reported as means and 95% confidence intervals. Relative to link 2, link 1 had 5% (0.5%-10%) more overall bytes sent, a 2% (0.1%-3%) higher video stability metric, and 0.1% (0.03%-0.25%) lower perceptual quality. The largest differences were related to rebuffers. Rebuffers are moments when video playback is interrupted because the client is unable to download a piece of video from the server. Relative to link 2, link 1 had 20% (13-27%) more sessions with rebuffers; there were four additional metrics related to rebuffers that also exhibited similar differences. All other metrics did not have statistically significant differences. Notably, we did not see differences in most metrics we will discuss in our experiment below, including RTT, throughput, video bitrate, cancelled starts, or packet retransmissions.

Traffic on these links is not perfectly balanced, but it is clearly quite similar. Although the pre-existing differences in rebuffers is large, it is important to note that in absolute terms rebuffers

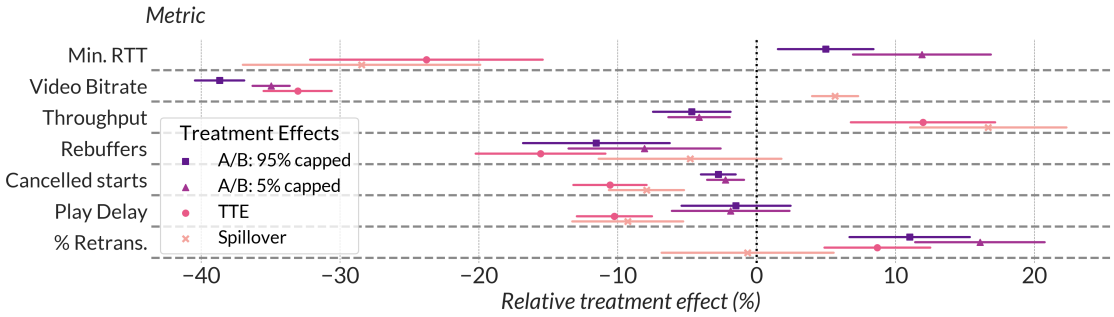


Figure 3.5: Treatment effects with 95% confidence intervals in our bitrate capping experiments. Each row is a metric of interest, with the naïve A/B Test estimates, and TTE and spillovers as estimated by the paired link experiment.

are rare. Given the similarity in other metrics, we believe they are caused by some other difference, such as the content served on the two links. Nevertheless, we carefully discuss our experimental findings regarding rebuffers in Section 3.4.3, where our observations suggest this difference in fact causes us to *underestimate* the extent to which naïve A/B tests are biased.

Being able to run an experiment like this is an extremely unusual situation. Operators work hard to avoid persistent congestion, so it is rare to have a pair of congested peering links. It is even rarer for the traffic to be balanced, and to be able to run separate experiments on each link. Netflix has hundreds of locations and thousands of peering links worldwide, but only *two* were suitable for this experiment.

3.4.2 Experiment design and analysis

We now describe the experiment we ran. Our goal was to estimate the effects when most traffic was capped, the TTE, and compare this to the results of A/B tests. We also wanted to measure the spillover of capped traffic on uncapped traffic.

To accomplish this, we ran a pair of A/B tests on the two links. On link 1, we allocated 95% of flows to treatment ($p = 0.95$). On link 2, we allocated 5% to treatment. Computing the naïve $\hat{\tau}(p)$ estimator on sessions *within* each link allows us to calculate $\hat{\tau}(0.95)$ and $\hat{\tau}(0.05)$. By comparing the mean of the 95% *treatment sessions on link 1* to the 95% *control sessions on link 2*, we obtain an approximate estimate of TTE. By comparing the mean of the 5% control sessions on link 1 to the 95% control sessions on link 2, we can obtain an approximate estimate of the spillover of capping. With this design, we ran A/B tests simultaneously on the pair of links. The experiment ran for five days, and included about fourteen million video sessions. We analyzed the experiment using techniques described in Appendix 3.9.

In practice, network experiments are usually run in one of two settings. The first is an initial

experiment with a relatively low level of initial treatment allocation, corresponding to the 5% A/B test. The second is a long-term holdback test, where almost all traffic is treated. We might naïvely hope that by treating more traffic, we would reduce congestion interference, and this corresponds to the 95% A/B test.

This experiment may at first appear a bit odd. We are measuring the difference in behavior when *almost* all traffic is capped and *almost* all is uncapped. This is an interesting quantity which tells us a lot about the behavior of bitrate capping during congestion, but it is only an approximation to TTE. The most straightforward way to estimate TTE in this network would be to cap 100% of sessions on link 1 as treatment, and uncap 100% of sessions on link 2 as control. We could then compare the means of each group to estimate TTE. However, if we did this, we would have no instances where capped and uncapped traffic shared a link, and we would be unable to compare the results to an A/B test or measure spillover. We could run other experiments other times on the links and compare the results, but we would be making strong assumptions about time invariance. This would require careful experimental design and analysis, and one of our goals here was to *validate* these designs.

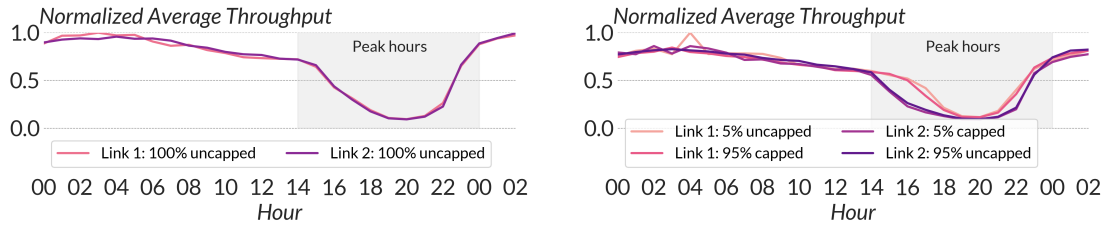
Putting it another way: one of our goals is to test the SUTVA assumption, and check whether treatment effects as measured by A/B tests give good predictions of what happens when an algorithm is widely deployed. If SUTVA holds, as in Figure 3.1a, spillover must be zero, and there must be no difference between the results of the two A/B tests and the approximate TTE we measure. If there is any difference between these quantities in our experiments, SUTVA cannot hold. Knowing that SUTVA does not hold, we would not expect slightly increasing the fraction of capped traffic to fix this problem.

3.4.3 Results

Our results can be summarized as follows: bitrate capping substantially reduced congestion and improved performance of uncapped traffic, and yet the naïve estimator would have largely failed to detect this.

Figure 3.5 reports our estimates of treatment effects and 95% confidence intervals for several important video streaming and network metrics. We report the results of 5% and 95% Naïve A/B test results (*i.e.*, $\hat{\tau}(0.05)$ and $\hat{\tau}(0.95)$), as well as our estimate of approximate TTE and our estimate of spillover. The naïve estimators are also wrong about the direction of improvement for minimum RTT and average throughput, and the magnitude of average play delay and video bitrate. The spillover is non-zero for most metrics.

Taking the example of average throughput, the two naïve A/B tests predicted a 5% *decrease* in throughput, which naïvely suggests that capping increased congestion. However, the TTE tells a very different story: that capping *increased* average throughput by 12%. Spillover shows that capping also benefited other traffic sharing the link: control traffic on the mostly capped link had



(a) Average throughput for the Saturday of the baseline test period.

(b) Average throughput for the Saturday of the main experiment.

Figure 3.6: Client-reported average throughput over time in the experiments, normalized to the largest hourly average. During peak hours, the links become congested and throughput decreases. Capping the majority of traffic in (b) causes Link 1 to be less congested and have higher throughput during most of the peak hours.

16% higher throughput than that on the mostly uncapped link.

These results can be explained by the way bitrate capping reduced congestion. There was significantly less capped traffic, so it took a larger number of users for the link to become congested. Since user demand was the same on both links, congestion started later, ended earlier, and was less severe on the majority-capped link. The naïve estimators were unable to detect this because both capped and uncapped traffic used the same congested link, and therefore saw similar performance.

This becomes clearer if we take a closer look at how the average throughput of sessions changes in Figure 3.6b, which can be contrasted with how the behavior during the baseline period in Figure 3.6a. We report the average of all client throughputs during each hour, normalized by the largest hourly throughput. Throughput slowly decreases as overall traffic increases throughout the day, and then suddenly drops when the link becomes congested during peak hours. During the baseline period, there is no difference between throughputs for the two links. During the main experiment, the mostly capped link remains uncongested for longer during peak hours, and has higher throughput before and after the most heavily loaded hours. Despite this difference, the capped and uncapped traffic on the same link have very similar performance.

In Figure 3.7, we show the four outcomes of throughput in the experiment: for capped and uncapped traffic as a function of allocation percentage. Both A/B tests confidently report that capped traffic reduces throughput relative to uncapped traffic. However by capping the majority of traffic, we improve throughput for all traffic using the link. This leads to an improvement as measured by TTE, and a positive spillover.

If we considered just one of the A/B tests in isolation, we would falsely conclude that capping traffic makes throughput slightly worse. This is our "smoking gun"—the confusion arises because treatment and control interfere with each other via congestion on the link.

We observed similar behavior for round-trip times in the experiment, as shown in Figure 3.8. During congested hours, large queues build up at the congested link, which causes all packets in

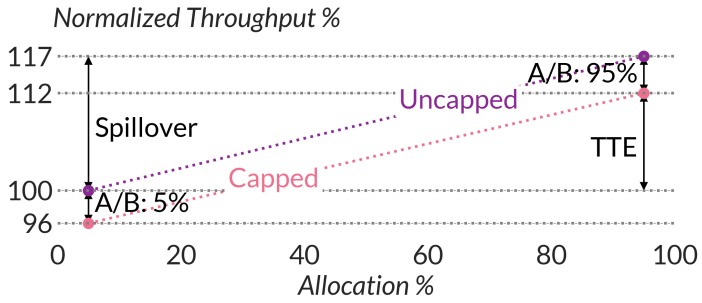


Figure 3.7: Average values of throughput in the cells in this experiment, with estimands of interest.

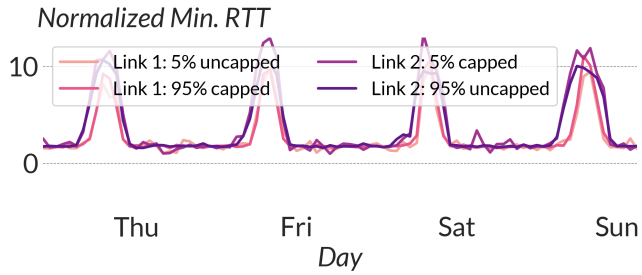


Figure 3.8: Average of minimum RTT in each connection, normalized to smallest cell value.

a session to be delayed, and leads to a sharp increase in the minimum RTT observed during each session. However, because bitrate capping delayed the onset of congestion, the majority-capped link (link 1) had empty queues for more time. The total treatment effect was a 24% *improvement* in the minimum RTT for the bitrate-capped sessions. The spillover was positive: capping traffic improved the minimum RTT by 27% for uncapped traffic. Again this was incorrectly estimated by the naïve A/B tests which both reported a 5% and 12% *increase* in minimum RTT.

We saw similar effects in start play delay, which is the time it takes a video to start playing. This is not surprising: improving throughput and reducing queueing delay should cause videos to load faster. Neither A/B test predicted a significant decrease in start play delay, whereas there was actually a 10% improvement in total treatment effect. The spillover was also positive: capping traffic reduced play delay by 9% for both itself and for uncapped traffic.

We measured a 33% reduction in video bitrate, with positive spillover. Capping the majority of traffic meant that the *uncapped* traffic was able to take up more bandwidth and achieve higher bitrates. It is surprising that despite the spillover, the two A/B tests still give reasonably good estimates of TTE. We believe this is because the majority of the reduction in bitrate comes from the artifical cap, which is applied independently of how other traffic behaves. The spillover is small relative to this effect, but might explain the difference between the 95% treatment effect and TTE.

We observed the total treatment effect for capping was a 10% *increase* in the fraction of sent

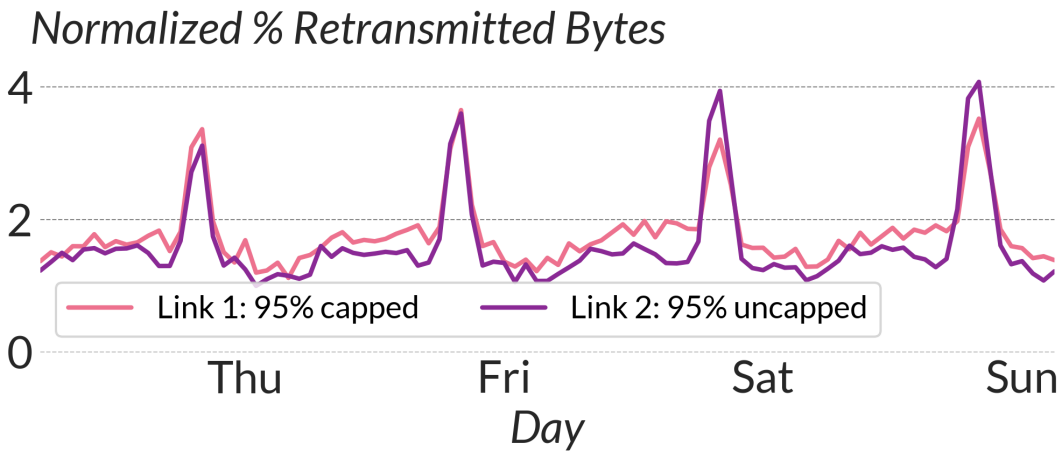


Figure 3.9: Capping bitrate generally reduced the fraction of retransmitted bytes during congested hours, but caused an increase in uncongested hours.

bytes that were retransmitted. This was driven by a 16% *increase* in the fraction of retransmitted bytes during off-peak hours, and a 20% *decrease* during peak hours as shown in Figure 3.9. This may seem surprising since bitrate capping reduced congestion, but in fact retransmits did not get worse. Capping reduced the *absolute* number of bytes retransmitted during both during peak and off-peak hours. The apparent increase in the percentage was caused by the absolute number of sent bytes decreasing more than the absolute number of retransmitted bytes. Although odd, Netflix observed similar behavior in a number of ISPs when removing bitrate capping.

Finally, we discuss the impact on rebuffers. Recall from Section 3.4.2 that we observed a 20% difference in rebuffers between the links from our baseline analysis prior to the experiment. Based on our experiment, we believe bitrate capping had at least some impact on rebuffers: we see a 15% decrease in rebuffers in the A/B tests within each link. We also measured that rebuffers for the mostly capped traffic in link 1 were 18% lower compared to the mostly uncapped traffic in link 2.

Given that rebuffer rates were not identical pre-experiment, we investigated further and measured rebuffer rates for both links during the month after we ran the experiment. We consistently found a difference: link 1 had on average 15% more rebuffers. In 70% of all hours, and in all but one peak hour, link 1 had more rebuffers than link 2. While we are not certain of the underlying reason for the difference, we believe an 18% improvement is probably an *underestimate* of the improvement of rebuffers. If we account for the underlying difference between links 1 and 2, it is closer to a 20%-30% improvement (rather than 15% improvement from the naïve estimate), suggesting congestion interference.

We conclude by highlighting one reason our results may underestimate the amount of congestion

interference. As discussed in Appendix 3.9, A/B test analysis usually assumes that sessions from different users are statistically independent of each other. By estimating standard errors only on data aggregated to the hourly level, our analysis effectively makes a nearly worst-case assumption that sessions in the same hour are *perfectly correlated*. This dramatically increases the size of the confidence intervals we report for TTE and spillover.

3.5 Unbiased Experiments at Scale

We care about two different things when evaluating a new algorithm: testing it safely and accurately measuring its performance. We want to experiment safely: if a new algorithm works so poorly that it could cause material harm to the service, we want to detect it quickly and avoid deploying it widely. We also want to be accurate: the goal of a new algorithm is usually to improve some metric, and we need to accurately evaluate whether it succeeded.

A/B tests are used today with the assumption that they are both safe and accurate. If the SUTVA assumption held, we can accurately estimate performance by running an A/B test on a very small fraction of users. This allows us to predict the performance of an algorithm at scale, without broadly deploying a harmful algorithm.

But in the worst case, congestion interference means that an A/B test is neither safe nor accurate. An algorithm which performs well in an A/B test might cause significant harm when it is deployed globally. But if an algorithm has marginal A/B test results and we do not deploy it globally, we may miss out on extremely effective algorithms.

This is a fundamental tradeoff with congestion interference, and what makes it so difficult to work with in practice. If we want to get a completely unbiased estimate of TTE, we need to allocate 100% of traffic to a treatment. But for safety reasons we would never allocate 100% of traffic to an untested or poorly performing algorithm.

In this section, we provide some guidance on how to run experiments in practice. We will not be able to completely resolve this tradeoff, but we will describe two ways of measuring congestion interference despite it.

Naïve A/B tests are biased in congested networks because of the combination of the A/B experiment design itself, and the flawed causal interference used when interpreting the results of that design. We will propose modifications to the A/B experiment design, and describe the improved causal inference that these modifications allow. First, we propose slightly modifying existing deployment practices to look for congestion interference. This is easy to do and helps build intuition around when congestion interference exists, at the cost of time-related bias and rejecting effective algorithms. To counter this, we also propose running small-scale, targeted switchback experiments to measure how a new algorithm behaves in a specific network.

3.5.1 Measure deployed algorithms with event studies

When deploying an algorithm, it is important to get an accurate estimate of TTE. Optimistically, an algorithm might perform better at scale than it did in small-scale evaluations. Perhaps when an algorithm is run by a larger fraction of traffic, it even further reduces congestion and improves performance than it did in small-scale experiments. Accurately quantifying the improvement is important to understanding its behavior and giving the team working on the algorithm the credit they deserve.

Pessimistically, a new algorithm might perform worse at scale than in small-scale evaluations. This might be a sign of some bug or unexpected behavior in the algorithm, and might suggest it increases congestion or interferes with other traffic on the internet. These are things that are important to know about, so they can be addressed.

Primarily for safety reasons, engineers have developed sophisticated techniques for deploying new algorithms. Engineers gradually deploy changes by slowly increasing the allocation fraction. They continually monitor the system, and stop the deployment if performance degrades.

While engineers typically use gradual deployments to safeguard against failure, they could also be used to conveniently measure the performance of a new algorithm and look for congestion interference. A gradual deployment is effectively a series of A/B tests with treatment allocations ranging from 0% to 100%. At each allocation (p_1, p_2, etc...) we can observe the outcomes for treatment and control. This gives us points on the graph of Figure 3.1, and we can use these values to estimate the average treatment effect $\tau(p_i)$, the spillover $s(p_i)$, and a *partial* treatment effect $\rho(p_i) = \mu_T(p_i) - \mu_C(0)$. Once the deployment is finished, we can compare 100% allocation to 0% allocation and estimate TTE. If there is no interference, for all allocations i and j, the average treatment effects are the same $\tau(p_i) = \tau(p_j)$, the partial treatment effects are the same as the average treatment effects $\rho(p_i) = \tau(p_i)$, and there is no spillover $s(p_i) = 0$. We can use statistical tests to check each of these relationships. If they do not hold, it could be a sign of congestion interference.

This is a type of observational design called an *event study* or an interrupted time series [114, Ch. 11]. In an event study, we introduce some change, and compare the state of the system before and after. This can be contrasted with a naïve A/B test, where we simultaneously compare units with and without the change. In the gradual deployment setting, the change is the increase of treatment allocation from p_i to p_{i+1}.

A major flaw with event studies is that it can be difficult to attribute observed behavior to a particular change. This is especially true because of seasonality: holidays, weekends, and political events all tend to have different traffic patterns than other times. Other teams or organizations regularly make changes and deploy software which can affect similar metrics. In the bitrate capping example, we had data from before and after deployment, but chose to run a more controlled experiment to rule out the possibility of other causes for the behavior we observed.

Another flaw is that this process works well for safely deploying new algorithms, but it is heavily

biased towards rejecting new algorithms. As an example, suppose we were testing a new algorithm which behaved like the pacing lab experiment in Section 3.3.2. In a small allocation A/B test, this algorithm would look worse: throughput would be down and loss would be unaffected. Seeing this, we might invest our time in other, more promising algorithms. We could slightly increase the size of the allocation to look for interference, but throughput increased quite slowly with allocation size. Even if we were able to detect this interference, it would look small. At this point, we might stop the deployment before the algorithm is able to clearly improve performance.

Despite these flaws, event studies are quick and easy ways to get estimates of TTE and spillovers. Large organizations continually deploy changes. When a deployment happens, it is easy to look at the already-collected metrics and use these metrics to estimate TTE and spillovers. Doing so will help build intuition around which algorithms could be affected by congestion interference.

3.5.2 Measure algorithms in development with targeted switchbacks

Running an event study when deploying a new algorithm is a good way to measure congestion interference and build intuition, but it is a bad way to experiment with new algorithms. We do not want to deploy marginal algorithms to all traffic, and so we may not invest in algorithms that perform poorly in an A/B test. We may miss out on algorithms that have very different effects when widely deployed, like bitrate capping, pacing, or changing the number of TCP connections.

Because of this, we recommend running small targeted experiments in addition to small A/B tests. A targeted experiment allocates a large fraction of traffic within a specific network. The network needs to be structured in such a way that the allocated traffic does not interact with non-allocated traffic. In the paired link experiment in Section 3.4, we targeted an experiment to two congested links. Using the results from the large fraction allocation, we can get a good estimate of TTE and spillover in this network.

Targeting an experiment allows us to estimate TTE and spillover within a network, without needing to run an algorithm on 100% of traffic globally. It is standard practice in online platforms [109, 155]. While we estimate TTE and spillover for a specific network instead of globally, this helps give additional context to A/B test results and improves our understanding of how a new algorithm behaves.

When running these targeted experiments, we recommend using *switchback designs*. A switchback design divides time into intervals; a given interval is randomly assigned to be either treatment or control. In a treatment interval, we treat almost all of the traffic with the new algorithm. In a control interval, almost all traffic runs the old algorithm.

At a high level, switchback experiments are analyzed by comparing the treatment and control intervals. While we could do 100% allocations in these intervals to get a good TTE estimate, we recommend a smaller allocation (e.g. 90-99%) as in the paired link experiment. Doing so allows us to additionally estimate spillover and the bias of A/B tests, which gives valuable insight into algorithm

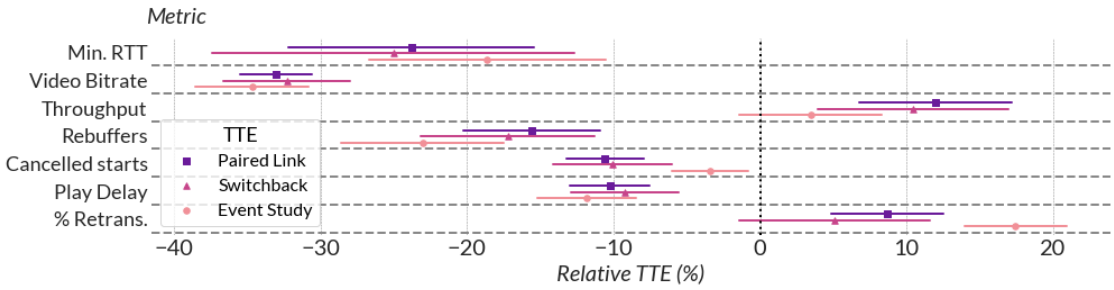


Figure 3.10: TTE as estimated by the paired link experiment, a switchback experiment, and an event study.

behavior. The allocation size should be large enough to give statistically significant results, and can be determined by a power calculation.

Like event studies, switchback experiments rely on the change between treatment and control intervals being due to the treatment. However, the assumption is weaker: instead of needing no other events to impact the outcome, a switchback requires that another event does not line up with the treatment intervals.

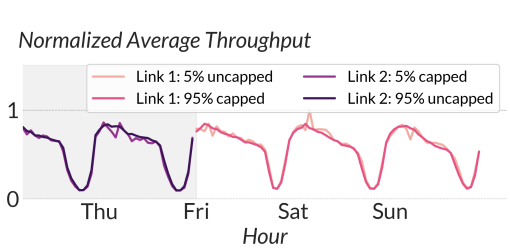
A switchback experiment can also be vulnerable to carryover effects [71, 26]. The presence of the treatment algorithm can influence the initial conditions of the control algorithm and vice versa. This can cause bias: imagine if we were to switch sessions between one and two parallel connections. Until all sessions that used two parallel connections had completed, the sessions using one would have lower throughput than necessary. If the system reacts poorly to switching between treatment and control, this could also cause problems.

Carryover effects can be mitigated with sufficiently long intervals. However, typically switchback experiments make the worst-case assumption that all sessions in an interval are dependent (see Appendix 3.9 for more details), which essentially means that each interval gives us one data point. Increasing the length of intervals effectively lowers the sample size of the experiment. For networking algorithms, we believe a switch interval of one day is a reasonably conservative place to start. Depending on the setting and the algorithm, it may be appropriate to use a shorter interval on the order of hours or minutes.

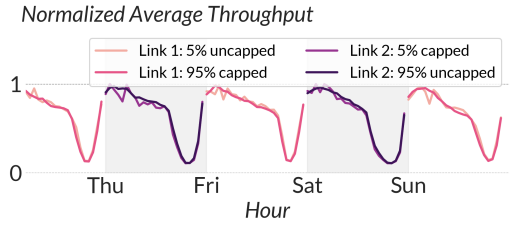
3.5.3 Evaluating alternate designs

Our paired link experiment gives us the results of simultaneous, comparable experiments. We previously analyzed that data to estimate TTE and spillovers. We now use it to evaluate event studies and switchback designs, and show that these designs also accurately estimate TTE.

Having two simultaneous experiments allows us to ask what *would* have happened if we ran only one experiment at a time. Our experiment in Section 3.4 ran from Wednesday through Sunday,



(a) Throughput in a bitrate capping event study. Between Thurs. and Fri., we apply 95% bitrate capping.



(b) Average throughput over time in a bitrate capping switchback experiment. 95% of traffic is capped on the first and third and fifth day.

Figure 3.11: Alternate experiment designs used to estimate TTE.

giving us five possible days of data. We can emulate an event study by using data from the 5% link for a few days and then switching to data from the 95% link, representing a deployment of bitrate capping to 95% of traffic. We can emulate a switchback experiment by switching between treatment days and control days more frequently.

We first used baseline data to calibrate a switchback experiment. We ran an A/A test [114, Ch. 19] on the paired links in the week following our main experiment: we applied the control to both links and looked for underlying differences. Using the data from the A/A test, we checked that there would have been no false positives with any switchback design. This increases confidence that there isn't a reliable difference between days in a way that would bias the experiment, and we would recommend doing this in most cases.

We also used baseline data to calibrate an event study. We observed that there were false positives in the majority of metrics with any event study in this experiment. We believe this is because weekends tend to have different traffic patterns than weekdays, and an event study must either treat all the weekend days or all the weekdays together. This is an advantage of using a switchback design.

For the event study, we switched to 95% bitrate capping between Thursday and Friday as shown in Figure 3.11a. For the switchback, we alternated between treatment and control, and randomly started with treatment. This assignment is shown in Figure 3.11b. All other ways of assigning treatment to days yielded similar results, provided at least one day was in treatment and at least one day was in control.

Figure 3.11b shows the average throughput for this example switchback design, which can then be compared with the throughput in the paired link experiment in Figure 3.6. Note that because we are switching between experiments, the clear difference in throughput in the paired link time series is much harder to see in the switchback. This highlights the power of running statistical analyses on switchback data.

Our goal with this approach was to use the clean results from our paired link experiment to

demonstrate the power of switchback experiments and event studies. If we had actually run these experiments, the results may have been slightly different. For instance, traffic from both links likely shares some bottlenecks in the provider network during offpeak hours, so it is possible that our results during offpeak hours are biased by congestion interference. However the congestion interference we detect is largely because of the behavior during congested hours on isolated congested links.

3.5.4 Results

The analysis approach for these experiments is identical to the paired link experiment, with the caveat that we only use the subset of the data corresponding to each experiment. We describe the details in Appendix 3.9.

Figure 3.10 shows the values of TTE estimated by the switchback experiment, event study, and paired link experiment. Both alternate experiments give reasonably good estimates of TTE. The switchback experiment results are very close, and the confidence intervals for its estimates include every TTE from the paired link experiment. It has larger confidence intervals because it includes half as much data. We expect that running the experiment for longer would have reduced the size of the confidence intervals.

The event study gives reasonably accurate estimates of TTE for most metrics, but is biased for throughput, cancelled starts, and % retransmitted bytes. As we observed in analyzing the baseline data, we believe this is because of seasonality issues: weekends tend to have different behavior than weekdays, and so it is more difficult to attribute the change to the treatment. This is one of the advantages of switchback experiments: randomly choosing intervals over many days helps avoid certain seasonality effects. Despite this, given that event studies are so easy to incorporate into existing workflows, we still recommend cautiously using them to estimate TTE and spillovers when deploying new algorithms.

3.6 Related Work

A/B tests are heavily used in industry research. There recently have been a number of published A/B tests comparing congestion control algorithms, including BBR [96, 36, 161, 34, 33], COPA [138], and Swift [117]. There have also been many other published A/B tests for other networking algorithms. These include work on initial congestion windows [53], TCP's loss recovery [63], PRR [52], QUIC [120, 104], failure recovery [119], and ABR algorithms [92, 127, 193]. We do not know how congestion interference affected these results.

We are aware of a few published results that include event studies: Dropbox and Verizon both used them to evaluate BBRv1 [96, 161], and Google reported one for Timely in [132]. In Section 3.5, we show how to design and analyze these event studies to measure TTE and spillover, and describe how switchback experiments give more reliable results.

Experiments on router performance, especially those related to buffer sizing [21, 166, 20], naturally must treat all traffic using the router. Because of this, they tend to have good estimates of total treatment effects.

Recent studies of social network and marketplace platforms have led to improved understanding of causal inference under interference (e.g., [125, 10, 14, 18, 23]), both through novel experimental design (e.g., [184, 102, 16, 26, 71, 88, 37, 155]) and improved inferential methodology (e.g., [14, 19, 18, 177]). We believe our work is the first to show that these issues affect networking experiments and bias their results at scale.

Switchback designs found recent favor as an approach to testing matching and dispatch policies in ridesharing and food delivery platforms, though they have also been used in applications as varied as agriculture [37, 109, 26, 148, 139]. We are unaware of any prior usage of switchbacks in networking.

We have heard some folklore predictions from the networking community that these sort of issues may exist. The only citeable version of this we know of is in [181].

Finally, our work is informed by the long line of work on fairness in networking. Unfairness between Cubic and BBR, which we describe in Section 3.3, was previously reported by [158, 94, 189, 190, 30, 93, 182, 183]. Unfairness between parallel connections was first observed by [17]. Unfairness between paced and unpaced Reno flows was shown by [2, 191]. Fairness work is about how algorithms *ought* to share resources, and usually shows that algorithms are unfair in simulations or in a lab [28, 118, 13, 158, 94, 51, 190, 30, 93, 105, 182, 183]. Our work does not address how algorithms *should* share resources, but rather how to avoid experimental bias when they *do*. One way of interpreting our work is as a way to measure unfairness between treatment and control at scale, in production networks.

3.7 Conclusion

Congestion interference biases the results of networking A/B tests at scale, and it is our responsibility as a community to be aware of this phenomenon. Our results suggest that we should be skeptical when interpreting the results of naïve A/B tests, and consider whether alternate experiment designs should be used instead.

As discussed in Section 3.5, experimenters can make small changes to existing deployment processes to begin to measure congestion interference, and use targeted switchbacks to further improve these measurements. We should be especially wary of interference when an algorithm changes traffic volumes, tries to control congestion, or is similar to algorithms discussed in the past fairness research in Section 3.6.

We would love to see more work in networking evaluated with congestion interference in mind, either with published switchback experiments, or at least event studies run during a gradual deployment. This is especially true for high consequence proposals, such as new internet standards.

On the research side, there is much more work to be done on evaluating algorithms at scale in congested networks. We encourage further studies to measure bias, in different networks and with different algorithms. We think it would be valuable to design new experiments and analyses specifically for congested networks. The bias of naïve A/B tests is both a cautionary tale and a significant opportunity for innovation. The internet surely works better thanks to A/B tests of algorithms run in congested networks. We hope that new algorithms tested with better experiments will help improve it even further.

3.8 Ethics

While our experiments involve live traffic running on a large video streaming service, our work is not human subjects research, and we have no way to identify the individual users of the platform. We only have access to performance-related data. We ran experiments which improved behavior during congestion, but they did so at the cost of reducing video quality. Netflix's customers have the ability to opt out of experiments, if they choose to.

3.9 Appendix: Analysis of experimental data

In this appendix we describe our general approach to analysis of data from experiments at scale, and how we apply this approach in the context of the experiments reported in Sections 3.4 and 3.5. For the duration of the appendix, we consider data for a fixed representative metric collected on a per-session basis (e.g., average throughput).

In our experiments units are video sessions, and we let A_i denote the treatment condition of session i, where $A_i = 1$ denotes treatment and $A_i = 0$ denotes control. Let Y_i denote the observed outcome on session i. Let $h_i \in \{1, \ldots, 24\}$ denote the hour of session i. Our first step in analysis is to aggregate data at the *hourly* level: for each hour $t = 1, \ldots, 24$ and each treatment condition $A = 0, 1$, we compute:

$$Z_t(A) = \frac{\sum_i Y_i \mathbb{1}_{h_i=t, A_i=A}}{\sum_i \mathbb{1}_{h_i=t, A_i=A}}.$$

This is the average outcome for sessions in treatment condition A during hour t.

Next, we use a regression approach to estimate the treatment effect [68, Ch 9], using the following model specification:

$$Z_t(A) = c + \beta_0 A + \beta_t + \varepsilon_i, \quad \text{for all } t, A.$$

Here $t = 1, \ldots, 24$ and $A = 0, 1$; β_0 is the coefficient on the treatment indicator; each β_t is a fixed effect to control for hour-of-day heterogeneity; c is an intercept term; and ε_i is the error term. We fit this model using least squares linear regression, and estimate confidence intervals using Newey-West

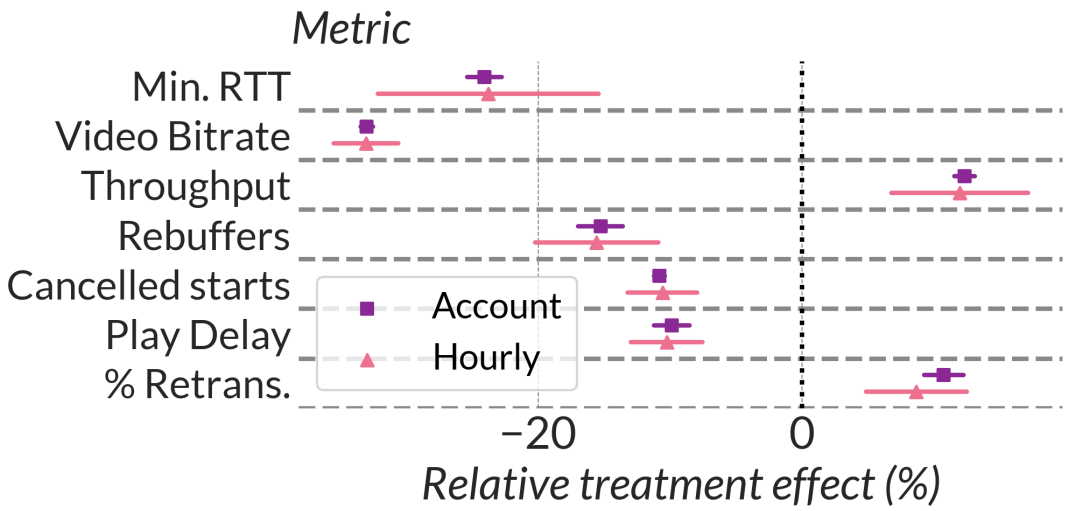


Figure 3.12: Comparison of treatment effect sizes and confidence intervals when aggregating by hour or by account.

robust standard errors [136] with a lag of two hours. This is a common approach in econometrics to account for autocorrelation between successive hours, and heteroskedasticity in the error terms ε_i. We use hats to denote the corresponding estimates; in particular, $\hat{\beta}_0$ is the estimated coefficient on the treatment indicator, and thus an estimator for the average treatment effect.

We note that the approach we take here—where we aggregate data to the hourly level—essentially makes a worst case assumption that sessions within a given hour and treatment condition are *perfectly correlated* with each other. This is a very conservative assumption, that we feel only strengthens the case in our paper. Though conservative, this is current practice in analysis of switchback experiments in other industries [109]. If we were to analyze the results using the standard account-level standard errors, we would get much tighter confidence intervals as shown in Figure 3.12. Correcting standard error estimates to properly estimate dependencies between sessions remains an active area of investigation.

We now describe how we apply this approach to our experiments in Sections 3.4 and 3.5.

3.9.1 Application to paired link experiment

In Section 3.4, sessions on link 1 were randomized 95% to treatment and 5% to control; and sessions on link 2 were randomized 5% to treatment and 95% to control.

We carry out four separate analyses on this data. First, to compute the approximate estimate $\widehat{\mathsf{TTE}}$ for TTE, we consider the 95% of all sessions in the treatment group on link 1 as our treatment sessions ($A_i = 1$); and the 95% of all sessions in the control group on link 2 as our control sessions

($A_i = 0$). We ignore all other sessions. We then follow the analysis workflow above, and set $\widehat{\mathsf{TTE}} = \hat{\beta}_0$ from the resulting fitted regression.

To estimate spillover, we use only the 5% control sessions on link 1 and the 95% control sessions on link 2. We set $A_i = 1$ for the control sessions on link 1, and $A_i = 0$ on link 2. We compute $\widehat{s}(0.95) = \hat{\beta}_0$ from the resulting fitted regression.

Finally we compute two "naïve" estimates using the difference in means estimator (3.1) from Section 3.2. In particular, for $p = 0.95$, we use only the sessions on link 1: we consider all sessions in the treatment group on link 1 as our treatment sessions ($A_i = 1$), and all sessions in the control group on link 1 as our control sessions ($A_i = 0$). All sessions on link 2 are ignored. An analogous approach is carried out for $p = 0.05$ using the treatment and control sessions on link 2 (ignoring all sessions on link 1), to compute $\hat{\tau}(0.05)$. We aggregate to the account level, not the hour level, as is standard when analyzing A/B tests.

Finally, all reported values are normalized to make them more interpretable. In particular, we divide all estimates by the average across all control sessions on link 2 (where 95% of the traffic was control). This approach ensures all reported values are a relative difference measured against the same global control condition.

3.9.2 Application to switchback experiments and event studies

In Section 3.5, we analyzed a switchback experiment and an event study that was emulated using the data from the paired link experiment. This analysis was carried out as follows. For the three days chosen to be treatment intervals, we define all treatment sessions on link 1 to have $A_i = 1$, and ignore all other sessions. For the two days chosen to be control intervals, we define all control sessions on link 2 to have $A_i = 0$, and ignore all other sessions. We then proceed with the analysis workflow above, and report $\hat{\beta}_0$ as our emulated estimate of TTE.

Chapter 4

Updating the Theory of Buffer Sizing

4.1 Introduction

Internet routers have packet buffers which reduce packet loss during times of congestion. Sizing the router buffer correctly is important: if a router buffer is too small, it can cause high packet loss and link under-utilization. If a buffer is too large, packets may have to wait an unnecessarily long time in the buffer during congested periods, often up to hundreds of milliseconds. While an operator can reduce the operational size of a router buffer, the maximum size of a router buffer is decided by the router manufacturer, and the operator typically configures the router to use all the available buffers. Without clear guidance about how big a buffer needs to be, manufacturers tend to oversize buffers and operators tend to configure larger buffers than necessary, leading to increased cost and delay.

This paper revisits two widely used rules of thumb for sizing router buffers in the internet. The two rules cover two different cases:

Case 1: When a network carries a single TCP Reno flow. Van Jacobson observed in 1990 [98] that a bottleneck link carrying a *single* TCP Reno flow requires a router buffer of size $B \geq \text{BDP}$, the bandwidth-delay product, in order to keep the link fully utilized.

Case 2: When a network carries multiple TCP Reno flows. Appenzeller, Keslassy, and McKeown argued in 2004 [8] that a bottleneck link carrying n long-lived TCP Reno flows requires a buffer of size $B \geq \text{BDP}/\sqrt{n}$ in order to keep the link highly utilized.

Much has changed since these rules were first introduced, and it is not clear whether these rules still apply in modern networks. The behavior of TCP Reno has changed; most notably when Rate-Halving [86, 160] and PRR [52] were introduced. New types of congestion control have become widespread, such as Cubic [78] (default in Linux, Android, and MacOS), and more recently BBR

(deployed by Google for YouTube) [33] and BBRv2 [35]. Given that the analysis underlying both buffer sizing rules depends on the specific way in which TCP Reno halves the congestion window when losses are detected, there is no particular reason for either rule to still hold in today's internet.

Existing rules of thumb help us pick the buffer size to achieve full link utilization, and do not predict behavior if the buffer is made smaller. Thus, theory falls short for recent congestion control algorithms (e.g. BBR and BBRv2) which no longer aim to keep a bottleneck link running at 100% utilization. Instead, they rely on short periods of under-utilization to keep queueing delay low and to estimate propagation delay.

In light of these changes, this paper examines buffer sizing for modern TCP algorithms. We show that the two rules still allow TCP Reno to fully utilize a link, despite changes due to Rate-Halving and PRR. We show that TCP Cubic, Scalable TCP, and BBR allows us to reduce buffer sizes.

We extend our analysis for the case when link utilization is less than 100%, and we show that very small buffers can still allow high (but not 100%) link utilization. More generally, *this paper sheds new light on how to size buffers for a given congestion control algorithm and desired link utilization, under a very broad set of conditions.* In doing so, we also show how future congestion control algorithms can be designed to further reduce buffer requirements.

Throughout the paper, we will illustrate and validate our results using measurements drawn from a physical network in our lab. This is challenging: while Linux can capture per-packet measurements in the end-host TCP stack, it is not normally possible to capture the full time series of buffer occupancy at the switch. Our measurement setup uses a P4-programmable Tofino switch which we program to report the precise time evolution of its buffer, to approximately 1 nanosecond resolution. This lets us precisely compare the evolution of the congestion window and the buffer size and validate our theoretical results. We start in Section 4.2 by describing this experimental setup.

In Section 4.3, we revisit case #1: a single TCP flow. Specifically, we show theoretically and experimentally that despite the introduction of PRR to TCP Reno, the $B \geq$ BDP result still holds. We also show that with TCP Cubic we can reduce the buffer size to 0.4BDP for a single flow, and for BBR we can reduce it further to 0.25BDP and still achieve full link utilization. We also describe how link utilization behaves when buffers are below these values.

Next, in Section 4.4, we examine case #2: multiple TCP flows. We prove that algorithms which respond to full queues (via losses or marks) create standing queues with particular properties during times of congestion. We show that the square root of n rule is a consequence of this behavior: if queues are always close to full for any buffer size, then intuitively the buffers can be shrunk without impacting utilization. Another, perhaps surprising, consequence is that link utilization is not a "cliff" function. We show that if the buffer is slightly too small for full utilization, then utilization still remains high. Specifically, we show that even with very small buffers, the utilization percent is at least $\Omega(1 - 1/\sqrt{n})$. We also show that design choices in BBR allow for a square root of n rule essentially without assumptions.

In Section 4.5 we support our results with fine-grained measurements from lab experiments. We show that the square root of n rule holds in our experiment, and show how loss-based algorithms keep queues full during congestion. We verify the square root of n result for BBR, and find experimentally that a BDP/n rule may be more accurate, allowing for even smaller buffers. We show that buffer sizes depend on how flows synchronize and whether they are fair. We describe how to check the assumptions in our theorems, and show that they hold in the lab. We also caution that enabling ECN can increase synchronization in certain cases, resulting in a larger required buffer, and show lab results to support our analysis.

In Section 4.6, we discuss measurements of large-scale production networks which show that RTTs are persistently elevated during times of congestion [124, 61, 45]. Our results explain why congestion control algorithms create these persistently full queues, and suggest that these queues can be made smaller. We also discuss the implications of our results on other situations when it may be more difficult to measure congestion.

In Section 4.7 we make recommendations to network operators who are interested in running buffer sizing experiments, and describe how our results can be used to easily design these experiments. We also give recommendations to designers of new congestion control algorithms who would like to ensure small buffer requirements. We discuss related work in Section 4.8 and conclude in Section 4.9.

Contributions: The main contributions of this paper are:

1. Single flow case: A simple proof of TCP's required buffer size, applicable to the latest versions of TCP Reno, as well as other algorithms including Cubic, Scalable TCP, and BBR.

2. Multiple flow case: A new, more general model of how buffer size is impacted by fairness and the amount of worst-case packet drops, and square root of n-style rules for TCP Reno and other algorithms.

3. A better understanding of how congestion control algorithms interact with buffers, including how utilization depends on buffer size, how algorithms can reduce buffer requirements, and how current congestion measurement techniques rely on certain algorithmic behavior.

4. A new measurement platform allowing precise observation of TCP and the router buffer.

4.2 Experiment Methodology

Throughout the paper, we will use measurements from our physical testbed to illustrate and validate our results. We have built a platform on programmable hardware which allows us to easily run experiments where TCP flows share a congested link, and observe how TCP and queues behave at a packet-by-packet level.

Min. Buffer Size	**Additional assumptions beyond Section 4.3.3**	**Citation**
BDP	Reno, silence after loss	[98, 185]
BDP	Reno	[47], Section 4.3.5
$(1/b-1)$BDP	Multiplicative decrease by b, silence after loss	[83, 121, 130]
$(1/b-1)$BDP	Multiplicative decrease by b	Section 4.3.5
$\frac{3}{7}$BDP	Cubic	[121], Section 4.3.5
$\frac{1}{7}$BDP	Scalable TCP	Section 4.3.5
$\frac{1}{4}$BDP	BBR during the probe bandwidth phase, with loss	Section 4.3.5
$\Theta(BDP/\sqrt{n})$	Reno, windows are i.i.d. uniform random variables	[8]
$O(\text{BDP}/\sqrt{n})$	$\sqrt{n}+O(n^2/BDP)$ almost fair flows see loss	Theorem 7
$O(\text{BDP}/\sqrt{n})$	Almost fair BBR flows in probe bandwidth phase	Theorem 9
$O(p\cdot\text{BDP}-n+np)$	A p fraction of fair flows see losses	[47]
$O(s\cdot\text{BDP}/n)$	At most $s+n^2/BDP$ almost fair flows see loss.	Section 4.4.6
$O(1)$	Reno, bounded window size, Poisson pacing	[60]
$O(1)$	$\sqrt{n}+O(n^2/BDP)$ almost fair flows see loss, utilization is $\Omega(1-1/\sqrt{n})$	Theorem 8

Table 4.1: Minimum buffer sizes required for full link utilization, our new results highlighted in gray.

While working on this paper, we were frequently baffled by experimental results that didn't match our understanding of TCP. Almost always, when we dug into the measurements, we found that TCP's actual behavior did not match our understanding. We hope that by including our measurements, we can make TCP's behavior (and our results) more understandable.

Setup Our test network consists of two servers with 32 2.4Ghz cores and 32 GB of RAM each, connected by a Barefoot Tofino switch and use up to 2 MB of buffers. The servers run Linux 5.5.0, each with an Intel 82599ES 10Gb/s NIC. Each NIC is connected to a port of a 6.5Tb/s Barefoot Tofino switch via 100G to 4×10Gb/s breakout cables. The sender server is connected to the Tofino with two 10G cables. The interfaces are bonded and packets are equally split between them, which ensures that congestion happens at the switch (otherwise we only see congestion at the sender NIC). We set MTUs to 9000 bytes so the servers can sustain a 10Gb/s rate. We add 1ms of delay at the sender using Linux's traffic controller `tc`, and use iperf3 to generate TCP traffic. We used congestion control algorithms available in Linux 5.5.0, including TCP Reno, Cubic, BBR (v1), and Scalable TCP. We also used Google's alpha release of BBR2 [35].

Measuring TCP Since 2017, Linux has had a tracing system which reports the congestion window, smoothed RTT, and other important information for each received acknowledgement [84]. We used this system to observe TCP's behavior in our experiments, and have created a small open source tool to make them easier to work with: https://github.com/brucespang/tcp_probe.

Measuring Buffers It is usually hard to precisely measure a switch's buffer occupancy over time. We wrote a P4 program running on the Barefoot Tofino switch, to measure (and report) the time series of the packet buffer occupancy. Each time the switch receives a packet it sends a small

UDP packet (often called a "postcard") to a collector which includes the current buffer occupancy and some packet metadata. At the collector, we record these postcards and match them with the TCP trace samples. Our ability to measure the complete time-series of a buffer is not novel (e.g. [81, 20, 112, 39, 29]). We believe this sort of fine-grained measurement should be a standard part of any TCP experiment.

Datasets and Source Code We have released our P4 code, infrastructure for collecting TCP traces, and the traces we collected online at https://github.com/brucespang/ifip21-buffer-sizing.

4.3 Sizing buffers for a single flow

Since the 1990s, it has been known that a network carrying a single TCP Reno flow requires a buffer size of one BDP in order to keep the bottleneck link fully utilized. However, TCP has changed dramatically since then, with Proportional Rate Reduction [52] and new congestion control algorithms like Cubic [78] and BBR [33]. It is not clear whether the buffer requirements for a single TCP flow have also changed.

In this section, we revisit the classic rule of thumb that a buffer should be at least one BDP for a single TCP connection. Some of the results in this section have been shown by prior authors, and we revisit their results to verify that they still hold experimentally, and as a warm-up for later sections. We find that despite the changes to the Linux kernel, the BDP rule still holds for TCP Reno. We find that a BDP is an *overestimate* of the buffering required for Cubic and BBR.

More specifically, we will

1. Show that the BDP rule still holds for a modern implementation of TCP Reno.
2. Show that Cubic requires a buffer of only $\frac{3}{7}$BDP and BBR requires a buffer of only $\frac{1}{4}$BDP.
3. Show how link utilization changes when buffers are smaller than required for 100% utilization.
4. Show that BBR2 does not fully utilize a link, but instead aims for high link utilization.
5. Show how the usual proof of the BDP rule depends on outdated TCP behavior, and give a simpler and more general proof.

4.3.1 The original BDP rule

The original BDP rule, attributed to Jacobson [98] and Villamizar and Song [185], states that for a single TCP Reno flow to fully utilize a link with a drop-tail buffer, the buffer must be at least the capacity of the link C times the RTT. This is the bandwidth delay product, BDP $= C \cdot RTT$.

Fact (BDP Rule)**.** *TCP Reno fully utilizes a link if and only if* $B \geq$ BDP.

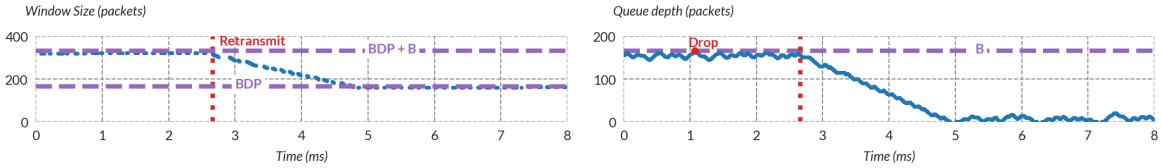


Figure 4.1: Per-packet window size (from kernel) and queue depth (from switch), for one TCP Reno flow with a BDP sized buffer. PRR keeps sending packets after a loss, but the queue still drains.

Most proofs of the BDP rule [98, 8] rely on TCP halving its window size W on a loss, and then waiting to send another packet until it receives $W/2$ acknowledgements so that the number of packets in-flight drops below the new window size. But this is not how modern TCP implementations work. For example, TCP Cubic decreases its window by less than a half on a loss. Even TCP Reno no longer stops sending packets on a loss. Instead it behaves according to Proportional Rate Reduction (PRR) [52], introduced by Dukkipati et al. in 2011, which supplanted Rate-Halving, proposed by Hoe [86] in 1995 and as an RFC by Mathis et al. [160] in 1999. Both algorithms gradually lower the size of the window after receiving a loss. When PRR decides to decrease the congestion window from w to w', it sends a new packet for every w/w' packets acknowledged. For TCP Reno, this means a new packet is sent for every two packets acknowledged. Most deployments of TCP Reno now use PRR.

4.3.2 Does the BDP rule still hold experimentally?

Our measurement infrastructure allows us to observe the behavior of PRR after a loss, which is shown in Figure 4.1 for TCP Reno (with PRR) and buffer size $B = \text{BDP}$. The figure is zoomed in to show what happens when a packet is dropped by the switch at about the 1ms marker. The TCP source detects the loss about 1.5ms later, and begins to gradually decrease its window over the next 2ms. After 2ms (about 2 RTTs), the window size has been halved and the queue is nearly empty.

Without PRR, TCP Reno stops sending packets as soon as the drop is detected and remains silent until the window is reduced by half. Even though TCP with PRR never stops sending, the queue almost completely drains, which suggests that a BDP worth of buffering is still necessary. In what follows, we prove this.

4.3.3 Technical Preliminaries

In order to prove that $B \geq \text{BDP}$ is still necessary with PRR, we need an argument that does not rely on TCP stopping sending after a loss. To this end, we set up some definitions that will help us prove our results throughout the paper. These are standard assumptions in the TCP and the buffer sizing literature, and apply to the common case when n TCP flows share a drop-tail queue.

Throughout the paper, we will refer to a *TCP flow* in theorem statements, and use it to mean

the following set of assumptions and definitions, plus commonly known TCP behavior.

Definition. *A packet is* in-flight *if it has been sent, but the acknowledgement has not yet been received by the sender. Let $W(t)$ be the aggregate window, or the number of in-flight packets at time t.*

For the case of n flows sharing a link, let $w_i(t)$ be the window of flow i, i.e. the number of in-flight packets for flow i at time t. Therefore the aggregate window is $W(t) = \sum_{i=1}^{n} w_i(t)$.

Throughout the paper, we will heavily use the following behavior of TCP Reno: when TCP Reno successfully sends a packet and receives an acknowledgement, it increases its window from w_i to $w_i + 1$. When it detects a congestion signal (e.g. a dropped packet or an ECN mark), it decreases the window from w_i to $w_i/2$.

Note that our definition of the window of flow i is the number of in-flight packets, which is not necessarily the same as the congestion window (*cwnd*). For example, when TCP Reno reduces its congestion window in response to a loss, the number of in-flight packets may briefly exceed the congestion window.

Definition. *The RTT at time t is the time taken from when a packet is sent at time t until we receive its acknowledgement. Our results assume that all flows have the same RTT.*

Definition. *The* sending rate *of flow i at time t is $x_i(t) = w_i(t)/RTT(t)$. The aggregate sending rate is $x(t) = W(t)/RTT(t)$.*

Links have queues to store packets before they are sent. We will also call them buffers. Packets are put into these queues as they arrive, and are later sent from the link. Queues have a length at any point in time, which we call the queue length, and a maximum number of packets they can fit before they need to drop packets, which we call the maximum queue length or buffer size.

Definition. *Let $Q(t)$ be the length of the queue at time t.*

A link with capacity C sends one packet from the queue every C^{-1} seconds, provided the queue is not empty. While some links, such as wireless links, have variable capacity, we only consider fixed data-rate links that send a packet every C^{-1} seconds.

We will care about the utilization of these links.

Definition. *For a link with departure rate $D(t)$ and capacity C, the link utilization is $\mu(t) = D(t)/C$*

A work-conserving queue will be fully utilized at time t if and only if $Q(t) > 0$. While it may seem counter-intuitive, dropping packets does not necessarily reduce link utilization. If packets are dropped from the tail of the queue (i.e. an arriving packet doesn't fit, so it is dropped), then packets are lost *before* the bottleneck link. If a work-conserving link always has a non-empty queue, it is fully utilized no matter how many packets are dropped.

Definition. *The bandwidth-delay product of a flow crossing a link of capacity C with round-trip time RTT is* $\text{BDP} = C \cdot RTT$.

We will use the following standard assumption about how the size of the queue relates to the number of in-flight packets [8, 47, 121, 7, 188].

Assumption 2. *Let B be the maximum buffer size, t be some time, and $W(t)$ be the number of in-flight packets at time t. We assume that at time t, the queue size is*

$$Q(t) = \begin{cases} 0 & \text{if} \quad W(t) \leq \text{BDP} \\ W(t) - \text{BDP} & \text{if} \quad 0 < W(t) < B + \text{BDP} \\ B & \text{if} \quad W(t) \geq B + \text{BDP} \end{cases}$$

Our results use this assumption heavily, and so it merits some discussion. It is a strong assumption, designed to let us compare our results to existing work. Consider the queue at some time t when there are $W(t)$ packets in-flight. Over the previous RTT, if the bottleneck link is fully-utilized it will (by definition of BDP) send exactly one BDP worth of packets spaced C^{-1} seconds apart. Our assumption is essentially that the congestion control algorithm will closely match this behavior; i.e. it will also send packets spaced C^{-1} seconds apart. In the case of TCP, this is the ACK clocking mechanism; packets sent from the bottleneck queue lead to a stream of ACKs at the sender, spaced C^{-1} seconds apart, which pace out the sent packets.

Assumption 2 holds for the results in this paper because of ACK clocking, and is validated by our experimental results. If we were to model new congestion control algorithms, we would need to first check that this assumption holds. If, to pick an extreme example, a new congestion control algorithm required the sender to remain silent for a long time and then suddenly send a burst of packets, then the assumption would not hold. Fortunately, this is not how currently deployed congestion control algorithms work.

We could relax this assumption if needed, provided we kept some relationship between queue length $Q(t)$ and BDP. As an example, we observed some amount of variability in the queue depth in our measurements (e.g. Figure 4.1), which we believe is due to packet bursts from the sender. We could model this variability by assuming that $(1-\varepsilon)(W(t) - \text{BDP}) \leq Q(t) \leq (1+\varepsilon)(W(t) - \text{BDP})$ for some constant $\varepsilon > 0$. We could then derive similar results. For instance, the $B \geq \text{BDP}$ rule for TCP Reno would become $B \geq (1+\varepsilon)\text{BDP}$. We have not carried this parameter through our results since it makes our theorems harder to understand and compare with existing work, and $\varepsilon = 0$ captures the general behavior we observe in our measurements.

With this assumption, we can prove the following well-known lemma which is key to showing all the buffer sizing results in this paper.

Lemma 3. *For TCP flows sharing a link at time t, the utilization $\mu(t) = \min(W(t)/\text{BDP}, 1)$.*

Proof. If $W(t) > \text{BDP}$, then applying Assumption 2 we get $Q(t) = \min(W(t) - \text{BDP}, B) > 0$. We assume queues are work conserving, so $\mu(t) = 1$. If $W(t) \leq \text{BDP}$ then during RTT t we send $W(t)$ packets at a rate of $x(t) = W(t)/\text{RTT}$. By Assumption 2, $Q(t) = 0$, so the departure rate $D(t) = x(t)$. Therefore the utilization is $\mu(t) = D(t)/C = W(t)/\text{BDP}$. □

Lemma 3 is simultaneously obvious and a bit surprising. In words, if the number of in-flight packets *ever* falls below a BDP, then we will lose utilization. One might imagine that if the queue is full and the arrival rate is slightly less than C (which happens if $W(t)$ is slightly less than a BDP), the queue would only drain a little bit. But by Assumption 2 (and perhaps because of the way TCP's in-flight mechanism works), this means the sender will only send at a rate of at most $W(t)/RTT$ packets, which is slightly less than C.

Finally, we will need the following assumption about when losses happen.

Assumption 4 (Loss)**.** *Let B be the maximum buffer size. We assume that TCP loses packets at time t if and only if $Q(t) \geq B$, which by Assumption 2 is equivalent to $W(t) \geq \text{BDP} + B$.*

Lemma 3 is easiest to use when a congestion control algorithm decreases its window in response to a full queue (e.g. a lost or marked packet), since Assumption 4 gives us a direct relationship between the buffer size and the number of in-flight packets upon a loss (or mark). In our experiments, Reno, Cubic, BBR, and Scalable TCP all decrease their windows in response to a loss or mark. Modeling other congestion control algorithms might require additional assumptions to determine a lower bound on the number of in-flight packets.

4.3.4 Settings Where Our Assumptions Do Not Hold

Our results only apply when our assumptions hold. These assumptions are used by existing work on buffer sizing, so we are confident that our results are comparable to existing work.

Our results rely on the assumption that if the bottleneck queue is always non-empty, it will send exactly BDP packets every RTT. This is a simple consequence of a work-conserving queue and a constant egress line-rate. If instead we wanted to model a packet scheduling algorithm at the bottleneck link (e.g. strict precedence service, where our flows are low precedence), then we would need to determine how many packets are sent when the queue is non-empty. We could substitute that value into Assumption 2 and most of our results would apply.

There are other interesting settings for buffer sizing, especially links using AQM (e.g. [65, 140, 121, 137]). We believe that our techniques would help prove buffer sizing results for AQM policies, given good models of how different AQM policies interact with TCP. We believe this would be a very interesting direction for future work.

4.3.5 Rules of thumb for Reno, Cubic, BBR, and Scalable TCP

Lemma 3 makes it easy to prove rules of thumb for buffer sizing. As a demonstration, here we use it to prove rules for TCP Reno and other common congestion control algorithms.

Our strategy will be to find the minimum window size and apply Lemma 3 to measure link utilization. Loss based algorithms only decrease their windows on a loss, so we will use Assumption 4 (or a similar assumption for BBR) to relate the link utilization to the buffer size.

Reno: Fix some time t when Reno experiences a loss and let s be the next RTT when $W(s) = W(t)/2$. By Assumption 4, $W(s) \geq \frac{1}{2}(\text{BDP} + B)$. So the utilization of the link is $\mu(s) \geq \min\left(\frac{1}{2}\left(1 + \frac{B}{\text{BDP}}\right), 1\right)$.

If we would like the link to be fully utilized (i.e. $\mu(s) = 1$), this result requires that $B \geq \text{BDP}$. For smaller buffers, every 1% reduction in utilization results in a 2% reduction in required buffer size. So for 90% utilization, we only need $B \geq 0.8\text{BDP}$. Provided our assumptions were to hold without a buffer, Reno would still have a link utilization of at least $C/2$.

This result was shown for full utilization by [98], and for full utilization using these assumptions by [47]. We are unaware of prior results for less than full utilization.

Decrease by β on a loss: Suppose a new congestion control algorithm sets its window to $W' = \beta W$ on a loss, and never decreased its congestion window otherwise. Let t be the time just before a loss, and s the time when $W(s) = \beta W(t)$. In this case, we would have $W(t) \geq \text{BDP} + B$, and by Lemma 3, $\mu(s) \geq \min\left(\beta\left(1 + B/\text{BDP}\right), 1\right)$.

For full link utilization $\mu(s) = 1$, we would need $B > (\beta^{-1} - 1)\text{BDP}$. This matches observations made by [83, 121, 130].

Cubic: On a loss, Cubic decreases its window by constant factor "beta_cubic," in Linux this constant is $717/1024 \sim 7/10$. Using the above result, Cubic requires $B \geq (10\gamma/7 - 1)\text{BDP}$. For full link utilization $\gamma = 1$, Cubic requires a buffer of size $\frac{3}{7}\text{BDP}$. This result was shown by [121]. For 90% link utilization, Cubic requires a smaller buffer of size 0.28BDP. Without a buffer, Cubic gets a utilization of 70%.

Scalable: On a loss, Scalable TCP decreases its window to 7/8. It therefore requires a buffer size of $B \geq (8/7\gamma - 1)\text{BDP}$ For full link utilization, this requires $B \geq \frac{1}{7}\text{BDP}$. For 90% utilization, it requires a very small buffer of just $B \geq 0.03\text{BDP}$. Without a buffer, Scalable gets a utilization of 87.5%. We do not know of any prior publications of this result, but it is an easy corollary.

BBR: BBR (v1) alternates between phases where it probes for bandwidth and probes for the minimum RTT. We will focus on the required buffer size for full utilization during the bandwidth probing phase. BBR may behave differently in other settings, but this model characterizes the behavior we see in our experiments.

While probing for bandwidth, BBR cycles through a series of pacing rates which limit the number of packets in flight. It picks a pacing rate of R and sends at $\frac{5}{4}R$ for an RTT, $\frac{3}{4}R$ for an RTT, and R for six RTTs. In our experiments, BBR encounters loss during the $\frac{5}{4}R$ phase. After seeing loss,

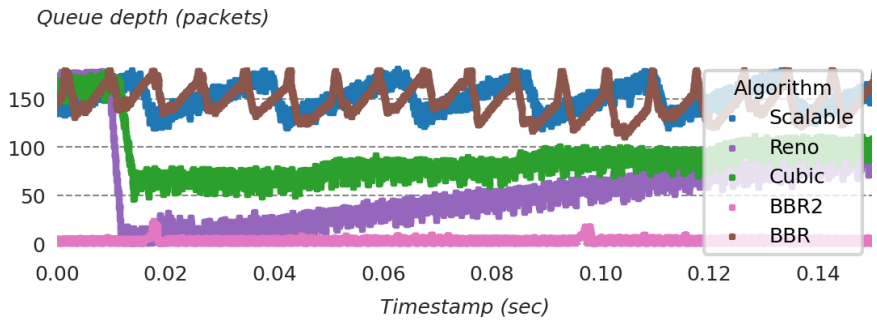


Figure 4.2: Queue occupancy versus time for one Reno, Cubic, BBR, and Scalable TCP flow with B = BDP. The buffer is (just) large enough for Reno to keep the queue non-empty and the link fully utilized. The buffer is oversized for Cubic, BBR, and Scalable TCP which keeps the queue persistently occupied.

it decreases its rate to $\frac{3}{4}R$ in response [31, Section 4.3.4].

Suppose BBR picks a pacing rate of $R \sim C$. When a loss occurs, the number of packets in-flight is at least BDP + B by Assumption 4. During the next RTT, the bottleneck queue drains at a rate of $C - \frac{3}{4}C = \frac{1}{4}C$. After one RTT, the number of packets in-flight decreases by $\frac{1}{4}C \cdot RTT$ so $W(s) = \frac{3}{4}BDP + B$. By Lemma 3, $\mu(s) = \min\left(\frac{3}{4} + B/\text{BDP}, 1\right)$. For full link utilization, we need $B \geq \frac{1}{4}$BDP. For 90% link utilization, we need a buffer of size $B \geq 0.15$BDP.

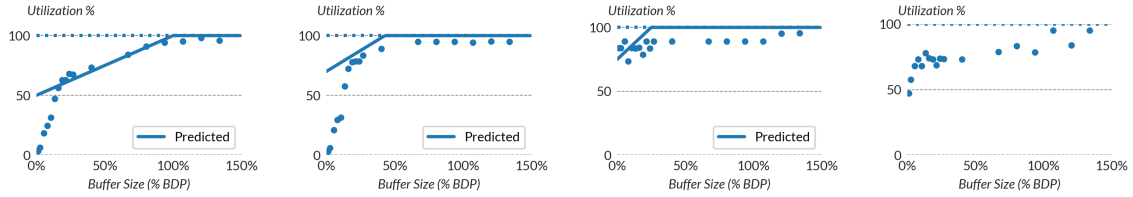
We do not know of prior publications of this result. Recently there has been work showing that BBR has higher loss and is unfair to other algorithms in smaller buffers [158, 32, 30, 93, 190]. Our results instead focus only on the buffer size needed to keep the link fully utilized.

BBRv2: Google is developing a second version of BBR, and recently released an alpha implementation. We want to apply our analysis to BBRv2 as soon as the algorithm is described in sufficient detail for us to model it correctly (we assume a technical report or paper will describe the alpha code). We do, however, include its behavior in our experimental setup using the alpha code. What we know so far is that instead of keeping the queue full and responding to losses (like all other algorithms we tested), BBRv2 keeps the queue close to empty and responds quickly to increases in delay. One consequence of this is that BBRv2 keeps the link highly utilized, but not quite 100%.

Hence, buffer sizing for BBRv2 is very different than for a loss-based algorithm, and BBRv2 is able to get good performance across a range of buffer sizes, while keeping the link utilization a little shy of 100%. This motivates us, later in this paper, to study buffer sizes that achieve high link utilization, but not quite 100%.

4.3.6 Experimental validation

Measurements from our physical network confirm the buffer sizing rules for TCP Reno, Cubic, BBR, and Scalable TCP in Section 4.3.5. For example, Figure 4.2 shows the queue occupancy, at the moment the window is decreased. As expected, if we set the buffer size to a BDP of 165 packets,



(a) Reno **(b)** Cubic **(c)** BBR **(d)** BBRv2

Figure 4.3: Link utilization for one TCP flow across a range of congestion control algorithms and buffer sizes.

TCP Reno allows the queue to drain by a full BDP and (almost) go empty. TCP Cubic allows the queue to drain by about 60-70%. BBR allows the queue to drain by about 25%. Scalable TCP drains the queue to within the range 11-25% of the BDP. The ranges we observe are consistent with the theory in Section 4.3.5.

The queue depth variability in our measurements is because the sending kernel transmits bursts. Except for BBR, which uses its own pacing algorithm to reduce burstiness.

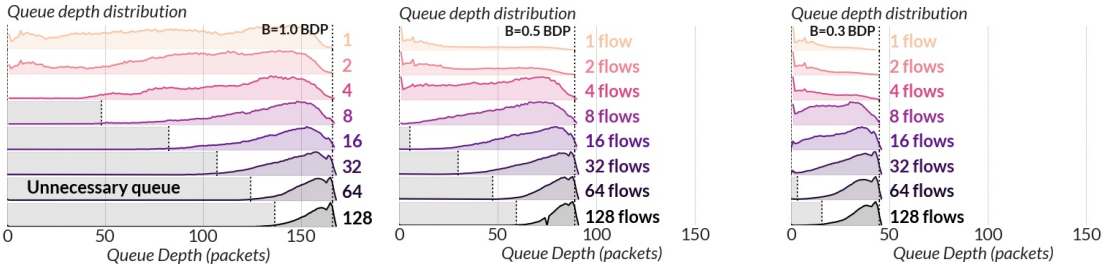
We measure link utilization in our experiments by recording `iperf's` aggregate throughput every 10ms, and report the 1st percentile value. Figure 4.3 shows the link utilization as a function of buffer size. Utilization is well-predicted by our models above a queue depth of about twenty packets. Below twenty packets utilization falls off quickly, which appears to be caused by burstiness (bursts cause the queue size to vary by about twenty packets); i.e., burstiness can cause loss with a smaller window than Assumption 4. Note that BBR, which has very little burstiness, has no utilization "cliff" below 20 packets.

4.4 Sizing buffers for multiple flows

In [8], Appenzeller et al. argue that when n TCP Reno flows share a connection, and n is large, a much smaller buffer of $BDP/\sqrt{n}$ is sufficient to keep the bottleneck link highly utilized. In particular, they prove the following theorem.

Theorem 5 (Square root of n rule)**.** *If for all times t, the windows of n TCP Reno flows are independent uniform random variables in the range $\left(1 \pm \frac{1}{3}\right) \frac{\text{BDP}+B}{n}$, and if $B \geq \text{BDP}/\sqrt{n}$, then in the limit as $n \to \infty$, $\mathbb{P}\left(\mu(t) < 1\right) \leq 0.02$.*

Appenzeller et al. show experimental evidence that the square root of n rule holds for TCP Reno, and the result has held up in later experiments [20]. However, Theorem 5 makes the strong assumptions that: (1) flows are TCP Reno, and (2) windows are independent uniform random variables in the range $[\frac{2}{3}\frac{BDP+B}{n}, \frac{4}{3}\frac{BDP+B}{n}]$. The rule does not apply to modern congestion control algorithms, and because of the strong assumptions it is not obvious whether a new or modified



(a) Buffer size: 1BDP. **(b)** Buffer size: 0.5BDP. **(c)** Buffer size: 0.25BDP.

Figure 4.4: Queue depth distributions for TCP Reno with decreasing buffer sizes. As the number of flows increases, the queue stays close to its maximum value (the right-most dotted line). The square root of n rule predicts that queues will not fall below a certain threshold, which is shaded in grey. The non-shaded region between them shows the required buffer size for our experiments: they are a good estimate, and hold as buffer sizes decrease.

algorithm will satisfy it.

In this section, we show that the square root of n result holds more broadly than for TCP Reno. We start, in Section 4.4.1, by showing that with much weaker assumptions, the aggregate window stays high when n TCP Reno flows share a link. This is depicted visually in Figure 4.4, showing how the queue depth distribution "concentrates" with increasing n. The square root of n result for TCP Reno (and other multiplicative-decrease algorithms) is an immediate consequence of this, which we show in Section 4.4.2. Basically, if the aggregate window stays above $\text{BDP} + B - \text{BDP}/\sqrt{n}$, and $B > BDP/\sqrt{n}$, then the queue never goes empty. More broadly, we will use this effect to analyze other congestion control algorithms.

We also prove results in Section 4.4.3 for the required buffer size when link utilization is *below* 100%. In particular, we show that utilization is $\Omega(1 - 1/\sqrt{n})$ even with a constant size buffer. As a perhaps surprising example, if more than 1,000 flows share a link, utilization is at least 97% even with a constant size buffer. For the same link utilization, the square root of n result of [20] requires a buffer of size $\text{BDP}/\sqrt{n}$.

Finally, in Section 4.4.4 we extend our results to other congestion control algorithms. We show that because BBR incorporates some randomness by design, we are able to prove a square root of n result with *only* an assumption about fairness. Using these ideas, we propose a modification for multiplicative decrease algorithms that provably guarantees that a small buffer is sufficient, provided fairness holds.

4.4.1 How Reno keeps links and queues full

TCP Reno tends to keep links and buffers full, especially as more flows use a network. Figure 4.4 shows the distribution of queue depths in our experiments. As the number of flows increases, TCP Reno keeps the queue depth distribution high. In this section, we will prove this.

In order to prove some sort of square root of n rule, we will need some concept of how the aggregate window is split across multiple flows. If windows are extremely imbalanced, we cannot hope for a square root of n rule. For instance, if the aggregate window is made up of only one flow, we are in the setting of the single flow case in Section 4.3 and cannot expect a square root of n rule. Furthermore, in our experiments in Section 4.5, we find that the required buffer size gradually increases as flows become less fair.

To deal with this dependence on fairness in our theoretical results, we will define a concept of *almost* fairness for TCP windows. Almost fairness has a parameter Δ, which relates to how far the window sizes are from equal. We will prove our results using this parameter, which gives us a convenient relationship between unfairness and buffer size.

Definition (Δ-almost fair)**.** *Consider n TCP flows sharing a link. Let $w_i(t)$ be the number of in-flight packets for flow i at time t. Let $\Delta \geq 1$. Flow i is Δ-almost fair if for all time t,*

$$\Delta^{-1}\frac{\text{BDP}}{n} \leq w_i(t) \leq \Delta\frac{\text{BDP}}{n}. \tag{4.1}$$

Δ-almost fairness is a property of a congestion control algorithm. The closer an algorithm keeps its windows to equal, the smaller value of Δ it will have. This may or may not be *desirable* behavior for a congestion control algorithm, but we will be able to prove smaller buffer requirements for smaller values of Δ. We can easily measure and calibrate this parameter in our experiments, and we report the results of this in Section 4.5.6. We are unaware of prior uses of this fairness metric. We could use other ways of measuring fairness, for instance Jain's fairness index [99], but Δ-almost fairness is convenient for our proofs and easy to measure in our experiments.

It may feel somewhat unnatural that we have not defined Δ-almost fairness as $w_i(t) \leq \Delta(\text{BDP}+B)/n$, since the aggregate window is at most $\text{BDP}+B$ and is divided among n flows. However, we are thinking about the regime where $B = O(\text{BDP})$, and can account for the dependence on B by increasing the value of Δ in our definition. Doing so makes our results significantly easier to prove and interpret, and the cost of a slightly looser bound on buffer size.

With this definition, we can prove our main result which relates the aggregate window after a loss to buffer size, fairness, and the number of flows. We will spend the rest of the section discussing its interpretation.

Theorem 6. *Consider n Δ-almost fair TCP Reno flows sharing a link with window size $w_i(t) \geq 2$. Fix two times $0 \leq t_1 \leq t_2$. If at most $\frac{n^2}{\Delta\text{BDP}} + \sqrt{n}$ flows decrease their windows between t_1 and t_2, then for all $t_1 \leq t \leq t_2$*

$$W(t) \geq \text{BDP} + B - \frac{\Delta\text{BDP}}{\sqrt{n}}.$$

Proof. See Appendix 4.11. □

We have stated Theorem 6 for TCP Reno to make the theorem statements more direct, but as in

Section 4.3, it is easy to adapt to other congestion control algorithms. Suppose a congestion control algorithm decreases by βw_i on a loss and otherwise increases by at least α. We would need to add a condition to Theorem 6 that the smallest window $w_i \geq \alpha/(1-\beta)$. This affects Equation (4.4) in the proof, but the overall result does not change.

4.4.2 Buffer size required for full link utilization

With Theorem 6, we can easily find the minimum buffer size required for full link utilization.

Theorem 7. *Consider n Δ-almost fair TCP Reno flows sharing a link with window size $w_i(t) \geq 2$. Fix two times $0 \leq t_1 \leq t_2$. If at most $\frac{n^2}{\Delta\text{BDP}} + \sqrt{n}$ flows decrease their windows between t_1 and t_2, and*

$$B \geq \frac{\Delta\text{BDP}}{\sqrt{n}}$$

then $\mu(t) \geq 1$ for all $t_1 \leq t \leq t_2$.

The key condition in Theorem 6 is that at most $\frac{n^2}{\Delta\text{BDP}} + \sqrt{n}$ decrease their in-flight packets. This expression is unusual, and deserves some explanation.

The first term $\frac{n^2}{\Delta\text{BDP}}$ is related to the number of fair flows which must decrease windows for the aggregate window to decrease. Suppose only one TCP Reno flow with a fair window size of BDP/n sees a loss and halves its window. If $\text{BDP}/n < n-1$, then the remaining flows will each increase their windows by one and the aggregate window will *increase* despite the loss. Appendix 4.13 formalizes this intuition.

The second term $\sqrt{n}$ is the number of *additional* flows which decrease their in-flight packets over and above this minimum, and measures the amount of synchronization. In Section 4.5, we show that this is the number of additional flows which decrease their windows in our experiments. Our results also gracefully degrade as the amount of synchronization increases. We will discuss these and more modifications to Theorem 6 in Section 4.4.6.

Theorem 7 helps us determine the correct value of n to use when sizing a buffer. Prior work has noted that Theorem 5 assumes all flows are active and contributing to the buffer size, whereas in practice they might not [130, 165, 187], making it difficult to size the buffer correctly. Consider, for example, an application that starts and stops every second; should its flows be counted, or only if it recently had packets in the buffer? Theorem 7 uses the number of flows which see packet loss during an RTT, allowing us to size the buffer accordingly.

4.4.3 Link utilization when n TCP Reno flows share a link

Prior buffer sizing work has looked for the smallest buffer required for full link utilization. Using Theorem 6, we can also understand what happens if the buffer becomes even smaller.

Theorem 8. *Consider n Δ-almost fair TCP Reno flows sharing a link with window size $w_i(t) \geq 2$. Fix two times $0 \leq t_1 \leq t_2$. If at most $\frac{n^2}{\Delta \text{BDP}} + \sqrt{n}$ flows decrease their windows between t_1 and t_2, then for all $t_1 \leq t \leq t_2$ and $B \geq 0$*

$$\mu(t) \geq 1 - \frac{\Delta}{\sqrt{n}}.$$

This suggests that even with very small buffers, as n grows TCP Reno approaches full link utilization quite quickly, at a rate of $O(1/\sqrt{n})$. For instance if $n = 10,000$ and $\Delta = 2$, then the link will always be at least 98% utilized—independent of buffer size.

This is a worst-case bound on instantaneous utilization, not on average utilization. This bounds the minimum utilization immediately after a loss. As TCP increases its window after a loss, utilization will increase and the longer term utilization will be higher.

Intuitively, the square root of n result says that TCP Reno will not decrease the aggregate window too far on a loss. If the buffer is fairly large, this means the buffer will tend to remain very full. If the buffer is very small, this means that link utilization will remain high.

4.4.4 BBR guarantees a square root of n rule

Our square root of n results so far require assumptions about how TCP flows behave. For example, Theorem 6, assumes only a limited number of flows decrease their windows in each RTT.

We can also prove a square root of n rule for BBR (v1), based on how it probes for available bandwidth. If BBR detects a loss, while probing for bandwidth, it *randomly* decides whether or not a flow should decrease its window, which in turn desynchronizes the flows sufficiently for the square root of n rule to hold. Our proof requires no assumption on the number of flows seeing a loss at the same time. Essentially by incorporating some randomness, BBR is able to *guarantee* a square root of n result with minimal assumptions.

Theorem 9. *Consider n Δ-almost fair BBR flows in the "probe bandwidth" phase sharing a link. Fix two times $0 \leq t_1 \leq t_2$. If*

$$B > \frac{\Delta \text{BDP}}{\sqrt{2n}} \sqrt{\ln 1/\delta}, \tag{4.2}$$

then $\mathbb{P}(\mu(s) < 1) \leq \delta$ for all $t_1 \leq t \leq t_2$.

Proof. See Appendix 4.12. □

We believe Theorem 9 could be much stronger. Figure 4.6b shows that BBR's queue depths are much more tightly concentrated than for other algorithms, and more tightly concentrated than Theorem 9. While the number of flows experiencing a loss may be about the same as with TCP Reno, fewer of them decrease their windows in BBR. We see no reason why a buffer size of $O(\text{BDP}/n)$ would not be more appropriate for BBR, but have not been able to prove or disprove it.

Finally, there are still many open questions about BBR's buffer behavior for future work. We have only considered BBRv1. We show BBRv2's experimental behavior in Figure 4.6c, but have

not analyzed it. We do not yet have a rule for BBR in the case when it decreases its window size when there is no packet loss. The main obstacle is that we do not observe this behavior in our experimental setup.

4.4.5 Guaranteeing a square root of n rule for a modified TCP Reno

Inspired by BBR, imagine that we added randomness to TCP Reno to reduce its buffer requirements (and increases its link utilization with small buffers). We are not arguing that TCP Reno should necessarily be changed; we are conducting a thought experiment to see how much it would reduce buffer requirements.

When our imagined algorithm detects a loss, it halves its window size $w(t)$ with probability $p = 1/w(t)$. This could be done either by randomly ignoring lost packets, or by randomly marking packets with a suitable ECN policy. Let's assume that this does not interfere with Δ-almost fairness, so that $\Delta^{-1}\text{BDP}/n \leq w(t) \leq \Delta\text{BDP}/n$, and $p \leq 1/(\Delta^{-1}\text{BDP}/n)$. The number of flows that randomly decide to decrease their windows in the same RTT is a sum of independent random variables, and we can use the following simple Chernoff bound to estimate the sum.

Lemma 10. *Let $D_i(t)$ be independent and identically distributed Bernoulli random variables, with $\mathbb{E}\, D_i = p \leq \frac{1}{\Delta^{-1}\text{BDP}/n}$. Let $D(t) = \sum_{i=1}^{n} D_i(t)$ Then*

$$\mathbb{P}\left(D(t) > \frac{n^2}{\Delta^{-1}\text{BDP}} + \sqrt{n}\right) \leq \exp(-1/2).$$

Proof. See Appendix 4.14 □

This bound essentially meets the conditions for Theorem 6, with the small caveat that we would need to adapt the proof to using Δ^{-1} instead of Δ. Doing so would give us a square root of n result without needing to make assumptions about which flows see a loss.

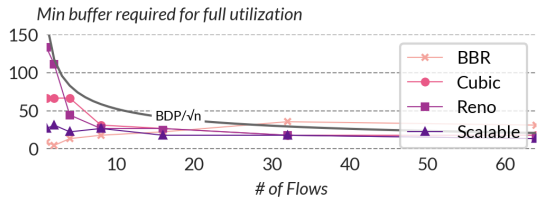
4.4.6 Desynchronized flows reduce buffer requirements

Before Theorem 6, we knew the required buffer size for two extreme cases. When all the flows are *perfectly synchronized*, they lose packets simultaneously and we need a BDP of buffering. At the other extreme, when flows are *not synchronized*, Theorem 5 tells us that the buffer size can be decreased by a factor of $\sqrt{n}$.

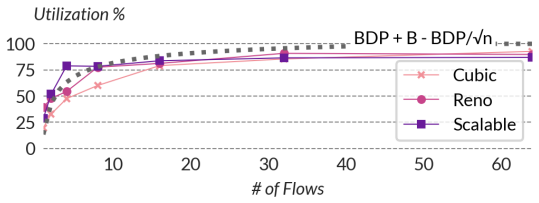
We can extend Theorem 6 to the intermediate case when $\frac{n^2}{\Delta\text{BDP}} + s$ flows see a loss. In this case

$$W(t) \geq \text{BDP} + B - s\frac{\Delta\text{BDP}}{n}. \tag{4.3}$$

In other words, Equation 4.3 tells us:



(a) Buffer size required for full link utilization across all our experiments.



(b) Utilization as a function of number of flows for a 20 packet buffer.

Figure 4.5: Buffer size and link utilizations behave according to a square root of n rule in our experiments

- If flows are slightly but not *perfectly* synchronized, then the buffer size can be made smaller than BDP. For some small constant c, if only $s = cn$ extra flows lose packets, then the required buffer size is cBDP which is below a BDP.
- If only a constant number of additional flows s see each loss, then a buffer of size $O(\text{BDP}/n)$ is possible. This is tantalizing, because, of course $\text{BDP}/n < \text{BDP}/\sqrt{n}$. For example, if 10,000 flows share a link, and $s = 1$, then we can further reduce the square root of n buffer requirement by another 100-fold. To be clear, our experiments do not exhibit this degree of desynchronization, but we see it as an exciting opportunity for a new congestion control algorithm.

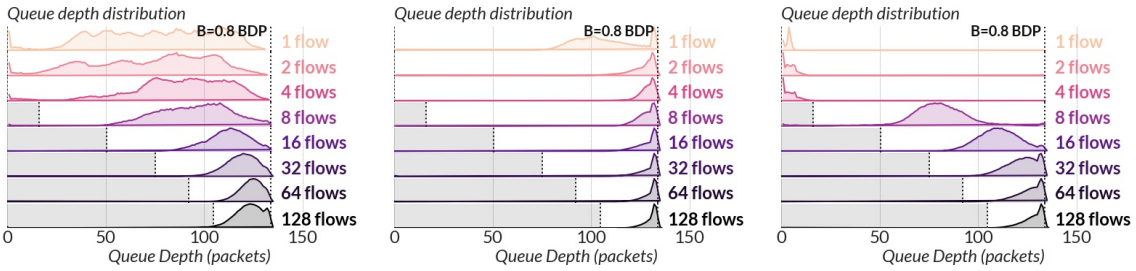
4.5 Experiments with the square root of n rule

We evaluated Theorem 6 in our physical network, using our per-packet measurement infrastructure described in Section 4.2. Our experiments show that Theorem 6 holds in our network, and build intuition on which factors impact buffer sizes.

4.5.1 The square root of n rule holds when algorithms respond to full queues.

The square root of n rule holds in our experiments for multiple congestion control algorithms (TCP Reno, Cubic, Scalable TCP, and BBR), despite the introduction of PRR. Figure 4.5a shows the minimum queue depth required for full link utilization in our experiments. Each experiment lasted one thousand RTTs. We measured minimum utilization by recording `iperf's` aggregate throughput over ten consecutive RTTs. We report the 1st percentile throughput.

In each experiment we found the smallest buffer size for which 99% of packets (or more) experienced a non-empty queue. The buffer requirement follows a square-root of n rule for TCP Reno. Other congestion control algorithms have similar buffer requirements for a large numbers



(a) Cubic **(b)** BBR **(c)** BBRv2

Figure 4.6: Queue depth distribution for various congestion control algorithms and buffer size 0.8BDP. The non-shaded area corresponds to a buffer size of $2\text{BDP}/\sqrt{n}$, which tends to overestimate the required buffer.

of flows (and smaller buffer requirements for smaller numbers of flows). Theorem 7 requires that $B \geq \Delta\text{BDP}/\sqrt{n}$, and our experimental results suggest it is an overestimate.

Theorem 8 predicts that utilization will grow at a rate of $1/\sqrt{n}$ when the buffer is too small for full link utilization. Figure 4.5b shows the utilization as a function of the number of flows for a 20 packet buffer. Utilization increases at a rate of $1/\sqrt{n}$. It is slightly underpredicted by Theorem 8, and again we believe that the constants may be slightly looser than necessary.

4.5.2 Algorithms keep queues full, as predicted by the square root of n rule.

The square root of n rule is primarily used to predict the minimum required buffer to keep the link utilized, as in Theorem 7. But Theorem 5 also predicts how far the queue occupancy will deviate from full if the buffer is larger than we need. Figure 4.4 shows the distribution of queue occupancy for different numbers of flows (and different buffer sizes). For example, with $B = \text{BDP}$ in Figure 4.4(a), the distribution of queue occupancy narrows as the number of flows increase. The shaded area to the left of the distribution is unused (and therefore unnecessary) buffer; we could use a smaller buffer and shift the distribution to the left. The remaining graphs show this as we decrease the buffer size.

We have seen that non-Reno congestion control algorithms also follow a square root of n rule in our experiments, and Figure 4.6 shows the distribution of queue depths for these algorithms. As with Reno, the distributions concentrate tightly around a full buffer.

4.5.3 The square root of n rule holds with non-uniform window distributions.

Figure 4.7 shows the distribution of windows for 16 TCP Reno flows over one second, when $B =$ BDP. The distributions are clearly not uniform (and we observe windows in the ranges of 0-10 and

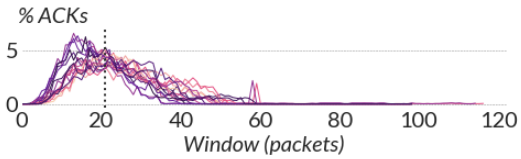


Figure 4.7: Distribution of window sizes measured for each received acknowledgement for 16 TCP Reno flows over one thousand RTTs with a BDP sized buffer.

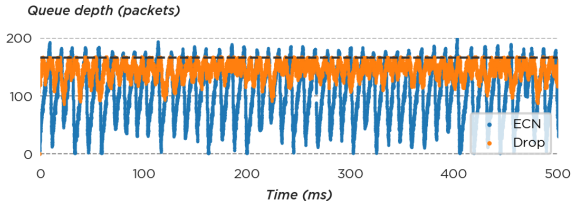


Figure 4.8: Queue occupancy over time for 16 TCP Reno flows sharing a queue. Packets are dropped or marked when the queue exceeds one BDP. In the ECN trace, more flows are marked when the queue is exceeded and flows become more synchronized.

40-60 packets, outside the $\frac{3}{4}$ and $\frac{5}{4}$ range). This suggests that the uniform window size assumption in [8] does not hold in practice.

Figure 4.7 also shows the degree to which flows are treated fairly, in the sense that the the average window size for each flow is close to $(\text{BDP} + B)/n$. It is not surprising that they are similar, since one of the goals of a congestion control algorithm is to fairly share a bottleneck link. However, we can see that the flows do not have identical window size distributions, with some average window sizes two to three times larger than the fair share of $(\text{BDP} + B)/n$, and some half the size. This is why we have defined Δ-almost fairness.

4.5.4 The square root of n rule does not hold when losses are synchronized.

Theorem 6 requires flows to be desynchronized, in that not too many flows can respond to loss at the same time. This is a necessary requirement and we would not expect the square root of n rule to hold when flows are synchronized. In the extreme, suppose all flows experience a loss between times t_1 and t_2. In the worst case $W(t_2) = \sum_{i=1}^{n} \frac{1}{2} w_i(t_1) = \frac{1}{2} W(t_1)$. As pointed out in prior work [8, 47], we are back in the same situation as for one flow and require a BDP of buffering.

While larger numbers of TCP Reno flows tend to be less synchronized in our experiments, this is not the case for all algorithms or in all settings. As a stark illustration, we describe an experiment in which synchronization caused a larger buffer requirement. Our experiment compared dropping packets to a specific way of ECN marking: we dropped or marked packets whenever the queue depth exceeded a fixed threshold.

We ran this experiment for sixteen TCP Reno flows. Figure 4.8 shows the queue depth over time during this experiment, comparing TCP Reno with ECN (blue line) and with dropped packets (orange line). It is clear that with ECN marking, the queue depth fluctuates much more, hence requiring a much larger buffer. With ECN the queue regularly overshot its marking threshold, so more packets were marked in the same RTT, which led to a synchronized reduction in window size.

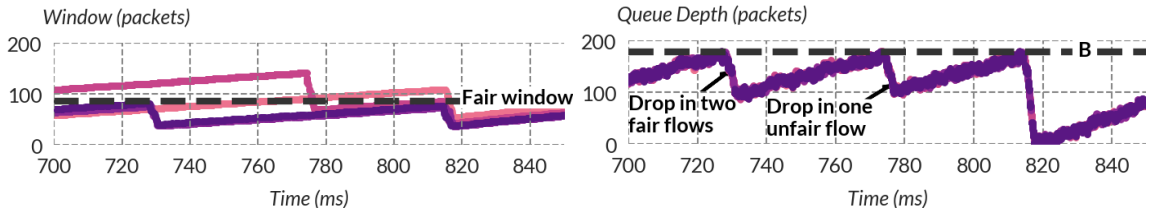


Figure 4.9: Windows and queue depths over time for four TCP Reno flows sharing a link. A loss for a flow with an unfairly large window results in the same queue depth decrease as for two fair flows, and half as large a decrease as a loss for all flows.

We would like to emphasize that this experiment should *not* be taken as a general statement about buffer sizing with ECN, since ECN is often used in conjunction with an AQM policy (e.g. RED [65]). However, ECN is used in this way for BBRv2 [35], DCTCP [7], and in other settings [174, 146].

4.5.5 The square root of n result does not hold when flows are too unfair.

Theorem 6 requires windows to be almost fair, and this is a necessary requirement. Figure 4.9 illustrates how unfairness between flows can affect the required buffer size. Consider the trace of four TCP Reno flows sharing a link, particularly the large red window and the two smaller purple windows (superimposed). When the small purple windows experience losses simultaneously at 725ms, both halve their windows from 80 to 40 packets and the buffer occupancy drops by 65 packets. Yet, on its own, the larger red window causes the buffer to drop by the same amount when it halves its window size – this is because it starts from a larger size. In general, bigger windows have a disproportionately larger effect on the buffer occupancy. In the extreme, imagine one flow used *all* of the link capacity, and the remaining $n-1$ flows used none. Such extreme unfairness means we are back in the single flow setting, and we can not do better than the BDP rule.

4.5.6 Checking theorem conditions in our experiments.

Theorem 6 depends on the value of Δ-almost fairness, and the number of flows which see a loss in a RTT. These are not well known values, and so we use our experimental setup to measure them and give a sense of their magnitude.

First, we will check whether the number of additional flows which see a loss is about $\sqrt{n}$. This is a strong assumption in general, but it appears to be a loose upper bound in our experiments. We ran a number of experiments with a buffer size of 1 BDP and increasing numbers of TCP Reno flows. We split each experiment into RTTs, and for each RTT where there was one loss, counted the number of flows which saw a loss in each RTT. Figure 4.10 shows the average number of flows that observed a loss in that RTT. The bound $\frac{n^2}{2\text{BDP}} + \sqrt{n}$ of Theorem 6 is also plotted. It appears to be a good estimate for a small number of flows, and an *overestimate* for larger numbers of flows.

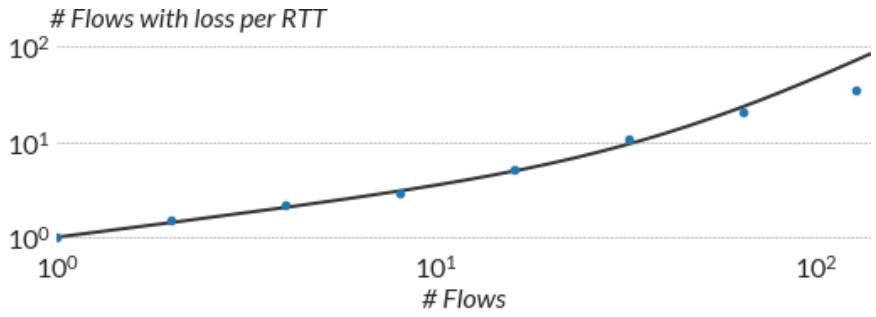
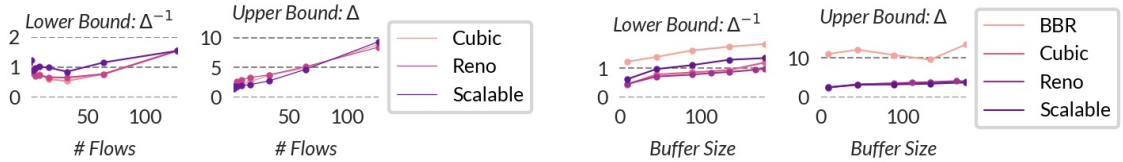


Figure 4.10: Fraction of TCP Reno flows which see a loss during loss events, for various numbers of simultaneous flows. Each point shows the mean number of flows seeing a loss during each RTT, and the line shows the bound of $\frac{n^2}{2\text{BDP}} + \sqrt{n}$ from Theorem 6.



(a) Varying numbers of flows

(b) Varying buffer size in packets

Figure 4.11: Worst-case Δ-almost fairness in our experiments.

Next, we check the values of Δ-almost fairness in our experiment. To measure Δ-almost fairness, we find the 1st and 99th percentile window sizes for all flows in an experiment. We calculate BDP/n from the experiment parameters. We then find the smallest value of Δ so that the definition of Δ-almost fair is satisfied between the 1st and 99th percentile window sizes.

We first look at how Δ depends on the number of flows. Figure 4.11a shows that both Δ^{-1} and Δ slowly increase with the number of flows. We next look at how changing the buffer size impacts fairness. We look at all experiments with sixteen flows, to avoid confounding with the dependence on the number of flows. We find that Δ^{-1} slowly decreases with buffer size, and approaches 0.5. For Cubic, Reno, and Scalable, Δ is about 4. For BBR, it is about 10. Δ does not depend on the buffer size for any of these algorithms.

4.6 Supporting evidence from real-world measurements

Square root of n results hold because the occupancy of router queues "concentrates" during times of congestion, leading to persistently occupied standing queues. This phenomenon has been widely observed outside of buffer sizing work. For example, Lee et al. [123] report that operating a link at 100% utilization resulted in a persistent 30-50ms increase in queueing delay. In [175], Täht and Reed report on an outage where they observed a 400ms increase in RTTs. In [38], Chandrasekaran

et al. report 20ms increases in RTTs over congested links in the internet core.

Recent measurement work uses these increases in RTT to measure congestion. Luckie et al. propose TSLP [124], which measures RTTs periodically and uses a persistent increase in RTT as an indicator of congestion. Their figures show that congestion can elevate minimum RTTs by tens of milliseconds. In [61], Fanou et al. use TSLP to measure congestion in the African IXP substrate, and find links with elevated minimum RTTs during congestion. Sundaresan et al. classify flows with a tightly concentrated RTT during slow-start as flows experiencing congestion [173]. In [45], Dhamdhere et al. measure the minimum RTT over a period of five and fifteen minutes to identify links experiencing persistent congestion.

In [166], Spang et al. report on buffer experiments run at Netflix. They show that large buffers can increase RTT for *all* traffic sharing a link. They run an experiment where a subset of Netflix traffic is randomly assigned to two routers, one with a 500MB buffer and one with a 50MB buffer. They observe that the minimum RTT increase by hundreds of milliseconds for all flows; and gets worse with larger buffer sizes. This suggests the queue remains relatively full for the entire hour.

Our results explain why congestion control algorithms that respond to full queues (via losses or marks) create standing queues during times of congestion. Our experiments in Section 4.5.2 show the same persistent standing queues, especially as the number of flows increases.

All of the measurements described above show evidence of concentration. While they do not prove that the concentration is of the order BDP/$\sqrt{n}$, our results suggest that all these buffers could be shrunk without impacting link utilization. Our results also suggest that link utilization would remain high in these cases, even with a very small buffer.

It is natural to ask whether concentration relies on current congestion control algorithms, and whether a new congestion control algorithm might break it. For instance, if a few large sources of traffic switched to BBR, would congestion on the internet look very different? Our results in Section 4.4, especially the ones for BBR, suggest it would not. But we cannot rule out some other, future congestion control algorithm with very small queues during congestion.

Finally, our results suggest intriguingly that there might be "dark congestion" in the internet, which current measurement techniques that look for persistent standing queues cannot detect. There might be routers that cause a large amount of synchronization, for instance by marking packets as in Section 4.5.4. This would lead to wildly varying queue occupancy, making it hard to detect congestion.

4.7 Recommendations

4.7.1 Recommendations for Operators

The networking community as a whole, both industry and academia, knows surprisingly little about buffer size requirements, and tends to oversize buffers in the WAN. We believe there might be big

benefits to network operators if they run experiments to determine the required buffer size in their network. Consider, for example, the difference in required values for a network with a BDP of $100ms \times 100Gb/s = 10^{10}$ bits carrying 10,000 TCP Reno flows. If we adopt the single flow rule, we need a 10Gbit buffer; if Theorem 6 applies we might need only 100Mbit buffers. BBR only needs a 100Mbit buffer, and BBRv2 might require even less. If we were happy with an instantaneous utilization higher than 99%, the buffer could be reduced a few tens of packets. This could make it possible to use simpler, cheaper routers with on-chip buffering.

We should point out that this paper focuses exclusively on the relationship between sizing a buffer and link utilization. Buffers can also have a large impact on application performance [47, 46, 87, 166, 21, 15], and it is important to consider these factors. More experiments would benefit operators (and the broader networking community) by improving our understanding of how buffers impact application performance.

Theorem 7 may be easy to verify in practice. If end hosts can be instrumented, one could measure the total decrease in windows over a short period to estimate the required buffer size. With fine-grained metrics from the switch, it would be possible to count the number of unique flows dropped during congestion events (e.g. by forwarding all lost packets to a collector), estimating the bandwidth of these flows (e.g. using coarse netflow statistics), and size the buffer accordingly. This could be done manually, or automated and be done periodically. If only loss statistics are available, it would be possible to estimate the number of flows which see simultaneous loss using simple probability calculations. Then Equation (4.3) could be used to size the buffer. Note that it is *not* necessary to measure the total number of flows to apply our results, which avoids the problems pointed out in [130, 165, 187].

4.7.2 Recommendations for Congestion Control Algorithm Designers

To someone working on congestion control, this paper may feel a bit backwards. As argued by [128], instead of sizing buffers based on unintended artifacts of TCP Reno (i.e. as in the single flow rule), we could instead design congestion control algorithms that work well for all queue sizes. If we want full link utilization and a small buffer, the congestion control algorithm only needs to prevent the aggregate window from dropping below a BDP.

In Section 4.3, we showed that smaller reductions in window size lead to smaller buffer requirements. In Section 4.4.4 and Section 4.4.5, we showed how adding a small amount of randomness to a congestion control algorithm can reduce its buffer requirements. We encourage congestion control algorithm designers to use these techniques to reduce buffer requirements. Our experimental results suggest that BBRv2 is a good first step in this direction. With a bit of work, new algorithms can be more friendly to buffers, and the people operating them.

4.8 Related Work

There have been many papers published about buffer sizing in the past twenty years. We will focus on work most closely related to our results, for broader surveys see [186, 130].

The rule of thumb for sizing router buffers with a single TCP Reno flow is attributed to [98, 185]. The rule was extended to multiplicative-decrease algorithms by [83, 121]. The proof of the BDP rule by [47] did not depend on silence after a loss. We believe our discussion of PRR and BBR are new.

For multiple flows [8] introduced the square root of n rule, which was tested experimentally by [20] in the Level 3 backbone. [47] described a rule incorporating a model of loss, and includes similar style of analysis as Theorem 6. [188] show that number of flows which see a loss is important for whether a square root of n rule applies.

TCP's tendency to observe loss simultaneously and synchronize has been observed in simulation since the 1980s [82, 196, 197, 198, 66, 103]. More recently, [121] observed synchronization using physical hardware. Although we only observe a small amount of synchronization in our experiments, our buffer size results can handle larger amounts. We think it is an interesting open question why it seemed to be more common in the 80s and 90s, and why it only sometimes appears today.

In the early 2010s, [69] revisited the question of buffer sizing, observing that the oversized buffers in cable modems cause massive delays and standing queues during congestion, often referred to as *bufferbloat*. Bufferbloat usually refers to home networks where a small numbers of flows typically share the buffer, while we are more focused on backbone routers with a large number of flows; but we view this work as complimentary. We discuss how our results are related to standing queues in Section 4.6, albeit for larger numbers of flows, and our results may shed light on other parts of the Internet where "Dark Buffers" lurk.

Our work focuses on the relationship between buffer size and link utilization. A separate, very interesting line of work has explored other impacts of buffer sizing. [47, 46] discuss impacts on TCP such as increased loss and throughput variability. A recent line of work [158, 32, 30, 93, 190] has shown that BBR can have large loss and worse fairness in small buffers. [87] describes a test-bed study showing that buffer sizing can cause a significant change in application QoE. [166] reports on production buffer sizing experiments at Netflix, and show that buffer size has a large impact on video performance. [21] reports on buffer experiments at Facebook, including impacts on flow completion time. [15] shows an example where shrinking the buffer size of a home WiFi router can degrade video performance.

4.9 Conclusion

The main takeaways from this paper are new results on buffer sizing; in particular, a better understanding of the relationship between link utilization and buffer size, and how congestion control algorithms can impact this relationship. Prior work suggested that buffers can be reduced by a factor

of square root n, offering dramatic buffer reductions in networks carrying many flows. However, the result required TCP Reno, did not make clear how we determine n, or what happens when windows are not independent. Our results clarify that n is the number of flows that reduce their window size in the same RTT, removes the need for independence, and holds for a broader class of congestion control algorithms. Our results explore what happens when buffers are sized too small for full link utilization. Our results make it easier to run buffer sizing experiments, which should give much more confidence that the results apply broadly.

There remains work to be done. For instance, we assumed that RTTs are the same for all flows, yet clearly this is not true in practice. Variance among RTTs should only reduce synchronization, and hence further reduce the buffer requirement, but we have not been able to prove it. We still do not understand the underlying causes of synchronization, or how to reduce it in drop-tail queues. We believe our BBR result can be improved, and that BBR may allow a very small buffer. We have focused on sizing the buffer for the worst-case behavior over a relatively long period of time, and it may be possible to dynamically adjust the buffer size on a much shorter timescale.

More generally, though, we still have only a rudimentary understanding of buffer size, despite its potentially big impact on application performance. Even if we know all the traffic using a link, we don't know how to predict the best buffer size. We encourage further experimentation and measurement with real applications.

Finally, we believe that future congestion control algorithms can significantly reduce buffer requirements. With small modifications, existing algorithms can reduce buffer requirements, or increase link utilization with small buffers. If packet arrivals can be managed perfectly, buffers can be made smaller or even eliminated all together. We see no reason why a future congestion control algorithm would need anything more than a very small buffer.

4.10 Acknowledgements

We would like to thank Vladimir Gurevich for all his incredible support around debugging P4. We would also like to thank Joseph Little for the invaluable IT support. This research was funded by Netflix. The opinions expressed in this article are the authors' own and do not reflect the view of Netflix.

4.11 Proof of Theorem 6

Proof. Fix some time t such that $t_1 \leq t \leq t_2$. Let D be the set of flows decreasing their windows by time t, let $w = W(t_1)$ and $w' = W(t)$. After D has decreased their windows, the aggregate window w' is

$$w' = w - \sum_{i \in D} \frac{1}{2} w_i(t_1) + \sum_{i \in \bar{D}} 1.$$

By a union bound,

$$w' \geq w - \frac{1}{2}|D| \max_{i \in D} w_i(t_1) + n - |D|.$$

Let $w^* = \frac{\Delta \text{BDP}}{n}$. By Δ-almost fairness, $\max_{i \in D} w_i(t_1) \leq w^*$. Substituting and simplifying we have,

$$w' \geq w + n - |D| \left(1 + w^*/2\right), \tag{4.4}$$

Since $w_i(t) \geq 2$, $w^* \geq 2$, and so $1 + w^*/2 \leq w^*$. Therefore,

$$w' \geq w + n - |D| w^*.$$

By the conditions of the theorem $|D| \leq \frac{n^2}{\Delta \text{BDP}} + \sqrt{n} = \frac{n + \sqrt{n} w^*}{w^*}$. Substituting, we have

$$w' \geq w - \sqrt{n} w^*.$$

Expanding the definition of w^*, this becomes

$$w' \geq w - \Delta \frac{\text{BDP}}{\sqrt{n}}.$$

By Assumption 4,

$$w' > \text{BDP} + B - \Delta \frac{\text{BDP}}{\sqrt{n}}.$$

□

4.12 Proof of Theorem 9

The proof of a square root of n rule for BBR takes a slightly different path from the usual AIMD proof. In particular, we don't need to assume that the number of flows which see a loss is limited.

During the probe bandwidth phase, BBR picks a pacing rate of R and sends at pacing rates $\{\frac{5}{4}R, \frac{3}{4}R, R, R, R, R, R, R\}$ for an RTT each. It begins this cycle at a random point (which is not $\frac{3}{4}R$).

When BBR experiences a loss, it only decreases its rate if it is sending at a pacing rate of $\frac{5}{4}R$ [31, Section 4.3.4]. In expectation, if flows were equally split between all the pacing rates, we would only expect about $n/8$ flows to decrease their rates by $1/4$. And even better, another $n/8$ flows would increase their rates by $1/4$ over the next RTT and the total change would be small.

Proof. In the worst case, *all* BBR flows see a loss at time t. Consider some flow i, and let $R_i(t)$ be the rate of flow i at time t, and $X_i(t) = R_i(t) \cdot \text{RTT} - R_i(t_1) \cdot \text{RTT}$. Because of the random order

of pacing rates, the $X_i(t)$ are independent and identically distributed according to

$$X_i(t) = \begin{cases} -\frac{1}{4}R_i(t_1) \cdot \text{RTT} & \text{with probability } \frac{1}{8} \\ +\frac{1}{4}R_i(t_1) \cdot \text{RTT} & \text{with probability } \frac{1}{8} \\ 0 & \text{with probability } \frac{6}{8} \end{cases}$$

Our goal will be to bound $\sum_{i=1}^{n} X_i(t)$ and apply the usual argument with Lemma 3.

Note that $\mathbb{E}\, X_i(t) = 0$. Let $c_i = |X_i(t) - \mathbb{E}\, X_i(t)| = |X_i(t)| = R_i(t_1) \cdot \text{RTT}/4$. By Δ-almost fairness, c_i is at most $\frac{\Delta \text{BDP}}{4n}$. Applying Azuma-Hoeffding,

$$\begin{aligned} \Pr\left(\sum_{i=1}^{n} X_i(t) < t\right) &\leq \exp\left(-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}\right), \\ &\leq \exp\left(-\frac{t^2 2n}{\Delta^2 \text{BDP}^2}\right). \end{aligned}$$

Fix some $\delta > 0$. If $t = \frac{\Delta \text{BDP}\sqrt{\ln 1/\delta}}{\sqrt{2n}}$, then

$$\Pr\left(\sum_{i=1}^{n} X_i(t) < t\right) \leq \delta.$$

By definition of $X(t)$, $W_i(t) = \text{BDP} + B - X(t)$. We have just shown that with probability at least $1 - \delta$, $W_i(t) \geq \text{BDP} + B - t$. Therefore by Lemma 3, if $B > t$ then $Q(t) > 0$ with probability at least $1 - \delta$. □

4.13 Minimum number of flows which must decrease windows

Many of our results have a condition on the number of flows which decrease their windows that includes an expression like $\frac{n^2}{\Delta \text{BDP}}$. This is related to the number of flows which must decrease their windows in order for $W(t+1)$ to be lower than $W(t)$.

To see this, let $D(t)$ be the set of Reno flows which decrease their windows during RTT t. Then the window in the next RTT is

$$W(t+1) = W(t) - \frac{1}{2}\sum_{i \in D(t)} w_i(t) + n - |D(t)|.$$

Let w^- be the smallest window in $D(t)$. By a union bound, we have

$$W(t+1) \leq W(t) - \frac{1}{2}|D(t)|w^- + n - |D(t)|.$$

In order for $W(t+1) \leq W(t)$, we need

$$|D(t)| \geq \frac{n}{1 + w^-/2}.$$

If we let $w^+ = \frac{\Delta \text{BDP}}{n}$, the condition we have on $|D(t)|$ in the theorem statements is

$$|D(t)| \geq \frac{n}{w^+}.$$

There are two differences between the two. Since we want a lower bound on $W(t+1)$ to bound the minimum buffer size, the condition has w^+ instead of w^-. We also chose to use a slightly looser bound of $(1 + w^+/2) \leq w^+$ in (4.4) to simplify the required buffer size.

4.14 Proof of Lemma 10

The proof of Lemma 10 is a standard application of a Chernoff bound.

Proof. First, we calculate $\mathbb{E}\, D(t)$,

$$\mathbb{E}\, D(t) = np \leq \frac{n^2}{\Delta \text{BDP}} \tag{4.5}$$

Note that $D(t) - \mathbb{E}\, D(t)$ is the sum of independent random variables, and each term is at most 1 in absolute value since

$$|D_i(t) - \mathbb{E}\, D_i(t)| \leq \max(1-p, p) \leq 1.$$

Applying Azuma-Hoeffding, for $\varepsilon > 0$ we have

$$\mathbb{P}\left(D(t) - \mathbb{E}\, D(t) > \varepsilon\right) \leq \exp\left(-\frac{\varepsilon^2}{2n}\right).$$

Plugging in (4.5) and $\varepsilon = \sqrt{n}$, we have the desired result:

$$\begin{aligned}\mathbb{P}\left(D(t) > \frac{n^2}{\text{BDP}} + \sqrt{n}\right) &\leq \exp\left(-\frac{n}{2n}\right), \\ &\leq \exp(-1/2).\end{aligned}$$

□

We could get a higher probability bound, for instance for any $\delta > 0$, by increasing the upper

bound on $D(t)$ to $\frac{n^2}{\Delta \text{BDP}} + \sqrt{2n \ln(1/\delta)}$. This would change the assumption in Theorem 6, and one could adapt the proof of Theorem 6 accordingly.

Chapter 5

Buffer sizing and Video QoE Measurements at Netflix

5.1 Introduction

Internet routers have buffers to avoid discarding packets during times of congestion. Despite decades of work by researchers and in industry, we still have a poor understanding of the correct size for a router buffer.

The networking community generally believes that a too-large router buffer will increase delay and a too-small buffer will increase loss. However, it is not known if this trade-off is fundamental or simply an artifact of a particular congestion control algorithm. More practically, the broader consequences of changing buffer size are also not well understood. There has been a long line of work on how buffer sizing affects network metrics [98, 185, 8, 20, 60, 171, 47, 69], but relatively little work [87] examining the impact of buffer sizing on application performance.

In this work, we describe the results of a series of buffer sizing experiments we ran in collaboration with Netflix, which begin to shed some light on this question. Netflix is one of the largest source of internet traffic in the world, and has a number of congested links in different locations. We tested a variety of buffer sizes at some of these congested locations, and observed the effects on TCP New Reno and on the video quality of experience.

We find that properly sizing a buffer is both crucial for improving network and video quality of experience, and also quite difficult to do. Video quality of experience depends on a number of factors including: the play delay—or time it takes to start the video, the visual quality of the video streamed by the client, and the number of rebuffers—the times when video playback pauses because a client has no video to stream. We have experimented with both too-small and too-large buffers, both of which can negatively impact video performance—affecting play delay on the order

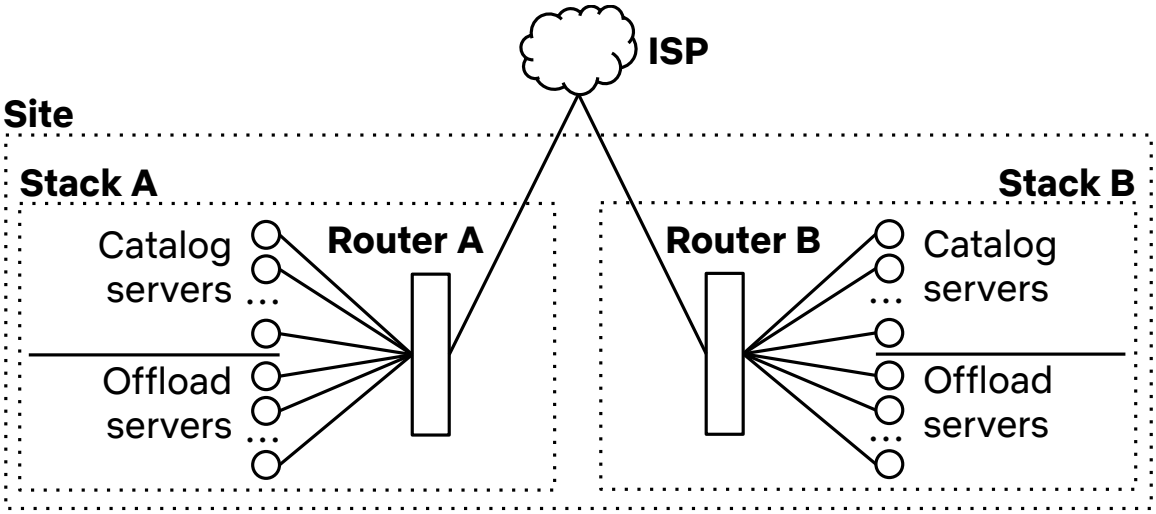


Figure 5.1: An example site used for experiments. Traffic from the ISP was randomly assigned between the A and B stacks

of seconds, decreasing the visual quality of video streamed by 5-10%, and increasing the median rate of rebuffers by 50%.

Our results also show that the question of sizing router buffers is affected by the internal workings of the router. Prior work on buffer sizing assumes that the router is *output queued*; i.e. when a packet arrives, it is immediately placed in a queue at the output port, and its departure time is unaffected by packets destined to different outputs. In our experiments, the routers used combined input and output queueing (CIOQ), with large virtual output queues (VOQs) at the ingress, and a small queue at each output. Arriving packets are initially placed in an ingress VOQ. An internal scheduling algorithm transfers packets from the VOQs to the output queue, approximating the behavior of an output queued switch.

While it is theoretically possible for a CIOQ switch to perfectly emulate an output queued switch [40], in practice the emulation is imperfect, depending on the internal speedup and scheduling algorithm. As we will see later, this imperfection complicates our results, and masks some of the clarity we were seeking in our experiments. For example, we are unable to tell whether a bandwidth-delay product is a good or bad size for a buffer, because we are unable to measure the rate at which buffers are served. We are able to make less specific conclusions, like on the general trend of whether a larger or smaller buffer improves performance. We plan to address these concerns in future work.

To summarize, our main observations are:

1. For metrics related to TCP New Reno, the behavior matches intuition: packet loss increases and RTT decreases as buffers shrink.

2. Video performance has a sweet spot in terms of buffer size: buffers can be both too small and

too large, and in both cases increase the number of rebuffers and decrease the video quality.

3. The ability of the router to emulate perfect output queueing has implications on performance and buffer size.

5.2 Related Work

Since at least 1990, the rule of thumb for sizing router buffers has been that they need to be at least as large as one bandwidth-delay product (BDP) [98, 185]. The justification for this rule of thumb comes from the buffer size needed for a single TCP Reno flow to keep a bottleneck link fully utilized.

In 2004, Appenzeller et al. [8] argued that as the number of flows increases, the bottleneck link can be kept fully utilized with a much smaller buffer. This is because the aggregate arrival rate concentrates around its expectation. In this setting, they argue that a buffer of size $\text{BDP}/\sqrt{N}$ (where N is the number of TCP Reno flows sharing the bottleneck link) is sufficient to keep a link fully utilized. Experiments by Level 3, Internet 2, and on the Stanford campus supported this result [20].

Continuing this line of work, Enachescu et al. [60] showed that if arrivals are distributed according to a Poisson process (either by assumption, multiplexing, or pacing), then the required buffer is small—at most one hundred packets. To reduce the dependence on specific models of TCP, Stanojević et al. [171] propose an adaptive algorithm to shrink the buffer as long as the link utilization is above a predetermined threshold.

After this line of work, one might think that buffers should be reduced as far as possible. In our experiments, we find cases where reducing the sizes of buffers both helps and hurts. Furthermore, we observe interesting QoE effects well before a link becomes under-utilized—which this line of work does not address.

Dhamdhere et al. argue the opposite in [47]. Among other results, they describe a model in which packet loss is proportional to the number of flows using a bottleneck link, so the buffer must grow proportionally to the number of flows in order to limit loss. We also observe similar behavior for loss, which we describe in Section 5.4.1. However, more loss is not necessarily a bad thing, and we report results from experiments in Section 5.4.2 where loss significantly increased but quality generally improved.

In the early 2010s, Gettys et al. [69] revisited the question of buffer sizing, observing that the oversized buffers in cable modems cause massive delays and standing queues during congestion, often referred to as *bufferbloat*. This observation led to, among other contributions, the development of new AQM algorithms [137, 140] and new congestion control algorithms [33]. We observe similar large delays and standing queues in internet routers in Section 5.4.1.

Hohlfeld et al. in [87] run a testbed study on the effect of buffer sizing on the QoE of VOIP, RTP video streaming, and web traffic. Among other findings, they observe that buffer sizing can cause

Experiment parameters				Application effects			TCP effects	
Site	A Size	B Size	Hr.	Sess. w/ Rebuff.	Low Qual. Sec.	Play Delay	Min. RTT	Retrans.
#1	50M	500M	46	+46.3%	+5.7%	-5.9%	-10.9%	+85.2%
#1	250M	500M	30	+5.5%	-1.6%	-3.6%	+9.6%	+17.1%
#1	750M	500M	13	+10.6%	+7.2%	+7.8%	-19.2%	-1.6%
#1	1G	500M	15	+4.9%	+9.6%	+9.5%	-16.4%	-10.4%
#2	5M	50M	14	+0.5%	-2.1%	-5.7%	-13.5%	+51.1%
#2	12M	50M	22	+33.9%	-4.0%	-6.0%	-19.1%	+68.7%
#2	25M	500M	90	-15.6%	-5.3%	-13.5%	-34.8%	+130.6%
#3	50M	500M	34	-22.1%	-7.0%	-14.8%	-5.1%	+134.8%

Table 5.1: Average percent change in application performance during congested hours. A positive value corresponds to an increase in that metric for the canary. Lower values correspond to an improvement in the metric for the "A Buffer" size. Statistically significant results (p=0.01) are highlighted in gray. For more information, see Section 5.4
.

a significant change in QoS metrics (e.g. packet loss, RTT) which result in a much smaller change in QoE metrics. We observe the same, in some cases doubling the rate of packet loss resulting in a much smaller change in QoE metrics.

5.3 Experimental Methodology

In this section we describe the Netflix architecture, our experiment, and the metrics we use to evaluate the results.

5.3.1 Netflix Open Connect Architecture

Netflix's CDN is called *Open Connect.* It consists of many points of presence around the world, called "sites". Figure 5.1 shows an example site. For fault tolerance, sites are split into two "stacks." Each stack consists of a router, a set of *catalog* servers which store the entire Netflix catalog, and a set of faster *offload* servers which store a smaller set of the most popular videos, so named since they "offload" the popular videos from the catalog servers. The traffic is video traffic. It is primarily long-lived TCP New Reno flows, though it also contains a non-negligible number of short-lived flows. Please refer to [135] for more information.

5.3.2 Router Architecture

Our experiments were done using Combined Input-Output Queued (CIOQ) internet routers. These routers employ a different buffer architecture than the output queued (or shared buffer) routers of existing work. In an output queued router, all input ports place packets into the same buffer, and

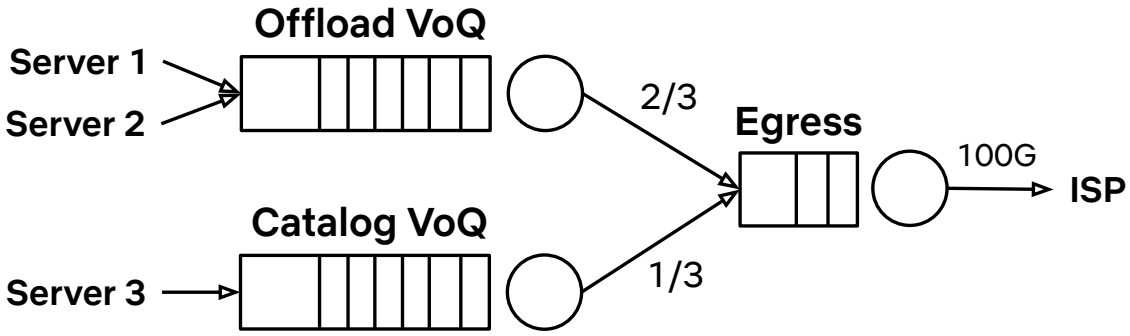


Figure 5.2: Diagram of router buffer architecture. Packets are queued in each VoQ upon arrival, and are drained at a rate chosen by the egress scheduling algorithm.

this buffer is drained by the output port. In a CIOQ router, input ports place packets into different Virtual Output Queues (VOQs), which are drained according to a scheduling algorithm.

The crucial difference between these two models is the rate at which these buffers are drained: in existing work, buffers are drained at a constant rate which corresponds to the port speed of the output port. In our routers, the rate at which a VOQ is drained is chosen by an internal scheduling algorithm, and can vary over time.

Figure 5.2 shows an example internal architecture of the routers used in our experiments. The routers are divided into line cards. The buffers for each line card are logically divided into VoQs. Each output port has a few dedicated VoQs on each line card. Each input port is assigned to one of these VoQs, and usually each VoQ has three or four associated servers.

When a packet arrives from a server, it is placed into the corresponding VoQ. The VoQ sends packets to the egress port when allowed to by a scheduler. The egress port has a small 100KB buffer, intended to briefly store packets before transmission and to aid with packet reconstruction.

The size of the VoQ is configurable, and we configure the router so that arriving packets are dropped when the VoQ is full. When we set a buffer size in our experiments, for instance 500MB, we are setting all VoQs to have a limit of 500MB.

We used the default scheduling algorithm for the router in our experiments, which is Deficit Weighted Round Robin (DWRR), with weights proportional to the aggregate capacity of the input ports. For instance in Figure 5.2, if all servers are connected with 100G links, VoQ 1 will have a weight of 300 and VoQ 2 a weight of 200. We discuss the implications of this scheduling algorithm in Section 5.5.

5.3.3 Our Experiment

We first identified a number of sites with persistent congestion to a peer during peak hours. In each site, we ensured that the ISP's traffic was assigned to the two stacks independently and uniformly

at random. We ran experiments with the same buffer sizes on both stacks, and observed minimal differences in the amount of traffic and quality metrics. This gave us a controlled A/B test which allowed us to measure the effect of buffer sizing on quality metrics.

We configured the pair of stacks to use a variety of different buffer settings. This resulted in setting the buffer size for each Virtual Output Queue (VoQ) in the router to the corresponding setting. We will discuss more about the implications of this in Section 5.5.

We observed the difference in performance on both application-level and TCP-level metrics. All traffic in our experiments used TCP New Reno.

5.3.4 Confounding issues

There are a few confounding issues which limit the generalizability of our results. The major one is the Router's VoQ architecture and scheduling algorithms. The scheduling algorithm means that each queue is served at a variable rate. It is possible that this biases the effect on metrics in one way or another, for instance if an offload VoQ is served at a low rate and a catalog queue is served at a high rate, this could result in some traffic experiencing worse congestion than it otherwise would.

Our experiment is not a perfect controlled trial. Netflix clients can switch between the different stacks. If a video chunk fails to be downloaded from one stack, for example, the client might switch to the other stack or another site entirely. This can also bias results. For instance, we could see an increase in rebuffers because a certain buffer size caused worse QoE, or because the *other* buffer size resulted in very bad QoE and the first stack became more heavily loaded.

5.3.5 Metric Definitions

A *congested hour* is an hour where the per-second link utilization, averaged over one hour, exceeds 98%. In our experiments, this is the point at which we begin to see an increase in RTT due to congestion.[1]

A *session* refers to one TCP connection between a client and a Netflix server.

The *minimum RTT* for a session is the minimum time between when a packet is sent and its acknowledgment arrives for all packets in a session. Note that it is the absolute minimum, not the minimum of TCP's smoothed RTT.

We measure loss via the *percentage of retransmitted bytes*, which is the fraction of all TCP bytes sent during a time period which are retransmitted.

A *rebuffer* is when video playback halts because data is not available to the client. We look at the percentage of sessions with at least one rebuffer.

[1] The fact that we do not observe an affect on RTT until exceeding 98% utilization is a bit surprising. Previous work defines congestion with a much lower threshold, for instance [20], defines a congested hour as one which exceeds 75% link utilization.

We measure video quality using VMAF [25], which models how people perceive the subjective quality of video. The *low quality seconds* metric measures the time-weighted fraction of frames with a VMAF below 80.

In a few cases, we've normalized sensitive metrics. This was done by dividing all metrics in a graph by the largest value in that graph.

5.4 Experiment results

In this section, we discuss the results of our experiments, both on TCP-level metrics and on video QoE metrics.

Table 5.1 summarizes the results of our experiments, and their effect on video performance. In each row, we report the average percent difference in various performance metrics during congested hours (where link utilization was at least 98%) between the traffic to the A and B stacks. A negative value corresponds to a lower number for the A Stack. For all presented metrics, lower is better. We compute bootstrap confidence intervals [59] for $p = 0.01$, and highlight cells where the confidence intervals does not include zero. As a sense of scale, all statistically significant results we observe are quite large compared to other experiments at Netflix.

5.4.1 Impact on TCP New Reno

For TCP New Reno, our results match with intuition: as buffers get smaller, we observe an increase in packet loss and decrease in RTT.

Figure 5.3 shows an example impact of very large buffers on RTT in Site #2. When buffers get very large, there are many negative effects—we observe high RTTs and large variation in RTT. Our preliminary results suggest this could be due to an increase in the percentage of sessions which are receive-window and not congestion-window limited.

We observe that buffers add a consistent delay to TCP flows. It is commonly assumed (*e.g.* [87]) that if a flow sends enough packets during congestion, eventually the flow will observe a RTT which corresponds to the propagation delay through the network with no queueing. Our results show that this is *not* the case. For instance, Figure 5.4 shows the minimum RTT distribution in Site #1, with two moderately sized buffers (50MB and 500MB). If this assumption were true, we would expect the RTTs during the uncongested and congested hours to be similar, but they are not.

One surprising thing about Figures 5.3 and 5.4 is how much larger the RTT is for the 500MB buffer in Site #2 than in Site #1. We will explain this in Section 5.5.

We observe that loss increases as the size of the buffer decreases, and as the number of flows increases. Figure 5.5 shows that the percentage of retransmitted bytes for an hour increases as load increases, and increases more quickly for smaller buffers. Since hours with higher load tend to have more flows, this aligns with the predictions of [47, 133, 144].

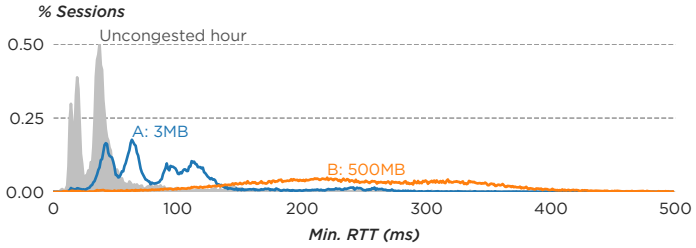


Figure 5.3: Distribution of minimum observed RTT by a TCP session during a congested hour in Site #2

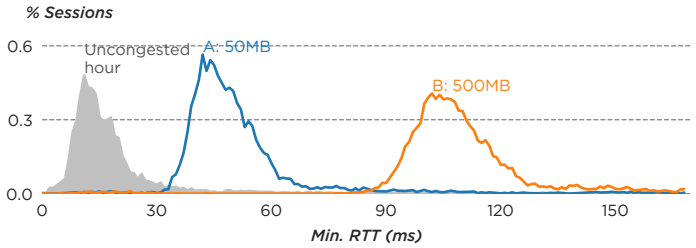


Figure 5.4: Distribution of minimum observed RTT by a TCP session during a congested hour in Site #1

We found this behavior a bit surprising. One might hope that due to the large number of flows at Netflix, the queue would approximately behave like an $M/M/1$ queue of size B. In particular, loss probability would not strongly depend on B once B is more than a hundred or so packets. This is not supported by our results, however.

5.4.2 Impact on Video

Buffer sizing has a big impact on video streaming quality. We generally find that there is a buffer sizing sweet spot for video streaming, and that a buffer which is too small or too large can negatively impact quality.

We consistently observe that reducing the size of a buffer reduces the video play delay.

Once the buffer size becomes too large (e.g. Site #2 and #3 with a 500MB buffer), we see an increase in the number of rebuffers, increases in the amount of low quality video streamed, and an increase in the time it takes to start a video.

If the buffer is too small (e.g. Site #1 , we see the opposite effect: an increase in rebuffers, increase in low quality video. The 50MB buffer for TCP New Reno in Site #1 is an example of a too-small buffer.

We note that the increase in rebuffers we observe due to the smaller buffer is not *solely* due to the increase in packet loss. While rebuffers are correlated with loss, Figure 5.6 shows that larger buffers tend to have a higher rate of rebuffers for similar loss rates. Furthermore, we see cases where

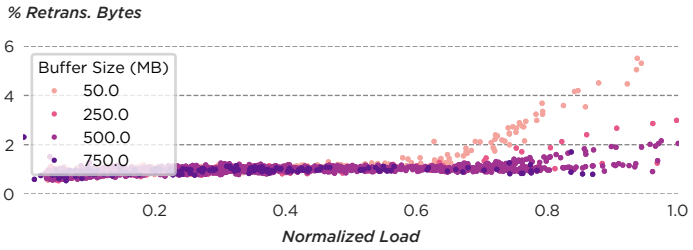


Figure 5.5: Percentage of retransmitted bytes as a function of normalized load in Site #1. Each point represents an hour. Retransmits tend to increase as load increases, and increase faster for smaller buffers.

Figure 5.6: Rebuffers as a function of the percentage of retransmitted bytes. For hours with similar percentages of retransmits, the smaller buffer router has lower rates of rebuffers.

a buffer size which causes an increase in retransmits can correspond with an increase in overall QoE, for instance Site #2 with a 25MB buffer or Site #3 with a 50MB buffer.

5.5 Router architecture impacts choice of buffer size

In Section 5.3.2, we described the architecture of the router buffers in our experiment. In this section, we describe some of the effects of this architecture in conjunction with the particular scheduling algorithm we used.

Recall that the VOQ scheduler used a Deficit Weighted Round Robin (DWRR) policy. DWRR associates a weight with each VOQ, and ensures that the *departure rates* of the VOQs are proportional to their weights. However, if one or more VOQs have a lower arrival rate than their weighted fair share, then that VOQ will be served at its arrival rate and the remaining bandwidth will be allocated among the remaining VOQs. For instance, if the "Catalog VOQ" in Figure 5.2 has an arrival rate lower than 33 Gbps, for instance 20 Gbps, then all 20 Gbps of its traffic will be sent, and the remaining 80 Gbps will be given to traffic from the "Offload VOQ". If this happens, the "Catalog VOQ" will experience no queueing delay.

On its own, this is standard behavior for a VOQ-based system. However, in our setting, there were a number of additional factors:

1. The VOQs were configured to be shared by many servers.

2. The scheduling algorithm set the weight of each VOQ based on the total available capacity plugged into the input ports for that VOQ. For instance, if three 100G servers were plugged into a VOQ, that VOQ would essentially have a weight of three.

3. Due to the cabling, most VOQs consisted of entirely offload servers or entirely catalog servers.

The combination of these factors caused lots of surprising behavior throughout our experiments. Including:

Large queueing delay. Figures 5.4 and 5.3 both show the observed Minimum RTT distributions in two different sites, during an experiment where one buffer was set to 500MB. Both are for congested 100G ports, so we expected to see an increase in Min. RTT of about 40ms. Instead, we observe an 80ms increase in Figure 5.4 and a *100-300ms* increase in Figure 5.3. This difference is due to the number of cores: Site #1 has two cores, and Site #2 eighteen, so the VOQ in Site #2 is being served at a fraction of the rate of Site #1.

Unfairness between cores. Figure 5.7 shows the distribution of RTTs for seven of Site #2's VOQs during an uncongested and a congested hour. Despite having the same size and a similar RTT distribution during the uncongested hour, the RTT distributions are very different during the congested hour. This is because each VOQ is being served at a different rate by the scheduling algorithm.

Uncongested traffic during congestion. We expected that all traffic sharing a congested port would experience congestion. However, if some VOQ did not have enough traffic to meet its weighted fair share, it would experience no congestion. Figure 5.8 shows the RTT distribution for the hours leading up to and after peak in Site #1, for traffic using a congested port. Because the catalog servers did not send their weighted fair share of traffic, they experienced no congestion.

We have been working with our router vendor to ameliorate some of these issues, and we believe that it will be possible to fix much of this behavior via a firmware upgrade. However it is not clear to us what the right scheduling behavior is nor how to size buffers given whatever scheduling behavior we observe, and we leave this question for future work.

5.6 Conclusion

We find that TCP Reno generally behaves as expected when changing buffer sizes, with a smaller buffer causing higher loss and lower delay.

The story for the best buffer size for video is much more complicated, but we find that improvements in buffer size can dramatically improve video QoE.

We also find that considering application QoE is crucial when sizing router buffers. Existing buffer sizing work has generally focused on network-level metrics, for instance whether a link is fully utilized [8, 20, 60, 171], loss, queueing delay [47]. In all our experiments there were interesting

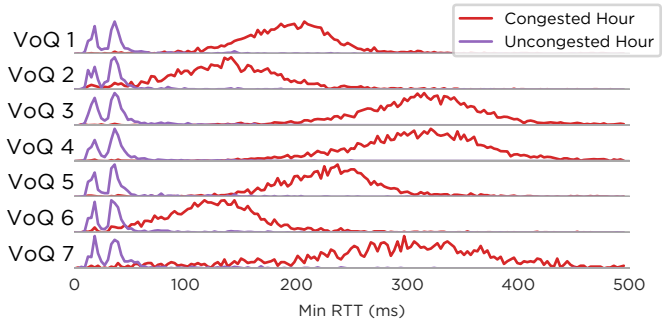


Figure 5.7: Distribution of the minimum RTTs observed by TCP flows sharing a router, one row per VOQ. Each VOQ has the same size.

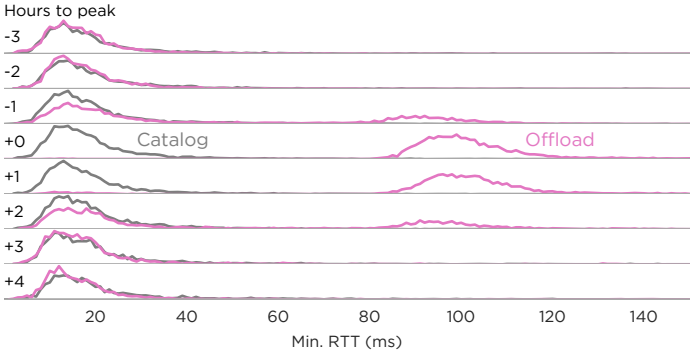


Figure 5.8: Normalized distributions of the minimum RTT observed by TCP flows from one stack, split by Catalog and Offload servers with one row per hour.

effects on video QoE which went beyond whether the link was fully utilized, and whether loss or delay increased.

An intuitive idea is that if a buffer never goes empty during periods of congestion, then it could be shrunk to improve (or at least not degrade) application performance. Our results suggest that this intuition is not true. In our experiments with smaller buffers, we observe negative effects on applications (e.g. an increase in rebuffers) while still seeing an elevated distribution of minimum RTTs.

We plan on continuing this line of work. We are exploring ways of understanding and experimenting with buffer sizing in the presence of the VoQ scheduling algorithms, getting a better understanding of the mechanisms by which buffer sizing affects video QoE, and understanding how different congestion control algorithms affect buffer sizing.

Chapter 6

Conclusion

6.1 Dissertation summary

Most internet traffic today is video, characterized by an on-off, bursty traffic pattern: alternating between periods of sending data and periods of silence every few seconds. This on-off behavior happens whenever throughput exceeds the video's bitrate. Modern congestion control algorithms try to maximize throughput, making video traffic more bursty as home network speeds increase [167, 43] and video bitrates decrease [108].

Chapter 2 shows how smoothing video traffic can improve performance for neighboring applications sharing the same network. By picking throughput based on video needs (instead of maximizing throughput), we can avoid congestion completely. This reduces packet loss and queueing delay, and gives neighbors more instantaneous throughput. Remarkably, video QoE does not suffer—in fact it *slightly* improves. Video services are also incentivized to smooth their traffic: because large scale streaming services compete with themselves in many networks across the internet, being friendlier to their neighbors means improving performance for themselves.

Chapter 3 shows the difficulties in experimenting with new algorithms like Sammy that affect internet congestion at scale. The networking research community typically evaluates these algorithms with A/B tests, but I show that this can lead to biased results and even switch the *direction* of improvement—an algorithm that appears worse in an A/B test might actually improve performance when deployed, and vice versa. In an experiment at scale in a congested network, I show that while bitrate capping reduces congestion, it appears to *increase* congestion in an A/B test. Chapter 3 also suggests ways for researchers to deal with this bias, including alternate experiment designs.

Finally, in Chapter 4 I revisit the long-standing problem of sizing buffers in routers. Prior work suggested that buffers can be reduced by a factor of square root of the number of flows n using the network, offering dramatic buffer reductions in networks carrying many flows. However, the result required TCP Reno, did not make clear how to determine n, or what happens when windows are not

independent. Our results clarify that n is the number of flows that reduce their window size in the same RTT, removes the need for independence, and holds for a broader class of congestion control algorithms. Our results explore what happens when buffers are sized too small for full link utilization and make it easier to run buffer sizing experiments, which should give much more confidence that the results apply broadly. In Chapter 5 I looked at how sizing buffers affect production video traffic, and found that video QoE is impacted well before link utilization is.

6.2 Reflections on congestion control

My thesis approaches congestion from the application perspective, investigating how modifying video traffic can reduce congestion on the internet. My primary approach to this was smoothing video traffic with Sammy, which dramatically reduces congestion both in a lab setting, and in experiments run at scale at Netflix (reducing RTTs by 20% and retransmits by 50%). I also explored a complimentary approach in Chapter 3: capping video bitrates. In our stack-based experiment, capping bitrates reduced RTTs by 25%, play delays by 10%, and increased throughput by 10%.

These two approaches effectively reduce congestion, but they are very different from the way networking research approaches congestion control today. While working on this research, I had a number of conversations in which people very reasonably argued that smoothing video traffic and capping bitrates were not congestion control algorithms at all. In light of this, in this section I discuss how we approach congestion control today and argue that we should reconsider our approach.

Today, the problem of congestion control is widely considered to be a problem of allocating resources at the transport layer [141, 192]. There's a resource (the link capacity), and it needs to be allocated among all the congestion control algorithms using it. The problem is to maximize throughput and ensure that the link is fully utilized, while ensuring that the allocation does not cause congestion (i.e. exceed the capacity of the link) and that the allocation is fair (i.e. equal, or optimizing some utility function).

This goal of sending as fast as possible is at the core of almost all congestion control research. In Jacobson's original paper on congestion control [97], he justifies sending faster when there is no congestion with the following example:

> You could have been sharing the path with someone else and converged to a window that gives you each half the available bandwidth. If she shuts down, 50% of the bandwidth will be wasted unless your window size is increased.

In the first papers on network utility maximization [110], which have formed the theoretical basis for recent congestion control algorithms [51, 11, 134, 131], the assumption is that utility functions are strictly concave and increasing. That is, that higher throughput gives strictly higher utility. In buffer sizing work, the goal is to reduce the size of the router buffer as much as possible without reducing link utilization [8, 98, 163]. Maximizing throughput is so core to the way we think about

congestion control today, that it's rarely questioned. Here's the way a recent book [141, Ch. 3.2.1] describes the goal of congestion control:

> A good starting point for evaluating the effectiveness of a congestion-control mechanism is to consider the two principal metrics of networking: throughput and delay. Clearly, we want as much throughput and as little delay as possible.

Because congestion control algorithms aim to maximize throughput, the congestion control problem is thought of as a zero-sum game. If one algorithm increases its throughput while sharing a network with another algorithm, that other algorithm gets less throughput. We think of this as "unfair" [17, 2, 191, 28, 118, 13, 158, 94, 51, 189, 30, 93, 105, 182, 183, 101] and talk about the harm to the algorithm which gets lower throughput [189]. One of the major challenges in designing new congestion control algorithms is remaining fair to current algorithms, while simultaneously improving throughput and reducing congestion in networks where current algorithms do not perform well. It's common for papers to come out shortly after a new algorithm is proposed, showing that the new algorithm is unfair to a particular existing algorithm in a particular network (e.g. [105, 183, 85]). At their core, these challenges come from way that congestion control algorithms are designed to maximize the throughput of individual flows.

Taking a step back, this is a strange way of thinking about internet traffic. The internet exists because people want to *do things* online. We even have good statistics about what people want to do: in 2022, 75% of internet traffic was video streaming, 6% was marketplaces and gaming, 5% was social networking, 5% was web browsing, and so on [152]. Users of the internet do not care about allocating network resources, they care about doing things like watching videos, playing games, or talking to their friends.

In light of this, this thesis approaches congestion control from the perspective of video traffic (by far the largest application today). There has been extensive research into what it means for video traffic to perform well: namely, that QoE is high. Because of this we can focus on maximizing QoE and avoid more typical goals of congestion control. Most notably, the goal of this work has been to *minimize throughput* without reducing QoE, which is the opposite *direction* of the goal of typical congestion control research. Because of this, we are able to step outside of the zero-sum game and focus on improving performance for neighbors. For example, the results of the lab experiments in Section 2.6 are by definition not zero-sum: Sammy decreases its throughput, maintains video QoE, and improves performance for neighboring applications.

The results of this new approach are promising. When video traffic is sending data, we reduce its throughput by more than 60%, reduce retransmits by more than 50%, and reduce round-trip times by more than 20%. We do this while improving performance for neighbors, in the lab improving throughput for TCP, reducing delay for UDP, reducing flow completion time for HTTP, and improving the play delay of video traffic.

6.3 Future directions

This thesis outlines a new approach to controlling congestion on the internet, and in doing so it opens up a number of new research directions.

6.3.1 Congestion control for video traffic

This thesis focuses on reducing congestion when network capacity is significantly higher than the maximum bitrate of the video—the most common case on the internet today. To do so, we have relied on existing congestion control algorithms, limited via application-informed pacing. But there are important networks where capacity is lower than maximum bitrates, such as in mobile networks and in parts of the world with worse connectivity, and we could go further and reduce congestion in these networks. Approaching congestion control with the goals of video traffic in mind gives us an opportunity to revisit congestion control for video traffic from the ground up.

I am imagining here a new algorithm for video traffic that is responsible for the tasks of existing ABR and existing congestion control algorithms. Such an algorithm would choose both video bitrates and when to send each individual packet. This algorithm could incorporate recent ideas from congestion control for video, including smoothing video traffic, VMAF-based fairness [134], and scavenger algorithms [131]. These recent congestion control algorithms rely on assumptions about the behavior of ABR algorithms (e.g. by assuming some utility function is provided that matches ABR behavior), and ABR algorithms in turn rely on assumptions about congestion control algorithms and the throughput they produce (e.g. assuming that past throughput is predictive of future requests). A jointly designed algorithm could avoid some of these assumptions, and potentially improve performance.

One particularly intriguing idea is to move congestion control into the application layer, and make the ABR algorithm responsible for congestion control. The ABR algorithm could request a particular sending rate for each HTTP request it issues, and the server would just send at that rate. This would make it easy to develop new congestion control algorithms (even in third-party CDNs, where a video provider currently cannot develop new algorithms). It would also allow the ABR algorithm to use additional information to choose throughputs, including buffer levels, historical throughput, video quality (and VMAF), player state, and the many other things ABR algorithms consider.

A glaring challenge is how to deal with congestion that starts during the middle of a chunk download. There are a number of possible options here:

1. The ABR algorithm could completely ignore the issue and make another decision whenever it issues the next HTTP request. This seems like it would not perform particularly well, but it would be interesting to measure the benefit of sub-chunk congestion control to video traffic and neighboring traffic.

2. The ABR algorithm could cancel partially completed requests after some timeout has elapsed, and issue new requests for the remaining bytes with a new sending rate. This could reduce the duration of congestion.

3. The ABR algorithm could choose how the transport layer responds to congestion. Perhaps by picking a rule that the transport layer uses to adjust the sending rate on each packet, or perhaps by picking a rule that detects congestion and have the transport layer cancel requests whenever the rule detects congestion.

4. We could reduce the size of HTTP requests (perhaps down to the level of just a few packets), so any sudden increase in congestion would be immaterial.

The other extreme would be to move ABR algorithms into the transport layer. This is somewhat common in real-time video traffic, for instance Salsify [67] uses packet trains to estimate throughput and adapts bitrates on a packet by packet level. This would be most useful if the ABR algorithm running at the transport layer could somehow change bitrates on a packet-by-packet basis. Current, widely-used video encoding technologies would not allow for this, but perhaps something like Scalable Video Coding [159] or some future encoding technologies would allow switching bitrates on packet boundaries, perhaps by decoding a lower bitrate frame from a subset of packets of a higher bitrate. It would take some thought to deploy such a scheme in today's large-scale video streaming systems, where clients run ABR algorithms and the transport layer is under the control of a third-party CDN.

Even short of redoing congestion control, continuing to look at how ABR algorithms can inform existing congestion control algorithms is a promising angle for future research. One particularly promising place to start is, naturally, the start of a TCP connection. Today, TCP connections start with an initial window size that is set globally [53, 150], and then proceed to run slow start and double the window until loss occurs [24]. This seems a bit inefficient for video, where requests are for a particular bitrate and ABR algorithms use historical throughput measurements to estimate initial throughput. An alternative could be to have ABR algorithms choose an initial throughput for each request, and have congestion control adapt as the request continues.

6.3.2 Smoothing out video traffic at other time scales

In this thesis we have focused on congestion at two different levels: congestion at the level of a single video session (in the case of Sammy), and congestion at the level of a network (in the case of bitrate capping). Future work could consider other algorithms to smooth out congestion at the network level. At scale, internet traffic has a diurnal pattern [178]: it is highest during peak hours (6-12pm local time), and is significantly lower off-peak. To get a sense of the difference, consider the daily traffic for a large Central European ISP shown in Figure 2a of [27]—the four least-loaded hours have less than 20% the traffic of the peak hour.

Smoothing this traffic out by moving traffic from peak to off-peak hours would benefit network operators in two dfifferent ways:

1. ISP networks are provisioned for peak traffic loads, and so if congestion occurs in ISP networks, it tends to be during peak hours. By reducing traffic volume during peak hours, we could reduce congestion.

2. Network bandwidth is billed based on peak hours. The standard way of billing networks is burstable billing [49], where a month is divided up into five minute buckets and sorted from most to least bytes transferred over these five minutes. The operator is billed for the number of bytes transferred in some percentile bucket, usually the 95th or 90th. By reducing traffic volume during peak hours, we could reduce cost.

But how could we move traffic from peak to off-peak hours? Imagine if video clients had a much larger buffer (large enough to store a couple of videos), and a video service provider could do a reasonably good job at predicting what videos a client might watch during peak hours. Video clients could download the predicted videos off peak. If the predicted videos were played during peak hours, the videos would not need to be downloaded and load would be lower. The benefits of this approach depend on how accurately providers could predict which videos would be watched at peak, and the ratio of peak to off-peak traffic in a particular network. Typically operators add new capacity to networks fast enough so that the load that creates congestion is only slightly higher than the capacity of the network. Even a slight decrease in load can significantly reduce congestion. A detailed exploration of this idea is left for future work.

6.3.3 Application-based congestion control for non-video applications

This thesis has focused on smoothing out video traffic, but there are many other applications that use the internet. One could take a similar approach to congestion control for major non-video uses of the internet (e.g. web browsing, gaming, software updates, etc...). The layering architecture of the internet encourages these other applications to use a similar strategy of allowing congestion control algorithms to select the maximum throughput without application input. We view Sammy as a starting point for using application-level logic to smooth out internet traffic: by using details about the behavior of other applications, we may be able to make other types of internet traffic into friendlier neighbors as well.

As an example, imagine a software update that downloads sometime in the middle of the night, while the user is asleep. If there's an eight hour period when the user is asleep, there's no need to try to download the update as quickly as possible.

As another example, imagine a browser loading a webpage. The browser downloads the HTML of the webpage, parses the HTML, and then issues a series of follow-up HTTP requests to download images, scripts, stylesheets, and so on. The follow-on content is then parsed and executed, which can

result in even more requests, creating a "waterfall" of requests. Given a waterfall and information about when its various elements were shown to a user, one could compute the minimum throughput for each HTTP request that would result in the same user-observed latency. If we were to pace each request at this throughput, we could potentially smooth out the web page traffic without impacting latency. The challenge of designing an algorithm to do this online is left as future work.

6.3.4 Experiments with smoothing video traffic at scale

Throughout this thesis, I have relied on lab experiments and formal reasoning to argue that smooth video traffic improves performance for neighbors. The question of how smoothing video traffic impacts neighboring traffic at scale is still intriguingly open. Using our work in Chapter 3, one could collaborate with other internet services to measure the impact of Sammy (and other algorithms) that impact congestion. For instance, if Netflix ran a switchback experiment with Sammy, they could call up various other large internet services or look at publicly available measurements, and compare the days when Sammy was run to the days when Sammy was not. This would give estimates of the impact of Sammy on its neighbors.

6.3.5 Buffer sizing for video traffic

This thesis discussed the impact of sizing router buffers on congestion control algorithms, but the impact of sizing router buffers on video QoE is an almost wide open research area. Existing work on buffer sizing, Chapter 4 included, focuses on sizing buffers to ensure that congestion control algorithms fully utilize the link. Chapter 4 goes slightly further in ensuring that link utilization is at least some percentage. But when smoothing video traffic, our goal is to make link utilization *as low as possible*. Chapter 5 presents some preliminary experiments, but there is no theoretical understanding on how small router buffers can be without reducing video QoE.

There may be some surprising results here—let's briefly consider the problem of buffer sizing for a single Sammy session. After playback starts, Sammy paces packets around 3x the maximum video bitrate. If this rate is lower than the capacity of the network, *no buffer is required* for good QoE. The packet-by-packet arrival rate is lower than the link capacity, and so no queues will build up. A complete understanding of the buffer size required for Sammy (including the initial phase and with multiple sessions) is still an open question.

6.4 Concluding remarks

This thesis suggests an alternate approach to reducing congestion on the internet, and is just the beginning of an area of research around application-based congestion control, and ways of reducing congestion at scale. As a community, we have an opportunity to further smooth internet traffic,

video traffic and beyond. After all, a smoother internet benefits everyone.

Bibliography

[1] Alberto Abadie, Joshua Angrist, and Guido Imbens. Instrumental Variables Estimates of the Effect of Subsidized Training on the Quantiles of Trainee Earnings. *Econometrica*, 70(1):27, January 2002.

[2] Amit Aggarwal, Stefan Savage, and Thomas E Anderson. Understanding the performance of TCP pacing. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, volume 3, pages 1157–1165, Tel Aviv, Israel, 2000. IEEE.

[3] Akamai. Common Media Client Data & AMD, February 2023.

[4] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proceeding of the 23rd ACM Workshop on Network and Operating Systems Support for Digital Audio and Video - NOSSDAV '13*, pages 19–24, Oslo, Norway, 2013. ACM Press.

[5] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: Auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, Budapest Hungary, August 2018. ACM.

[6] Julia Alexander. Amazon and Apple are reducing streaming quality to lessen broadband strain in Europe, March 2020.

[7] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center TCP (DCTCP). *ACM SIGCOMM Computer Communication Review*, 40(4):63–74, August 2010.

[8] Guido Appenzeller, Nick McKeown, and Isaac Keslassy. Sizing router buffers. In *ACM SIGCOMM Computer Communication Review*, pages 281–292. ACM, August 2004.

[9] João Taveira Araújo, Raúl Landa, Richard G. Clegg, George Pavlou, and Kensuke Fukuda. A longitudinal analysis of Internet rate limitations. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 1438–1446, April 2014.

[10] Peter M. Aronow and Cyrus Samii. Estimating average causal effects under general interference, with application to a social network experiment. *Annals of Applied Statistics*, 11(4):1912–1947, December 2017.

[11] Venkat Arun and Hari Balakrishnan. Copa: Practical Delay-Based Congestion Control for the Internet. In *Proceedings of the Applied Networking Research Workshop on - ANRW '18*, pages 19–19, Montreal, QC, Canada, 2018. ACM Press.

[12] Consumer Technology Association. CTA Specification: Web Application Video Ecosystem - Common Media Client Data. Technical Report CTA-5004, Consumer Technology Association, September 2020.

[13] Rukshani Athapathu, Ranysha Ware, Aditya Abraham Philip, Srinivasan Seshan, and Justine Sherry. Prudentia: Measuring Congestion Control Harm on the Internet. In *N2Women Workshop*, page 2, 2020.

[14] Susan Athey, Dean Eckles, and Guido W. Imbens. Exact p-Values for network interference. *Journal of the American Statistical Association*, 113(521):230–240, 2018.

[15] Daniel Atkinson, Gustavo Santos, Gaspare Bruno, Edmundo de Souza e Silva, and Renata Teixeira. Buffering at the Edge: Measuring from Home-routers. In *BS '19: 2019 Workshop on Buffer Sizing*, December 2019.

[16] Pat Bajari, Brian Burdick, Guido Imbens, James McQueen, Thomas Richardson, and Ido Rosen. Multiple randomization designs for interference, 2019.

[17] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz. TCP behavior of a busy Internet server: Analysis and improvements. In *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98*, volume 1, pages 252–262 vol.1, March 1998.

[18] Guillaume Basse, Avi Feller, and Panagiotis Toulis. Randomization tests of causal effects under interference. *Biometrika*, 106(2):487–494, February 2019.

[19] Guillaume Basse, Hossein Azari Soufiani, and Diane Lambert. Randomization and the pernicious effects of limited budgets on auction experiments. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and*

Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016, volume 51 of *JMLR Workshop and Conference Proceedings*, pages 1412–1420. JMLR.org, 2016.

[20] Neda Beheshti, Yashar Ganjali, Monia Ghobadi, Nick McKeown, and Geoff Salmon. Experimental Study of Router Buffer Sizing. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement*, IMC '08, pages 197–210, New York, NY, USA, 2008. ACM.

[21] Neda Beheshti, Petr Lapukhov, and Yashar Ganjali. Buffer Sizing Experiments at Facebook. In *Proceedings of the 2019 Workshop on Buffer Sizing*, pages 1–6, Palo Alto CA USA, December 2019. ACM.

[22] Abdelhak Bentaleb, May Lim, Mehmet N. Akcay, Ali C. Begen, and Roger Zimmermann. Common media client data (CMCD): Initial findings. In *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 25–33, Istanbul Turkey, July 2021. ACM.

[23] Thomas Blake and Dominic Coey. Why marketplace experimentation is harder than it seems: The role of test-control interference. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, EC '14, pages 567–582, New York, NY, USA, 2014. Association for Computing Machinery.

[24] Ethan Blanton, Vern Paxson, and Mark Allman. TCP Congestion Control. Request for Comments RFC 5681, Internet Engineering Task Force, September 2009.

[25] Netflix Technology Blog. Toward A Practical Perceptual Video Quality Metric, April 2017.

[26] Iavor Bojinov, David Simchi-Levi, and Jinglong Zhao. Design and Analysis of Switchback Experiments. *arXiv:2009.00148 [stat]*, January 2021.

[27] Timm Böttger, Ghida Ibrahim, and Ben Vallis. How the Internet reacted to Covid-19: A perspective from Facebook's Edge Network. In *Proceedings of the ACM Internet Measurement Conference*, pages 34–41, Virtual Event USA, October 2020. ACM.

[28] Bob Briscoe. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM Computer Communication Review*, 37(2):63–74, March 2007.

[29] Swapna Buccapatnam, Yaron Koral, Simon T Tse, Xiaoqi Chen, and Ken Duell. Fine-grained P4 Measurement Toolkit for Buffer Sizing in Carrier Grade Networks. In *BS '19: 2019 Workshop on Buffer Sizing*, December 2019.

[30] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. When to use and when not to use BBR: An empirical analysis and evaluation study. In *Proceedings of the Internet Measurement Conference*, pages 130–136, Amsterdam Netherlands, October 2019. ACM.

[31] Neal Cardwell. BBR Congestion Control, July 2017.

[32] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, Ian Swett, Jana Iyengar, Victor Vasiliev, and Van Jacobson. BBR Congestion Control: IETF 100 Update: BBR in shallow buffers, November 2017.

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

[34] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Kevin Yang, Ian Swett, Victor Vasiliev, Bin Wu, Luke Hsiao, Matt Mathis, and Van Jacobson. BBR v2: A Model-based Congestion Control Performance Optimizations, November 2019.

[35] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. BBR v2: A Model-based Congestion Control, March 2019.

[36] Erik Carlsson and Eirini Kakogianni. Smoother Streaming with BBR, August 2018.

[37] Nicholas Chamandy. Experimentation in a ridesharing marketplace, December 2016.

[38] Balakrishnan Chandrasekaran, Georgios Smaragdakis, Arthur Berger, Matthew Luckie, and Keung-Chi Ng. A server-to-server view of the internet. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies - CoNEXT '15*, pages 1–13, Heidelberg, Germany, 2015. ACM Press.

[39] Xiaoqi Chen and Hyojoon Kim. Measuring Queues in Campus Network via Link Tapping. In *BS '19: 2019 Workshop on Buffer Sizing*, December 2019.

[40] Shang-Tse Chuang, Ashish Goel, Nick McKeown, and Balaji Prabhakar. Matching Output Queueing with a Combined Input Output Queued Switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, September 2006.

[41] David D Clark. The Design Philosophy of the DARPA Internet Protocols. *ACM SIGCOMM Computer Communication Review*, 18(4):106–114, August 1988.

[42] Federal Communications Commission. Broadband Performance. Technical Report OBI Technical Paper No. 4, Federal Communications Commission, August 2010.

[43] Federal Communications Commission. Measuring Fixed Broadband - Eleventh Report. Technical report, Federal Communications Commission, December 2021.

[44] Bruno Crépon, Esther Duflo, Marc Gurgand, Roland Rathelot, and Philippe Zamora. Do Labor Market Policies have Displacement Effects? Evidence from a Clustered Randomized Experiment. *The Quarterly Journal of Economics*, 128(2):531–580, May 2013.

[45] Amogh Dhamdhere, David D. Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky K. P. Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C. Snoeren, and Kc Claffy. Inferring persistent interdomain congestion. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 1–15, Budapest Hungary, August 2018. ACM.

[46] Amogh Dhamdhere and Constantine Dovrolis. Open issues in router buffer sizing. *ACM SIGCOMM Computer Communication Review*, 36(1):87, January 2006.

[47] Amogh Dhamdhere, Hao Jiang, and Constantinos Dovrolis. Buffer sizing for congested internet links. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 2, pages 1072–1083, Miami, FL, USA, 2005. IEEE.

[48] Nikos Diamantopoulos, Jeffrey Wong, David Issa Mattos, Ilias Gerostathopoulos, Matthew Wardrop, Tobias Mao, and Colin McFarland. Engineering for a Science-Centric Experimentation Platform. *arXiv:1910.03878 [cs]*, October 2019.

[49] Xenofontas Dimitropoulos, Paul Hurley, Andreas Kind, and Marc Ph. Stoecklin. On the 95-Percentile Billing Method. In Sue B. Moon, Renata Teixeira, and Steve Uhlig, editors, *Passive and Active Network Measurement*, Lecture Notes in Computer Science, pages 207–216, Berlin, Heidelberg, 2009. Springer.

[50] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. *ACM SIGCOMM Computer Communication Review*, 41(4):362–373, August 2011.

[51] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, P Brighten Godfrey, and Michael Schapira. PCC Vivace: Online-Learning Congestion Control. In *NSDI*, page 15, April 2018.

[52] Nandita Dukkipati, Matt Mathis, Yuchung Cheng, and Monia Ghobadi. Proportional Rate Reduction for TCP. In *Internet Measurement Conference*, page 15, November 2011.

[53] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing TCP's initial congestion window. *ACM SIGCOMM Computer Communication Review*, 40(3):26–33, June 2010.

[54] Eric Dumazet. Pkt_sched: Fq: Fair Queue packet scheduler [LWN.net], August 2013.

[55] Eric Dumazet. Tcp: TSO packets automatic sizing [LWN.net], August 2013.

[56] Eric Dumazet. Tc-fq(8) - Linux manual page, September 2015.

[57] Ted Dunning. The t-digest: Efficient estimates of distributions. *Software Impacts*, 7:100049, February 2021.

[58] Dean Eckles, Brian Karrer, and Johan Ugander. Design and Analysis of Experiments in Networks: Reducing Bias from Interference. *Journal of Causal Inference*, 5(1), February 2016.

[59] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1–26, January 1979.

[60] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Routers with Very Small Buffers. *INFOCOM*, pages 1–11, 2006.

[61] Rodérick Fanou, Francisco Valera, and Amogh Dhamdhere. Investigating the causes of congestion on the african IXP substrate. In *Proceedings of the 2017 Internet Measurement Conference on - IMC '17*, pages 57–63, London, United Kingdom, 2017. ACM Press.

[62] Fastly. Fastly Developer Hub, February 2023.

[63] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing Web Latency: The Virtue of Gentle Aggression. In *SIGCOMM*, page 12, August 2013.

[64] Ken Florance. Reducing Netflix traffic where it's needed while maintaining the member experience, March 2020.

[65] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, Aug./1993.

[66] Sally Floyd and Van Jacobson. The Synchronization of Periodic Routing Messages. *IEEE/ACM Transactions on Networking*, page 28, April 1994.

[67] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S Wahby, and Keith Winstein. Salsify: Low-Latency Network Video Through Tighter Integration Between a Video Codec and a Transport Protocol. In *NSDI*, April 2018.

[68] Andrew Gelman and Jennifer Hill. Causal inference using regression on the treatment variable. In *Data Analysis Using Regression and Multilevel/Hierarchical Models*, Analytical Methods for Social Research, pages 167–198. Cambridge University Press, Cambridge, 2006.

[69] Jim Gettys, Kathleen Nichols, and Kathleen Nichols. Bufferbloat: Dark Buffers in the Internet. *Communications of the ACM*, 55(1):15, 2012.

[70] Monia Ghobadi, Yuchung Cheng, Ankur Jain, and Matt Mathis. Trickle: Rate Limiting YouTube Video Streaming. In *USENIX ATC*, page 6, June 2012.

[71] Peter Glynn, Ramesh Johari, and Mohammad Rasouli. Adaptive experimental design with temporal interference: A maximum likelihood approach. *arXiv preprint arXiv:2006.05591*, 2020.

[72] Hadas Gold. Netflix and YouTube are slowing down in Europe to keep the internet from breaking, March 2020.

[73] Nirmal Govind. A/B Testing and Beyond: Improving the Netflix Streaming Experience with Experimentation and Data. . . , June 2018.

[74] Carlo Augusto Grazia, Martin Klapez, and Maurizio Casoni. The New TCP Modules on the Block: A Performance Evaluation of TCP Pacing and TCP Small Queues. *IEEE Access*, 9:129329–129336, 2021.

[75] Ilya Grigorik. HTTP: HTTP/1.X - High Performance Browser Networking (O'Reilly), 2013.

[76] Ilya Grigorik and Surma. Introduction to HTTP/2 — Web Fundamentals, September 2019.

[77] Huan Gui, Ya Xu, Anmol Bhasin, and Jiawei Han. Network A/B Testing: From Sampling to Estimation. In *Proceedings of the 24th International Conference on World Wide Web*, pages 399–409, Florence Italy, May 2015. International World Wide Web Conferences Steering Committee.

[78] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, July 2008.

[79] Bruce Hajek. The Proof of a Folk Theorem on Queuing Delay with Applications to Routing in Networks. *Journal of the ACM*, 30(4):834–851, October 1983.

[80] M. E. Halloran and C. J. Struchiner. Causal inference in infectious diseases. *Epidemiology (Cambridge, Mass.)*, 6(2):142–151, March 1995.

[81] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I Know What Your Packet Did Last Hop - Using Packet Histories to Troubleshoot Networks. *NSDI*, 2014.

[82] Eman Salaheddin Hashem. *Analysis of Random Drop for Gateway Congestion Control*. PhD thesis, MIT, November 1989.

[83] Sofiane Hassayoun and David Ros. Loss synchronization and router buffer sizing with high-speed versions of TCP. In *IEEE INFOCOM Workshops 2008*, April 2008.

[84] Masami Hiramatsu. [net-next,v4,1/6] net: TCP: Add trace events for TCP congestion window tracing, December 2017.

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

[86] Janey C. (Janey Ching-Iu) Hoe. *Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes*. Thesis, Massachusetts Institute of Technology, 1995.

[87] Oliver Hohlfeld, Enric Pujol, Florin Ciucu, Anja Feldmann, and Paul Barford. A QoE Perspective on Sizing Network Buffers. In *Proceedings of the 2014 Conference on Internet Measurement Conference - IMC '14*, pages 333–346, Vancouver, BC, Canada, 2014. ACM Press.

[88] David Holtz, Ruben Lobel, Inessa Liskovich, and Sinan Aral. Reducing Interference Bias in Online Marketplace Pricing Experiments. *arXiv:2004.12489 [econ, stat]*, April 2020.

[89] Guanglei Hong and Stephen W. Raudenbush. Evaluating Kindergarten Retention Policy. *Journal of the American Statistical Association*, 101(475):901–910, September 2006.

[90] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference - IMC '12*, page 225, Boston, Massachusetts, USA, 2012. ACM Press.

[91] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM - SIGCOMM '14*, pages 187–198, Chicago, Illinois, USA, 2014. ACM Press.

[92] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. Using the Buffer to Avoid Rebuffers: Evidence from a Large Video Streaming Service. *arXiv:1401.2209 [cs]*, January 2014.

[93] Per Hurtig, Habtegebreil Haile, Karl-Johan Grinnemo, Anna Brunstrom, Eneko Atxutegi, Fidel Liberal, and Ake Arvidsson. Impact of TCP BBR on CUBIC Traffic: A Mixed Workload Evaluation. In *2018 30th International Teletraffic Congress (ITC 30)*, pages 218–226, Vienna, September 2018. IEEE.

[94] Geoff Huston. TCP and BBR, May 2018.

[95] Guido W. Imbens and Donald B. Rubin. *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction*. Cambridge University Press, USA, 2015.

[96] Alexey Ivanov. Evaluating BBRv2 on the Dropbox Edge Network. *arXiv:2008.07699 [cs]*, August 2020.

[97] Van Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, August 1988. Association for Computing Machinery.

[98] Van Jacobson. Modified TCP congestion avoidance algorithm, April 1990.

[99] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *ACM Transactions on Computer Systems*, September 1984.

[100] Junchen Jiang, Vyas Sekar, Henry Milner, Davis Shepherd, Ion Stoica, and Hui Zhang. CFA: A Practical Prediction System for Video QoE Optimization. In *NSDI*, March 2016.

[101] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming With Festive. *IEEE/ACM Transactions on Networking*, 22(1):326–340, February 2014.

[102] Ramesh Johari, Hannah Li, Inessa Liskovich, and Gabriel Weintraub. Experimental design in two-sided platforms: An analysis of bias. *arXiv preprint arXiv:2002.05670*, 2020.

[103] Youngmi Joo, Vinay Ribeiro, Anja Feldmann, Anna C. Gilbert, and Walter Willinger. TCP/IP traffic dynamics and network performance: A lesson in workload modeling, flow control, and trace-driven simulations. *ACM SIGCOMM Computer Communication Review*, 31(2):25–37, April 2001.

[104] Matt Joras and Yang Chi. How Facebook is bringing QUIC to billions, October 2020.

[105] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at QUIC: An approach for rigorous evaluation of rapidly evolving transport protocols. In *Proceedings of the 2017 Internet Measurement Conference*, pages 290–303, London United Kingdom, November 2017. ACM.

[106] Seong-ryong Kang, Xiliang Liu, Min Dai, and Dmitri Loguinov. Packet-Pair Bandwidth Estimation - Stochastic Analysis of a Single Congested Node. *ICNP*, pages 316–325, 2004.

[107] Brian Karrer, Liang Shi, Monica Bhole, Matt Goldman, Tyrone Palmer, Charlie Gelman, Mikael Konutgan, and Feng Sun. Network experimentation at scale. *arXiv:2012.08591 [cs, stat]*, December 2020.

[108] Damian Karwowski, Tomasz Grajek, Krzysztof Klimaszewski, Olgierd Stankiewicz, Jakub Stankowski, and Krzysztof Wegner. 20 Years of Progress in Video Compression – from MPEG-1 to MPEG-H HEVC. General View on the Path of Video Coding Development. In *International Conference on Image Processing and Communications*, volume 525, pages 3–15, October 2017.

[109] David Kastelman and Raghav Ramesh. Switchback Tests and Randomized Experimentation Under Network Effects at DoorDash, February 2018.

[110] Frank P Kelly, Aman K Maulloo, and David K H Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49(3):237–252, 1998.

[111] S Keshav. A control-theoretic approach to flow control. *ACM SIGCOMM Computer Communication Review*, 1995.

[112] Changhoon Kim, Parag Bhide, Ed Doe, Hugh Holbrook, Anoop Ghanwani, Dan Daly, Mukesh Hira, and Bruce Davie. In-band network telemetry (INT), June 2016.

[113] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Nils Pohlmann. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1168–1176, Chicago Illinois USA, August 2013. ACM.

[114] Ron Kohavi, Diane Tang, and Ya Xu. *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing*. Cambridge University Press, first edition, March 2020.

[115] S. Shunmuga Krishnan and Ramesh K. Sitaraman. Video stream quality impacts viewer behavior: Inferring causality using quasi-experimental designs. In *Proceedings of the 2012 Internet Measurement Conference*, IMC '12, pages 211–224, New York, NY, USA, November 2012. Association for Computing Machinery.

[116] Jonathan Kua, Grenville Armitage, and Philip Branch. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming Over HTTP. *IEEE Communications Surveys & Tutorials*, 19(3):1842–1866, 2017.

[117] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. 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*, pages 514–528, Virtual Event USA, July 2020. ACM.

[118] Ike Kunze, Jan Ruth, and Oliver Hohlfeld. Congestion Control in the Wild—Investigating Content Provider Fairness. *IEEE Transactions on Network and Service Management*, 17(2):1224–1238, June 2020.

[119] Raul Landa, Lorenzo Saino, Lennert Buytenhek, and João Taveira Araújo. Staying Alive: Connection Path Reselection at the Edge. In *NSDI 2021*, page 20, April 2021.

[120] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 183–196, New York, NY, USA, August 2017. Association for Computing Machinery.

[121] Wolfram Lautenschlaeger and Andrea Francini. Global synchronization protection for bandwidth sharing TCP flows in high-speed links. In *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–8, July 2015.

[122] Will Law. Clever Monkeys Communicating Discreetly, October 2022.

[123] Changhyun Lee, D. K. Lee, Yung Yi, and Sue Moon. Operating a network link at 100%. In *Proceedings of the 12th International Conference on Passive and Active Measurement*, PAM'11, pages 1–10, Atlanta, GA, March 2011. Springer-Verlag.

[124] Matthew Luckie, Amogh Dhamdhere, David Clark, Bradley Huffaker, and kc claffy. Challenges in Inferring Internet Interdomain Congestion. In *Proceedings of the 2014 Conference on Internet Measurement Conference - IMC '14*, pages 15–22, Vancouver, BC, Canada, 2014. ACM Press.

[125] Charles F. Manski. Identification of treatment response with social interactions. *The Econometrics Journal*, 16(1):S1–S23, 2013.

[126] Ahmed Mansy, Bill Ver Steeg, and Mostafa Ammar. SABRE: A client based technique for mitigating the buffer bloat effect of adaptive video flows. In *Proceedings of the 4th ACM Multimedia Systems Conference on - MMSys '13*, pages 214–225, Oslo, Norway, 2013. ACM Press.

[127] Hongzi Mao, Shannon Chen, Drew Dimmery, Shaun Singh, Drew Blaisdell, Yuandong Tian, Mohammad Alizadeh, and Eytan Bakshy. Real-world Video Adaptation with Reinforcement Learning. *arXiv:2008.12858 [cs]*, August 2020.

[128] Matt Mathis and Andrew McGregor. Buffer Sizing: A Position Paper. In *BS '19: 2019 Workshop on Buffer Sizing*, page 4, December 2019.

[129] Aditya Mavlankar, Liwei Guo, Anush Moorthy, and Anne Aaron. Optimized shot-based encodes for 4K: Now streaming!, August 2020.

[130] Nick McKeown, Guido Appenzeller, and Isaac Keslassy. Sizing router buffers (redux). *ACM SIGCOMM Computer Communication Review*, 49(5):69–74, November 2019.

[131] Tong Meng, Neta Rozen Schiff, P. Brighten Godfrey, and Michael Schapira. PCC Proteus: Scavenger Transport And Beyond. 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*, pages 615–631, Virtual Event USA, July 2020. ACM.

[132] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. TIMELY: RTT-based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 537–550, London United Kingdom, August 2015. ACM.

[133] Robert Morris. TCP behavior with many flows. In *Proceedings 1997 International Conference on Network Protocols*, pages 205–211, October 1997.

[134] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. End-to-end transport for video QoE fairness. In *Proceedings of the ACM Special Interest Group on Data Communication - SIGCOMM '19*, pages 408–423, Beijing, China, 2019. ACM Press.

[135] Netflix. A cooperative approach to content delivery. Technical report, Netflix, 2021.

[136] Whitney K. Newey and Kenneth D. West. A Simple, Positive Semi-Definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix. *Econometrica*, 55(3):703–708, 1987.

[137] Kathleen Nichols and Van Jacobson. Controlling Queue Delay. *ACM Queue*, 10(5):20:20–20:34, May 2012.

[138] Garg Nitin. COPA congestion control for video performance, November 2019.

[139] Samuel D. Oman and Esther Seiden. Switch-back designs. *Biometrika*, 75(1):81–89, March 1988.

[140] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A lightweight control scheme to address the bufferbloat problem. In *2013 IEEE 14th*

International Conference on High Performance Switching and Routing (HPSR), pages 148–155, July 2013.

[141] Larry Peterson, Lawrence Brakmo, and Bruce Davie. *TCP Congestion Control: A Systems Approach*. Self-published, version 1.1-dev edition, 2022.

[142] Meta Platforms. Ax: Adaptive Experimentation Platform, February 2023.

[143] Michele Polese, Federico Chiariotti, Elia Bonetto, Filippo Rigotto, Andrea Zanella, and Michele Zorzi. A Survey on Recent Advances in Transport Layer Protocols. *IEEE Communications Surveys & Tutorials*, 21(4):3584–3608, 2019.

[144] Lili Qiu, Yin Zhang, and Srinivasan Keshav. Understanding the performance of many TCP flows. *Computer Networks*, 37(3-4):277–306, November 2001.

[145] World Snail Racing. World Snail Racing Championships, February 2023.

[146] K. K. Ramakrishnan and Raj Jain. A binary feedback scheme for congestion avoidance in computer networks. *ACM Transactions on Computer Systems (TOCS)*, 8(2):158–181, May 1990.

[147] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, Don Towsley, Chadi Barakat, and Walid Dabbous. Network characteristics of video streaming traffic. In *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, pages 1–12, Tokyo Japan, December 2011. ACM.

[148] James Robins. A new approach to causal inference in mortality studies with a sustained exposure period—application to control of the healthy worker survivor effect. *Mathematical Modelling*, 7(9-12):1393–1512, January 1986.

[149] Donald B Rubin. Causal inference using potential outcomes: Design, modeling, decisions. *Journal of the American Statistical Association*, 100(469):322–331, 2005.

[150] Jan Rüth, Christian Bormann, and Oliver Hohlfeld. Large-scale scanning of TCP's initial window. In *Proceedings of the 2017 Internet Measurement Conference on - IMC '17*, pages 304–310, London, United Kingdom, 2017. ACM Press.

[151] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. Carousel: Scalable Traffic Shaping at End Hosts. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 404–417, Los Angeles CA USA, August 2017. ACM.

[152] Sandvine. Sandivne Global Internet Phenomena Report 2023. Technical report, Sandvine, January 2023.

[153] Yusuf Sani, Andreas Mauthe, and Christopher Edwards. Adaptive Bitrate Selection: A Survey. *IEEE Communications Surveys & Tutorials*, 19(4):2985–3014, 2017.

[154] Kozo Satoda, Hiroshi Yoshida, Hironori Ito, and Kazunori Ozawa. Adaptive video pacing method based on the prediction of stochastic TCP throughput. In *2012 IEEE Global Communications Conference (GLOBECOM)*, pages 1944–1950, December 2012.

[155] Martin Saveski, Jean Pouget-Abadie, Guillaume Saint-Jacques, Weitao Duan, Souvik Ghosh, Ya Xu, and Edoardo M. Airoldi. Detecting Network Effects: Randomizing Over Randomized Experiments. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1027–1035, Halifax NS Canada, August 2017. ACM.

[156] Robert Sayre. Change max-persistent-connections-per-server to 6., March 2008.

[157] Nate Schloss and Ben Maurer. This browser tweak saved 60% of requests to Facebook, January 2017.

[158] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. Towards a Deeper Understanding of TCP BBR Congestion Control. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9, Zurich, Switzerland, May 2018. IEEE.

[159] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 17(9):1103–1120, September 2007.

[160] Jeff Semke, Jamshid Mahdavi, and Matt Mathis. The Rate-Halving Algorithm for TCP Congestion Control.

[161] Anant Shah. BBR evaluation at a large CDN, November 2019.

[162] Steve Souders. Roundup on Parallel Connections, March 2008.

[163] Bruce Spang. Brucespang/tcp_probe, May 2020.

[164] Bruce Spang, Veronica Hannan, Shravya Kunamalla, Te-Yuan Huang, Nick McKeown, and Ramesh Johari. Unbiased experiments in congested networks. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC '21, pages 80–95, New York, NY, USA, November 2021. Association for Computing Machinery.

[165] Bruce Spang and Nick McKeown. On estimating the number of flows. In *BS '19: 2019 Workshop on Buffer Sizing*, page 4, December 2019.

[166] Bruce Spang, Brady Walsh, Te-Yuan Huang, Tom Rusnock, Joe Lawrence, and Nick McKeown. Buffer sizing and Video QoE Measurements at Netflix. In *Proceedings of the 2019 Workshop on Buffer Sizing*, Palo Alto CA USA, December 2019. ACM.

[167] Speedtest. Internet Speed around the world, February 2023.

[168] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15(2s):1–29, April 2019.

[169] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. *IEEE/ACM Transactions on Networking*, 28(4):1698–1711, August 2020.

[170] Jerzy Splawa-Neyman, Dorota M Dabrowska, and TP Speed. On the application of probability theory to agricultural experiments. Essay on principles. Section 9. *Statistical Science*, pages 465–472, 1990.

[171] Rade Stanojevic, Robert N. Shorten, and Christopher M. Kellett. Adaptive Tuning of Drop-Tail Buffers for Reducing Queueing Delays. *ACM SIGCOMM Computer Communication Review*, 37(1), January 2007.

[172] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 272–285, Florianopolis Brazil, August 2016. ACM.

[173] Srikanth Sundaresan, Mark Allman, Amogh Dhamdhere, and kc claffy. TCP congestion signatures. In *Proceedings of the 2017 Internet Measurement Conference*, pages 64–77, London United Kingdom, November 2017. ACM.

[174] Mohit P. Tahiliani, Vishal Misra, and K. K. Ramakrishnan. A Principled Look at the Utility of Feedback in Congestion Control. In *Proceedings of the 2019 Workshop on Buffer Sizing*, pages 1–5, Palo Alto CA USA, December 2019. ACM.

[175] Dave Täht. Bufferbloat on the Internet backbone, November 2017.

[176] Diane Tang, Ashish Agarwal, Deirdre O'Brien, and Mike Meyer. Overlapping Experiment Infrastructure: More, Better, Faster Experimentation. In *KDD'10*, page 10, July 2010.

[177] Eric J. Tchetgen Tchetgen and Tyler J. VanderWeele. On causal inference in the presence of interference. *Statistical Methods in Medical Research*, 21:55–75, 2012.

[178] K. Thompson, G.J. Miller, and R. Wilder. Wide-area Internet traffic patterns and characteristics. *IEEE Network*, 11(6):10–23, November 1997.

[179] Martin Tingley. Streaming Video Experimentation at Netflix: Visualizing Practical and Statistical Significance, September 2018.

[180] Linus Torvalds. Tcp_input.c - Linux (v5.11-rc5), January 2021.

[181] Donald F Towsley. TCP, congestion control, November 2015.

[182] Belma Turkovic, Fernando A. Kuipers, and Steve Uhlig. Fifty Shades of Congestion Control: A Performance and Interactions Evaluation. *arXiv:1903.03852 [cs]*, March 2019.

[183] Belma Turkovic, Fernando A. Kuipers, and Steve Uhlig. Interactions between Congestion Control Algorithms. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*, pages 161–168, June 2019.

[184] Johan Ugander, Brian Karrer, Lars Backstrom, and Jon Kleinberg. Graph cluster randomization: Network exposure to multiple universes. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 329–337, New York, NY, USA, 2013. Association for Computing Machinery.

[185] Curtis Villamizar and Cheng Song. High Performance TCP in ANSNET. *SIGCOMM Comput. Commun. Rev.*, 24(5):45–60, October 1994.

[186] Arun Vishwanath, Vijay Sivaraman, and Marina Thottan. Perspectives on Router Buffer Sizing: Recent Results and Open Problems. *ACM SIGCOMM Computer Communication Review*, 39(2):6, 2009.

[187] G. Vu-Brugier, R. S. Stanojevic, D. J. Leith, and R. N. Shorten. A critique of recently proposed buffer-sizing strategies. *ACM SIGCOMM Computer Communication Review*, 37(1):43, January 2007.

[188] Mei Wang and Yashar Ganjali. Unifying Buffer Sizing Results Through Fairness. Technical Report TR06-HPNG-060606, Stanford University, June 2006.

[189] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. Beyond Jain's Fairness Index: Setting the Bar For The Deployment of Congestion Control Algorithms. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 17–24, Princeton NJ USA, November 2019. ACM.

[190] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. Modeling BBR's Interactions with Loss-Based Congestion Control. In *Proceedings of the Internet Measurement Conference*, pages 137–143, Amsterdam Netherlands, October 2019. ACM.

[191] David X. Wei, Pei Cao, and Steven H. Low. TCP Pacing Revisited. In *INFOCOM*, volume 2, 2006.

[192] Keith Winstein. Why congestion control?, April 2023.

[193] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: A randomized experiment in video streaming. In *NSDI*, page 16, Santa Clara, CA, USA, February 2020.

[194] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 325–338, New York, NY, USA, August 2015. Association for Computing Machinery.

[195] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. An Axiomatic Approach to Congestion Control. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 115–121, Palo Alto CA USA, November 2017. ACM.

[196] Lixia Zhang. *A New Architecture for Packet Switching Network Protocols*. Thesis, Massachusetts Institute of Technology, 1989.

[197] Lixia Zhang and David D Clark. Oscillating behavior of network traffic: A case study simulation. *Internetworking: Research and Experience*, 1:101–112, June 1990.

[198] Lixia Zhang, Scott Shenker, and David D Clark. Observations on the Dynamics of a Congestion Control Algorithm - The Effects of Two-Way Traffic. *SIGCOMM*, pages 133–147, 1991.

[199] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. SENSEI: Aligning Video Streaming Quality with Dynamic User Sensitivity. In *NSDI*, April 2021.

[200] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, pages 313–327, Irvine California USA, December 2016. ACM.