# Understanding the Modern Internet's Heterogeneous Congestion Control Landscape

Ayush Mishra

National University of Singapore
2024

# Understanding the Modern Internet's Heterogeneous Congestion Control Landscape

Ayush Mishra
(B.Tech., NIT Trichy)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2024

Supervisor:
Associate Professor Ben Leong Wing Lup

Examiners:
Professor Chan Mun Choon
Professor Tay Yong Chiang

# DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all sources of information which have been used in the thesis.

This thesis has not been submitted for any degree in any university previously.

Ayush Mishra
September 7, 2024

*"All I know is that I know nothing."*

-Socrates

(about Internet Congestion Control, probably)

# Acknowledgements

This thesis would not have been possible without the kind support of many people. I am grateful to my advisor, Ben Leong. I have the deepest gratitude for his patience and guidance during my PhD. Ben has been a philosophical compass who has always placed special emphasis on applying first principles thinking to not just research, but also life in general. Perhaps the greatest lesson I've learned from him is how much can be achieved by asking his two favorite questions: *Why?* and *Why not?* Our whiteboard brainstorming sessions, which were invariably interspersed with anecdotes and life lessons, will remain the most cherished part of my PhD. I am also thankful to my thesis committee, Prof Chan Mun Choon and Prof Tay Yong Chiang for their invaluable feedback and comments on my thesis.

I would also like to thank all my fellow researchers and collaborators, who have been a pleasure to work with. Thanks to Raj Joshi, Xiangpeng Sun, Atishya Jain, Sameer Pande, Jingzhi Zhang, Melodies Sim, Sean Ng, Sherman Lim, Tiu Wee Han, Vishal Bindal, Lakshay Rastogi, Peirui Cao, Gao Ya, and Archit Bhatnagar. Special mention to Raj, who has been a mentor and a friend since I began doing research during my first research internship. I have learned a lot about the importance of resilience and diligence from working with him for many years now. I am also thankful for my other lab mates and friends at NUS: Zixiao Wang, Oana Barbu, Ashish Dandekar, Nishant Buddhev, Aseem Pahuja, Soundarya Ramesh, Chahwan Song (Mason), Xin Zhe Khooi, and Alen Sabu. They have all been excellent company not just at the lab, but also during many weekend hikes and way too many coffee breaks. A lot of our papers would also not have been possible without the support of the admin staff here at NUS. Special thanks to Iris Chang, who made dealing with our grants, equipment purchases, and hiring so easy. Thanks also to Line Fong and Lee Kheng Goh for their administrative support.

I feel very fortunate to have friends like Akash Warty, Nagaraj Archak, Syamantak Das, Aadithya Vidyasagar, Yash Tewari, John V George, Rachit Rajat, Harsh Bajaj, Eesha Saxena, Akshay Chanda, Dennis James, Abdullah Siddiqui, and Keerthi Chandra - all of whom have brought me a lot of cheer and joy at various times during my PhD. Special mention to the *Westies*, who have invariably been the best company to return to at the end of both good and bad weeks.

Last but not least, I owe a debt of gratitude to my parents and my sister. They have always provided unquestioning and unconditional support for everything I have done. This thesis is dedicated to them.

# Contents

**Abstract**

Research in Internet congestion control has seen a renaissance in the past few years driven by two key developments. In 2016, Google proposed and deployed BBR, a congestion control algorithm that represents a departure from traditional loss-based algorithms like CUBIC and New Reno. Internet transport is also moving to the userspace, with the adoption of QUIC, a new transport stack that is already widely deployed and is set to be the default with HTTP3. While both these developments pose their own unique challenges, they both introduce a large amount of heterogeneity in the Internet's congestion control landscape.

One of the reasons Internet congestion control has largely remained stable in the past is that it has benefited from a homogeneous ecosystem of well-understood AIMD congestion control algorithms. However, the recent advancements in BBR and QUIC will challenge this paradigm. In this thesis, we examine the impact that these two developments will have on the Internet's congestion control landscape, and how they will likely influence its evolution in the near future.

We present a 5-year study into how congestion control on the Internet is evolving in response to the deployment of BBR and QUIC from 2019 to 2023, with snapshots of the composition of congestion control algorithms deployed on the Internet in 2019 and late 2023. Our measurement study revealed that within three short years of being introduced in 2016, BBR was quickly becoming a dominant congestion control algorithm on the Internet and was deployed by 18% of the Alexa Top 20,000 websites. This rapid adoption was likely a result of early adopters of BBR reporting higher and more consistent throughput after switching to it from more traditional congestion control algorithms like CUBIC. To better understand how these performance incentives can drive BBR's adoption on the Internet, we developed a mathematical model that can predict how BBR's throughput advantage over CUBIC will change as more flows on the Internet start adopting BBR. Based on our game theoretic analysis of this model, we make the bold prediction that BBR will never completely replace CUBIC on the Internet. This claim was validated by our 2023 snapshot of the Internet, where we found BBR's share of websites on the Internet to be similar to that in 2019.

We also studied how congestion control algorithms are implemented in QUIC, and how well they conform to the standard implementations of CCAs in the Linux kernel that they try to emulate. Our investigations on this front have revealed that the QUIC ecosystem risks

introducing significant CCA heterogeneity on the Internet because it is easy to modify and implement new CCAs in QUIC. We have found evidence that QUIC stacks deployed by Meta and Google have implementations of BBR and CUBIC that do not conform to their implementations in the Linux kernel. To this end, we propose techniques for identifying such modified CCAs in QUIC stacks and correcting them.

Overall, this thesis reveals that there is significant heterogeneity in the Internet congestion control landscape today. Our results suggest that this heterogeneity will only increase in the future. The novel measurement methodologies proposed in this thesis will allow us to monitor the ongoing evolution of Internet congestion control and inform the future design of modern congestion control algorithms for a heterogeneous Internet.

# List of Publications

1. **Ayush Mishra**, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, Ben Leong, "The Great Internet TCP Congestion Control Census". Proceedings of ACM SIGMETRICS 2020. Boston, Massachusetts, USA. June 2020 [1], (Chapter 3)

2. **Ayush Mishra**, Jingzhi Zhang, Melodies Sim, Sean Ng, Raj Joshi, Ben Leong, "Conjecture: existence of Nash Equilibria in Modern Internet Congestion Control". Proceedings of $5^{th}$ Asia-Pacific Workshop on Networking (APNet 2021). Shenzhen, China. June 2021 [2], (Chapter 4)

3. **Ayush Mishra**, Tiu Wee Han, Ben Leong, "Are we heading towards a BBR-dominant Internet?". Proceedings of ACM IMC 2022. Nice, France. October 2022. [3], (Chapter 4)

4. **Ayush Mishra**, Sherman Lim, Ben Leong, "Understanding Speciation in QUIC Congestion Control". (Short paper) Proceedings of ACM IMC 2022. Nice, France. October 2022. [4], (Chapter 5)

5. **Ayush Mishra**, Ben Leong, "Containing the Cambrian Explosion in QUIC Congestion Control". Proceedings of ACM IMC 2023. Montreal, Canada. October 2023. [5], (Chapter 5)

6. **Ayush Mishra**, Lakshay Rastogi, Raj Joshi, Ben Leong, "Keeping an Eye on Congestion Control in the Wild with Nebby". Proceedings of ACM SIGCOMM 2024. Sydney, Australia. August 2024. [6], (Chapter 6)

# List of Figures

# List of Tables

Chapter 1

# Introduction

The Internet has come a long way since its inception in 1969. From being a modest DARPA networking project with 4 inter-connected computers supporting tens of users, it has evolved into a civil infrastructure with billions of nodes that support more than 5 billion users [12] today. As a result of this evolution, Internet research no longer enjoys the scientific amnesty afforded by other disciplines in computer science. Arguably, critical algorithms and protocols that are deployed on the Internet should be subjected to rigorous evaluation and testing to achieve good performance and prevent another *congestion collapse* [7].

**C**ongestion **C**ontrol **A**lgorithms (or CCAs for short) are one of the many critical algorithms that keep the Internet reliable and stable. CCAs regulate how quickly a sender can send data on the network. As the name suggests, congestion control algorithms keep the network from becoming too congested, thereby avoiding packet loss and unnecessary additional delays. Good congestion control algorithms are essential to web applications to achieve quick response times, faster downloads, and lower latency. Since networks support multiple connections, CCAs must also ensure that all these connections (or *flows*) can equitably share network resources like bandwidth.

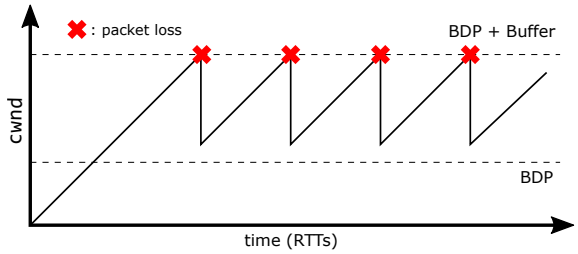



**Figure 1.1:** `cwnd` regulation in classic CCAs like Tahoe and Reno [7]. BDP is the bandwidth-delay product.

## 1.1 Traditional CCA design

The earliest congestion control algorithms, like Tahoe[13] and Reno [7], regulated how quickly they sent data on the network by limiting the number of unacknowledged packets in flight. This limit was called the congestion window, or `cwnd` for short.

Since these algorithms were devised to avoid excessive retransmissions caused by dropped packets, they treated packet loss as a sign of *congestion*. In the absence of congestion (no packet loss) they would probe the network for more bandwidth by additively increasing their `cwnd`. However, when a packet loss occurs, they would infer that the network was congested and halve their `cwnd`. For this reason, these algorithms were often called **A**dditive **I**ncrease **M**ultiplicative **D**ecrease (AIMD) algorithms. We illustrate the behavior of a classic AIMD algorithm in Figure 1.1.

In the past, we have been able to ensure the Internet was stable and predictable because the mix of CCAs that have been deployed on the Internet has largely been a homogeneous mix of loss-based AIMD or MIMD (**M**ultiplicative **I**ncrease **M**ultiplicative **D**ecrease) algorithms [14, 15, 16, 17]. Because they were relatively simple, they were predictable and well-understood [18]. While they differed in how they probed for additional bandwidth, they all responded to the same congestion signal (packet loss).

In addition, because they were deployed on the Internet for so long, they had decades of research and real deployment experience to prove that they were a stable congestion

control solution for the Internet. Therefore, the stability of the Internet has not been a major concern since the first congestion collapse as most people used some flavor of an AIMD congestion control algorithm on the Internet [19, 20, 21]. However, this paradigm is changing quickly today.

## 1.2 Internet Transport is Evolving

Recent developments in Internet congestion control mean that we can no longer expect the Internet to be a homogeneous mix of simple AIMD/MIMD CCAs. In particular, the deployment of the BBR [22] congestion control algorithm in 2016 and the adoption of the QUIC standard [23] in 2021 are forcing us to re-evaluate the stability and predictability guarantees of AIMD algorithms and Internet congestion control in general.

**BBR.** In 2016, Google proposed and deployed BBR [22], a congestion control algorithm with a drastically different congestion control philosophy compared to traditional loss-based AIMD algorithms like CUBIC [16] and New Reno [7]. BBR is largely loss-agnostic and instead modeled the network through bandwidth and **R**ound **T**rip **T**ime (RTT) estimates to operate at the Kleinrock optimal point [8]. The deployment of BBR on the Internet represents a paradigm shift in Internet Congestion Control. Not only does BBR disrupt the ecosystem of traditional AIMD CCAs, but it also interacts with *legacy* algorithms like CUBIC and Reno in often unfair ways [24, 25]. Over the years, there have been many revisions and modifications to the basic BBR algorithm, both by Google [26, 27] and other third parties that often deploy variants of BBR [4]. Today, BBR and its many versions represent a family of model-based congestion control algorithms that regularly interact and compete with the incumbent loss-based CCAs on the Internet. Naturally, how these two contrasting congestion control philosophies interact and co-exist will impact the Internet's stability and fairness.

**QUIC.** Internet transport is also moving to the userspace, with the adoption of

QUIC [23], a new transport stack that is already widely deployed [28] and is set to be the default with HTTP3 [29]. While QUIC represents a change in the network transport stack and not the congestion control algorithms themselves, new implementations of QUIC will re-implement standard congestion control algorithms without any safeguards checking the correctness and stability of these implementations. QUIC also lowers the barrier to the implementation and deployment of new congestion control algorithms because it is implemented in the userspace. It is therefore much easier for someone to deploy untested and objectively unfair CCAs on the Internet.

While both these developments pose their own unique challenges, they both introduce *heterogeneity* in the Internet's congestion control landscape at levels we have never seen before. It is easy to forget that key provisioning decisions are made based on the expected mix of congestion control algorithms on the Internet. The composition of the Internet's congestion control landscape impacts how we size router buffers [30, 31], think about inter-flow fairness [3, 24, 25], and even decide on the deployability of new congestion control algorithms (CCAs) on the Internet [32].

## 1.3 Summary of Thesis Contributions

Overall, these recent developments suggest that we need to rethink Internet congestion control in the future. While it was reasonable to expect that the mix of CCAs deployed on the Internet was homogeneous, slow-moving, and predictable in the past, this is not true anymore. However, before we respond to these rapid changes, we need to understand what changes are happening on the Internet and potentially predict how the Internet will continue to evolve. To this end, we need ways to track how the mix of CCAs deployed on the Internet evolve, understand how these deployed algorithms interact and behave, and how these interactions can be expected to drive their future adoption on the Internet. This thesis represents a first step towards answering these questions.

Overall, we present a 5-year study into understanding how the Internet is evolving in response to the introduction and deployment of BBR and QUIC using measurements and mathematical modeling. More specifically, we make the following contributions:

1. In view of BBR's deployment in 2016, we provide a snapshot of the Internet's congestion control landscape in 2019 (§3). To do these measurements, we developed a new tool called Gordon [11] that was one of the first measurement tools to successfully identify websites on the Internet deploying BBR. Gordon's measurements showed that BBR, just 3 short years since being proposed, was already being deployed by close to 18% of the Alexa Top 20,000 websites on the Internet. Since most of these websites streamed video or delivered large files, BBR's share of downstream traffic was estimated to be even larger, more than 40%.

2. We evaluate if BBR's rapid adoption meant that we can expect it to replace legacy CCAs like CUBIC and Reno in the future (§4). To this end, we present a mathematical model for competing CUBIC and BBR flows and show that BBR's performance benefits are going to diminish with its increased adoption on the Internet. On the basis of our model, empirical results, and game theoretic analysis, we make a bold prediction that BBR is unlikely to completely replace CUBIC and the Internet will remain a heterogeneous mix of CCAs for the foreseeable future.

3. We also investigate how CCAs are implemented in popular QUIC stacks and how their behavior can vary across implementations (§5). To this end, we built QUICbench [33], a benchmarking tool for open-source QUIC stacks that can measure how conformant a QUIC CCA implementation is to its Linux kernel counterpart. QUICbench assigns CCA implementations this conformity score based on a new metric we call the *Performance Envelope*. Using QUICbench we show how popular QUIC stacks, like those deployed by Meta and Google, often modify standard CCAs like BBR and CUBIC and risk introducing even more heterogeneity

on the Internet.

4. Given the large degree of current and future heterogeneity we expect to persist on the Internet, we propose Nebby (§6), a future-proof CCA classification and identification tool that works not only with TCP but is also one of the first CCA measurement tools that can work with QUIC and live browser traffic. Nebby addresses the shortcomings of the CCA measurement tools that came before it (including Gordon, which as of 2024 gets blocked by most websites) by aiming to to be as passive as possible. We use Nebby to capture a snapshot of the mix of CCAs on the Internet in 2023. In addition to catching the deployment of then unannounced variants like BBRv3, we also observed stagnation in BBR's adoption on the Internet since 2019. This is in line with our prediction (made in §4) that BBR's performance benefits will stagnate and it will never completely replace CUBIC on the Internet. We look at these results in full context with our measurements from 2019 in §7 as well.

We provide a more detailed overview of these contributions below.

### 1.3.1 Understanding the Internet's evolving heterogeneous CCA landscape

In this thesis, we present two snapshots of the Internet's congestion control landscape. The first of these snapshots was captured in 2019 via an Internet-scale measurement study [1] of the Alexa Top 20,000 websites on the Internet. This measurement study was done in response to the deployment of BBR in 2016. We wanted to understand how commonly BBR was deployed on the Internet.

To this end, we designed and implemented *Gordon*, a tool that allows us to measure the exact congestion window (`cwnd`) corresponding to each successive RTT in the TCP connection response of a congestion control algorithm. To compare a measured flow to

known CCA variants in the Linux kernel, we created a localized bottleneck where we can introduce a variety of network changes like loss events, bandwidth change, and increased delay. An offline classifier was used to identify the TCP variant based on its `cwnd` trace over time.

Our measurements revealed that in 2019, CUBIC remained the dominant CCA on the Internet and was deployed on about 36% of the websites in the Alexa Top 20,000 list. However, in just three short years since it was first proposed, BBR was in second place and already deployed by 22% of the measured websites. That said, its total share of traffic volume was likely to be disproportionately larger (40%) since most websites deploying BBR hosted bandwidth-intensive content like videos and large files. We also found many Akamai-hosted websites deploying an undocumented CCA, that we referred to as *AkamaiCC*. This challenges the conventional assumption that CCAs deployed on the Internet would come from a known set of CCAs.

In late 2023, we repeated this measurement study to understand how the mix of CCAs on the Internet had evolved since our last measurement study in 2019. Because Gordon no longer works (see §6.1.1), this second snapshot was captured using a much more sophisticated tool called Nebby that was capable of classifying CCAs over not just simple `wget` TCP connections, but also over live browser sessions and QUIC connections. This second measurement study not only confirmed the stagnation in BBR's adoption as predicted by our mathematical model in 2022 [3], but also revealed some interesting trends in the preference of BBR over CUBIC depending on region and content type. Nebby was also able to capture an early deployment of BBRv3 [27] before it was formally announced by Google and catch the latest evolution of *AkamaiCC*, which was no longer limited to Akamai-hosted websites.

### 1.3.2 Nash Equilibria in Internet Congestion Control

When we observed an unprecedented adoption of BBRv1 on the Internet in 2019, just three short years since it was first proposed, we could not help but wonder if BBR was poised to replace CUBIC and other *legacy* congestion control algorithms on the Internet. Most of BBR's early deployments seemed to be driven by the fact that it provided better throughput than CUBIC on the Internet. This was confirmed by early deployment results published by Google [34], Dropbox [35], and Spotify [36]. To investigate the performance benefits that could drive BBR's adoption on the Internet, we studied the interactions between CUBIC and BBR and demonstrated that these two CCAs competing for bandwidth can be modeled as a *normal-form* game. Our game-theoretic analysis, mathematical model, and testbed measurements suggested that while BBR seems to achieve somewhat better performance than CUBIC on the Internet today, this advantage was likely to decrease as more and more people on the Internet adopted BBR.

Therefore, we predicted that the distribution of congestion control algorithms on the Internet would likely reach a Nash Equilibrium, where no flow has the incentive to switch from CUBIC to BBR, or vice versa. We also found that the distribution of CUBIC and BBR flows in this Nash Equilibrium will be dependent mainly on the size of the bottleneck buffer, and marginally on the RTT distribution of the flows. All these results suggested that the future Internet was likely to continue to be heterogeneous and that buffer sizing would continue to have a significant impact on Internet congestion control.

Our methodology is also applicable to other recently proposed congestion control algorithms, like BBRv2 [26] and PCC Vivace [37]. Given the results of this study, we made a bold prediction that BBR is unlikely to completely replace CUBIC on the Internet for the foreseeable future. Our measurements from late 2023 indicate that this is indeed the case, with BBR's share on the Internet remaining stagnant since our last

measurement study in 2019.

### 1.3.3 Speciation in QUIC Congestion Control

Besides BBR, QUIC is also a significant development for Internet Congestion Control. The QUIC standard is expected to replace TCP in HTTP 3.0 [38]. In 2022, more than 25 open-source QUIC stacks were already being tested, with a large number of them being deployed on the Internet by major players like Google and Meta.

While QUIC implements a number of the standard features of TCP differently, most QUIC stacks re-implement standard congestion control algorithms. This is because these algorithms are well-understood and time-tested. However, there is currently no systematic way to ensure that these QUIC congestion control protocols are implemented correctly and predict how these different QUIC implementations will interact with other congestion control algorithms on the Internet.

To address this gap, we developed QUICbench [33], a benchmarking tool to allow QUIC developers to verify the correctness of their CCA implementations. QUICbench recognizes that since each CCA represents a trade-off between network metrics like throughput and delay, each CCA implementation should be characterized and then tested for conformance using a metric that captures this trade-off. To this end, we presented a new metric called the *Performance Envelope*, that captured the throughput-delay trade-off space a CCA implementation exists in. QUICbench then uses novel metrics like *conformance* and *conformance-T* (which are used to compare the performance envelopes of different implementations) to determine if a CCA implementation has been incorrectly tuned or implemented and even suggest corrections.

We used QUICbench to measure the CCAs implemented by 11 open-source QUIC stacks following the IETF QUIC standard and compared them to their respective reference implementations of CUBIC, Reno, and BBR in the Linux kernel. Our investigations revealed that there is already a significant deviation between the existing QUIC

implementations of standard congestion control algorithms from the reference implementations. QUICbench has been instrumental in revealing these implementation and stack-level differences. Overall, we have found 7 CCA implementations of CUBIC, Reno, and BBR that do not conform to their kernel counterparts. QUICbench has been able to provide hints on how we can modify four of these variants to be more conformant.

Overall, our investigations of the CCA implementations in existing open-sourced QUIC stacks has revealed that they need to be closely monitored and tested for conformance, and risk introducing additional CCA heterogeneity on the Internet. Perhaps more alarmingly, they can even undermine our understanding on how we expect prominent congestion control algorithms like CUBIC and BBR to interact because their behavior can now no longer be expected to be faithfully replicated by all of their implementations.

## 1.4 Organization

The thesis is organized as follows. We first review related work in Chapter 2. We then present the results of our first Internet-scale CCA measurement study (titled *The Great Internet TCP Congestion Control Census*) in Chapter 3. We investigated the question of whether BBR's unprecedented adoption would likely result in a BBR-dominated Internet congestion control landscape in Chapter 4. We also examine QUIC's impact on general CCA heterogeneity and propose metrics to verify the correctness of CCA implementations in QUIC stacks in Chapter 5. As the final concluding work, we propose a new future-proof CCA identification tool called Nebby in Chapter 6 and use it to produce a more recent snapshot of the Internet's congestion control landscape. Finally, we conclude by discussing the implications of our findings and future work in Chapter 7.

Chapter 2

# Background and Related Work

Studying the mix of congestion control algorithms on the Internet and mathematically modeling how they interact has been of interest to the networking community for as long as the Internet has existed. For example, the first Internet-scale measurement study for identifying congestion control algorithms in the wild was done more than two decades ago in 2001 [39]. This is expected because the performance of congestion control algorithms is highly contextual and depends on the other congestion control algorithms they compete with. Furthermore, the mix of CCAs on the Internet dictates how we decide to provision router buffers [40] and ensure fairness and stability on the Internet [41].

As the Internet continues to evolve, so have these measurement and modeling techniques. However, as we will discuss in this chapter, most of these early techniques were very rudimentary because they did not anticipate the current levels of heterogeneity on the Internet. As discussed in §1, our goal in this thesis is to uncover the current heterogeneity of CCAs on the Internet and to predict how the deployment of BBR and QUIC will drive its evolution in the future. In particular, we wanted to

1. Identify congestion control algorithms deployed by web servers in the wild

2. Understand how CUBIC and BBR, the two most dominant CCAs on the Internet

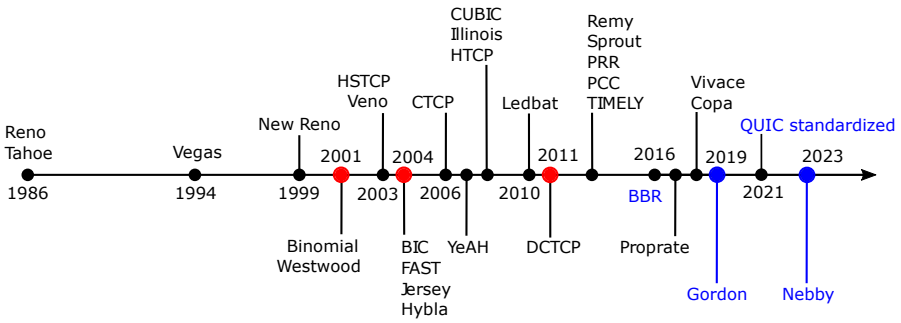



**Figure 2.1:** The evolution of Internet transport. Previous snapshots of Internet's CCA landscape are marked in red, and the ones presented in this thesis are marked in blue.

today, interact

3. Study how congestion control is implemented in QUIC

In this chapter, we will discuss why existing methods fall short of achieving these goals and how our novel measurement and modeling techniques can help researchers get a handle on the modern heterogeneous congestion control landscape of the Internet.

## 2.1 Identifying Congestion Control Algorithms in the wild

Identifying congestion control algorithms in the wild is not a new problem. To the best of our knowledge, before our efforts in 2019, there have been four prior studies attempting to characterize TCP congestion control variants deployed in the wild. We summarize the evolution of Internet congestion control along with these previous measurement studies and our own two measurement studies in Figure 2.1.

The general approach to identifying congestion control algorithms (CCAs) in the wild has largely remained unchanged over the years. It involves making a connection with a remote web server under very specific network conditions (some emulated delay, bandwidth, and/or packet drops), and then observing the server's sending behavior in response to these network conditions. If this response is measured using the right metric and the network conditions are chosen such that they elicit a unique response from every

CCA, then it becomes possible to classify the CCA run by the remote server. Over the years, as the CCAs deployed in the wild have become more diverse and sophisticated, the techniques used for CCA classification on the Internet have evolved as well.

In 2001, Padhye et al. [19] used a tool called TBIT that performed a specialized 25-packet drop and accept pattern, which allowed it to detect if a web server was running one of the four target congestion control variants: Reno, New Reno, Reno Plus and Tahoe. Because its target set of congestion control algorithms was relatively small, TBIT could get away with having a very ad hoc classification strategy that only worked for these four CCAs. At the time of publication, the consensus was that Reno was the most widely deployed variant. However, their results showed that most of the Internet was already using New Reno. In 2004, Medina et al. [20] followed up on the work by Padhye et al. by using TBIT to perform active and passive measurements of more than 84,000 hosts on the Internet. Although they were only able to classify 33% of their target hosts, the categorized hosts showed a continued trend of moving from Reno to New Reno, as previously observed by Padhye et al. [19].

However, by 2011, TBIT was already reaching the limits of its capabilities. As we can see in Figure 2.1, there was an explosion of new CCAs since TBIT was first designed in 2001. Since TBIT was ad hoc and was not designed to identify any CCAs outside of variants of Reno and Tahoe, it would have been blind to the deployment of all these new CCA variants. The most significant of these variants were CUBIC [16] and Compound TCP (CTCP), which were in the process of becoming the default congestion control algorithms for Linux and Windows operating systems, respectively. Therefore, in 2011, Yang et al. [21] had to build a new CCA classification tool for their measurement study, which they called CAAI. This study was also the last update on the distribution of congestion control variants on the Internet prior to our measurement study in 2019. CAAI classifies TCP variants on the Internet using `cwnd` traces collected via two distinct network profiles. It extracts feature vectors from these `cwnd` measurements and identifies

them through a classifier trained on `cwnd` traces from controlled servers in a local testbed. CAAI uses delayed ACKs to bloat the RTT in an attempt to 'space out' the individual `cwnd`s in a connection. This technique was sufficient for classifying the suite of window-based CCAs that were popular on the Internet in 2011. However, as we will cover in §3, just like TBIT, CAAI's measurement methodology also became ineffective in the face of BBR's deployment in 2016. Nevertheless, in 2011, their measurements showed that BIC, CUBIC, and Compound TCP (CTCP) together had become more popular than New Reno. Separately, Yang et al. also identified delay-based variants such as YeAH [42], Vegas [43], Veno [44] and Illinois [45] [46]. They found that about 4% of the Internet hosts tested used these delay-based congestion control variants. In 2020, Gong et al. [47] conducted another Internet-scale CCA measurement study using their tool Inspector Gadget (IG). IG largely followed CAAI's measurement methodology with a rudimentary classifier for BBR. However, as we will cover in §6, Gordon, IG, and all the other CCA classification tools before them are not future-proof or client-agnostic. For example, none of them can work with QUIC connections or live browser traffic. Nebby does not have any of these limitations.

Outside of these Internet-scale CCA measurement studies that are directly comparable to our own measurement studies conducted in 2019 (using Gordon, §3) and 2023 (using Nebby, §6), there have also been some work on TCP-related measurements that focus on evaluating congestion control algorithms and their implementations.

Chen et al. used deep neural networks to analyze passive measurements taken from TCP receivers and identify the congestion control variant used by a TCP sender [48]. They used traces of long continuous flows to train a Long Short Term Memory (LSTM) neural network that classifies the trace behaviors into the congestion control variants by using features such as RTT, packets in flight and throughput. Their evaluation was done only on a controlled testbed, and so it is not surprising that neural networks can classify relatively well behaved traces. Because evaluation was not performed on actual

Internet hosts, no attempts were made to address the noise from packet losses on the Internet. We have reason to believe that such noise would introduce significant errors.

Hagos et al. used machine learning to infer the state of a TCP sender [49]. Comer et al. used active probing techniques to reveal implementation flaws, protocol violations, and design decisions of the 5 commercial black box congestion control implementations [50]. Sun et al. [51] and Lubben et al. [52] also evaluated the correctness of TCP implementations in controlled testing environments. None of these are directly applicable for identifying TCP variants on the Internet.

### 2.1.1 Why existing techniques fail

A common thread that connects all the CCA classification tools before Gordon and Nebby is a lack of generality and future-proofness. All these tools did not anticipate how heterogeneous the Internet's congestion control landscape can be. They aimed to identify a small known set of CCAs on the Internet, which resulted in the design of ad hoc classification techniques that were often very tightly coupled with their `cwnd` measurement methodologies. Case in point, even though TBIT [19, 20], CAAI [21], and Inspector Gadget [47] all use `cwnd` traces to identify CCAs in the wild, they do not have classifiers that can work interchangeably. This limited these tools from being adapted to newer congestion control algorithms that were deployed on the Internet after these measurement studies were conducted.

Because of this limitation, both our measurement tools, Gordon and Nebby, decouple the measurement methodology from the classifier so that they can accommodate more sophisticated classifiers in the future. We elect to collect traces over standard network profiles that not only sufficiently capture the characteristic behaviors of our target set of CCAs, but also help us get insights into the congestion control logic of any unknown CCAs we might encounter on the Internet. Our classifier is also designed keeping in mind that we might encounter unknown and undocumented variants on the Internet. It

is for this reason we elect not to use machine learning to build our classifier. This is because while supervised learning approaches have been reasonably successful in identifying known TCP variants, they will not be able to uncover new undocumented variants that are surprisingly common. This issue carries over to other CCA-identification tools that were proposed since Gordon as well, like Inspector Gadget [47]. For example, even though both Gordon and Inspector Gadget performed their measurements in 2019, Gordon was able to catch the deployment of a proprietary CCA by Akamai [1] while Inspector Gadget could not do so.

Another reason why we can't use previous measurement tools for classifying congestion control algorithms on the Internet anymore is the proliferation of rate-based CCAs on the Internet since BBR was introduced in 2016. All previous measurement studies used `cwnd` measurements to classify congestion control algorithms. This worked well on the pre-BBR Internet since all the target congestion control algorithms were window-based. The `cwnd` was therefore a true representation of their response to different network conditions and therefore sufficient for differentiating them from other CCAs. This is not true for rate-based CCAs. Rate-based CCAs typically use the `cwnd` as a safeguard and not an operating point. Because of this, we run the risk of not actually capturing a rate-based CCA's behavior in response to different network conditions if we measure the `cwnd`. In fact, our first measurement tool, Gordon, also suffers from this limitation and had to design an ad-hoc classifier for BBR. Learning from these limitations our latest CCA identification tool, Nebby, measures the bytes in flight (`BiF`) of a remote sender in order to reliably capture its response to different network conditions.

In summary, previous measurement tools do not work well because they were solutions that were very specific to their target set of CCAs, and not general enough to be future proof. We discuss how both our measurement tools, Gordon and Nebby, address these limitations in §3 and §6 respectively.

## 2.2 Studying the interactions between CUBIC and BBR

Unsurprisingly, BBR's unprecedented and rapid adoption on the Internet since its proposal in 2016 has inspired many studies into studying its general performance over a variety of network conditions, its fairness properties, and how it interacts with traditional congestion control algorithms [24, 25, 53, 54]. In this section, we will first describe BBR's model-based approach to congestion control and how it differs from traditional loss-based and window-based CCAs. We will then cover related works that have looked into BBR's performance and how it interacts with traditional congestion control algorithms like CUBIC.

### 2.2.1 A Primer on BBR

First proposed in 2016, BBR [22] is a model-based congestion control algorithm that tries to model the network path by estimating the bottleneck bandwidth and the minimum RTT. It estimates the bottleneck bandwidth via the receive rate (the rate at which it receives ACK packets) and tries to match it to minimize queuing at the bottleneck. It does so to reduce the additional queuing delay at the bottleneck queue while matching the bottleneck rate to ensure utilization. Therefore, in theory, BBR aims to operate at the Kleinrock optimal point [8] (see Figure 2.2). In practice, since the optimal point is likely to be a moving target, BBR periodically probes the network to get updated estimates on the network path's bottleneck bandwidth ($C$) and minimum RTT ($RTT_{min}$). It probes for spare bandwidth by periodically increasing it's sending rate 1.25 times the receive rate every 8 RTTs. It then immediately reduces the rate to 0.75 times the receive rate in order to drain any queuing the previous bandwidth probing cycle might have created. In order to estimate the minimum RTT, all BBR flows collaboratively reduce their inflight to a handful of packets every 10 seconds to completely drain the bottleneck buffer.

While the model-based approach works well when all the flows at the bottleneck are

BBR flows, BBR's estimates can often go off-kilter when competing with a traditional loss-based congestion control algorithm like CUBIC. We will explore how these interactions govern how CUBIC and BBR flows share the bottleneck link in more detail in §4 - but at a high level, BBR flows and CUBIC flows do not interact well because of two critical differences:

1. **Congestion Inference.** While CCAs like CUBIC infer network congestion via a packet loss, BBR flows are completely loss-agnostic. In the event of a packet loss while a CUBIC flow would back off and reduce its `cwnd`, BBR will be completely unaffected. BBR's sending behavior is regulated only by what its internal model thinks the network looks like. Therefore, since CUBIC and BBR respond differently to packet loss, the lossy-ness of a network and the bottleneck buffer size can often decide how they share the bottleneck link.

2. **Operating Points.** A corollary of CUBIC only using packet-loss as a congestion signal is that it typically likes to fill the entire bottleneck buffer. For this reason, loss-based CCAs are often called buffer fillers, because they only slow down once they see a packet loss due to buffer overflow. Therefore, while BBR aims to operate at the Kleinrock optimal point, CUBIC usually keeps the buffer filled (see Figure 2.2).

### 2.2.2 BBR's interactions with CUBIC

Hock et al. conducted the first independent study into understanding how BBR interacts with CUBIC flows [24]. They observed that in shallow buffers of less than 1BDP, BBR flows take a larger share of the bandwidth compared to competing CUBIC flows. They argued that BBR's bandwidth probing causes buffer overflows and bursty losses for competing CUBIC flows. The resulting packet loss causes CUBIC to reduce its `cwnd`, which in turn allows BBR to take a larger share of the bandwidth. This cycle is perpetuated

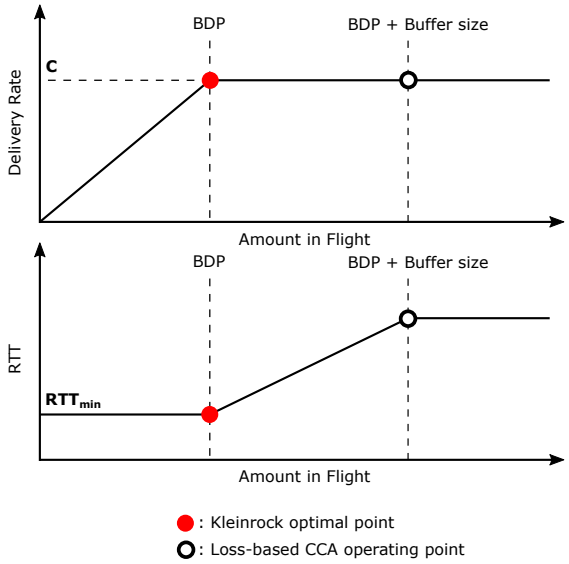



**Figure 2.2:** Kleinrock optimal point [8] (C is the link capacity).

with every bandwidth probe, leading to CUBIC starving for bandwidth. This observation is corroborated by other studies [53, 54]. Scholz et al. conducted experiments with up to 10 BBR flows competing with 10 CUBIC flows [53] and showed that BBR flows are always able to claim at least 35% of the total bandwidth. Dong et al. also made a similar observation that when a single BBR flow competes with an ever-increasing number of CUBIC flows, BBR's fraction of the bandwidth remains the same [55]. While these early results were interesting, they were ultimately empirical and did not provide any insights into how BBR can be expected to compete with CUBIC in any given network.

To the best of our knowledge, the best state-of-the-art model for the interactions between CUBIC and BBR before our work was published in 2022 was the model by Ware et al. [25]. Ware et al. demonstrated that BBR's performance is governed by its `cwnd` in deep buffers. They claimed that for *very deep* buffers, BBR flows collectively take up a fixed share of the bottleneck buffer. Unfortunately, some of the assumptions their model makes are not applicable for a lot of networks and hence it is not accurate over a large range of buffer sizes. One of its key shortcomings is that it assumes that bottleneck

buffers are always full. Our model, which is discussed in more detail in Chapter 4 makes none of these assumptions and is a lot more accurate as a result.

Since we proposed our mathematical model, there have also been proposals for using fluid models to predict how CUBIC and BBR interact. Scherrer et al. [56] used a fluid model to analyze transient oscillations that can happen when CUBIC and BBR compete in a bottleneck buffer and what impact that can have on bandwidth fairness between them. While their model was more accurate than our traditional steady-state model (see §4), they do not produce any interpretable closed-form equations. They therefore do not reveal any more insights into how CUBIC and BBR interact than network simulators and test bed experiments.

### 2.2.3 Predicting BBR's future adoption on the Internet

To the best of our knowledge, while there have been several previous works studying how CUBIC and BBR interact, none of them have applied their insights to predicting how their differences can drive their future adoption on the Internet. In this respect, we are the first to apply game theory to predict how BBR's performance advantage over CUBIC will evolve - and how this will in turn drive its adoption on the Internet.

That said, game theory has been previously applied to congestion control [57, 58, 59], albeit in different settings and contexts. Chien and Sinclair were the first to study the interactions between modified AIMD congestion control algorithms and evaluate the efficiency of the Nash Equilibrium bandwidth distributions between them [58]. They showed that the Nash Equilibrium between Reno and Tahoe flows can be efficient in drop-tail buffers and inefficient with RED-enabled buffers. The main difference between their work and ours is that their strategies (congestion control algorithms) for the players (individual flows) are fixed. Chien and Sinclair attempted to calculate the Nash Equilibrium *bandwidth distribution*, while we are focused on predicting the Nash Equilibria in terms of the distribution of congestion control algorithms. In the remaining

two works [57, 59], the focus was on investigating the Nash Equilibrium bandwidth distributions between two flows running Reno and Vegas.

Overall, our work in studying the interactions between CUBIC and BBR using a mathematical model and predicting how it could drive BBR's adoption on the Internet in the future addresses a research problem that has not been looked into before. While there have been other mathematical models describing CUBIC and BBR, the state-of-the-art steady-state model before us [25] had limited accuracy. For this reason, we had to develop our own steady-state model that had improved accuracy. Because no one had applied game theory to predict CCA adoption on the Internet before, we also had to define our own notion of a *Nash equilibrium* distribution of CCAs on the Internet. We describe these contributions in more detail in §4.

## 2.3 QUIC

The widespread adoption of QUIC by major tech companies has inspired numerous studies in recent years [60, 61, 62, 63, 64]. These works include studies on the effectiveness of its new mechanisms, interoperability, differences in parameterization, and general performance.

**Comparing gQUIC with TCP.** Most of the earlier studies evaluating the performance of QUIC against TCP were done using gQUIC[60], Google's original version of QUIC before the IETF QUIC standard was released, as it was the only implementation available. Langley et al. completed an extensive study on gQUIC's performance for Google's large-scale deployment of QUIC and reported gQUIC's performance against TCP CUBIC [60]. Langley et al. reported that gQUIC outperformed TCP in metrics such as Google Search's latency, YouTube's video latency, and YouTube's video rebuffer rate. For example, YouTube's video rebuffer rate was reduced by 18.0% for desktop users and 15.3% for mobile users when TCP was replaced with gQUIC. A limitation of this

study is that it measured the performance of specific applications (and not QUIC CCA implementations) and the results were obtained from proprietary data that is not easily reproducible. In our work, we study the general transport-layer performance of CCA implementations for a large number of QUIC stacks and not just gQUIC. Our source code is available at [33] and our experiments are fully reproducible.

Other gQUIC studies evaluating gQUIC's performance against TCP mostly used page-load time measured via controlled experiments as the main metric to compare their application-layer performance [61, 62, 63]. Carlucci et al. found that gQUIC achieved higher goodput and smaller page-load times in networks with small buffers or high packet loss rates [61]. Biswal et al. found that gQUIC outperformed TCP in networks with low bandwidth, high RTT, or high packet loss rates. Megyesi et al. had similar conclusions, except that they reported, to the contrary, that TCP performs better in networks with high packet loss rates. These studies all evaluated the QUIC's protocol performance against TCP using application-level metrics without any attempt at root-cause analysis. Our work is focused on the transport-layer performance of the QUIC CCAs and we show that our methodology provides hints that allow us to (i) deduce potential reasons for differences in behavior, and (ii) to verify if modifications made would make a QUIC CCA implementation more conformant to the reference kernel implementation.

Another study on gQUIC by Kakhki et al. focused on evaluating gQUIC's transport-layer performance against TCP [64]. In their study, Kakhki et al. performed a root-cause analysis using execution traces captured by instrumenting gQUIC's source code. They discovered that gQUIC's default parameter values were not tuned and proceeded to calibrate their gQUIC implementation in their experiments so that they would perform similarly to those deployed in production. Kakhki et al. highlighted that this calibration was not done in prior studies and led to them wrongly concluding that gQUIC would under-perform in high bandwidth networks.

Kakhki et al. observed that the congestion control implementation in QUIC was likely

different from TCP despite both of them implementing the same CUBIC CCA because the gQUIC flow was observed to have twice the bandwidth of a competing CUBIC flow at the same bottleneck link. They concluded that this is because gQUIC's CUBIC increased its congestion window (`cwnd`) more frequently and by a larger amount than TCP CUBIC. Although Kakhki et al. showed that there was a deviation and even found the root cause, those findings are not relevant today as the QUIC standard published by IETF (IETF QUIC) has many significant differences compared to the old gQUIC standard. Moreover, the QUIC ecosystem has grown much larger and includes many more stacks. At some level, this limitation also extends to the results of the above studies. To the best of our knowledge, our work is the most comprehensive measurement study covering the largest number of open-sourced QUIC implementations to date.

**Evaluating IETF QUIC implementations.** There are other more recent studies that evaluated IETF QUIC stacks, but most of them have limited scope, and focus on application-layer metrics [65, 66, 67]. Saif et al. compared the `quiche` stack against TCP and found that `quiche` QUIC has greater average throughput but worse user quality of experience metrics as measured by the Lighthouse tool [67, 68]. A key limitation of these studies is that they only report the results for a single QUIC stack. As demonstrated in our latest study, QUIC's performance can differ significantly across implementations and these differences are often artifacts of the implementations and not a result of the QUIC protocol.

### 2.3.1 Speciation in QUIC Congestion Control

To the best of our knowledge, Marx et al. were the first to report speciation in different QUIC stacks [69]. Their study highlighted differences in implementation details of 15 IETF QUIC stacks. These differences were uncovered by analyzing the inner workings of the QUIC stacks through visualizations produced by the Qvis tool. They found significant differences in domains where the QUIC standard had minimal specifications,

i.e. congestion control and flow control. Differences were found even for parameters that were specified in the QUIC standard. For example, 3 of the stacks did not follow the QUIC standard's initial congestion window value specification and 10 of the stacks did not follow the QUIC standard's recommendation of 2 for the acknowledgment frequency. Marx et al. did not investigate how the performance differences arose from these implementation deviations. More recently, we had earlier reported significant deviations between the different QUIC implementations of existing congestion control algorithms and their respective kernel implementations for 4 common QUIC stacks [4].

Overall, bulk of the literature that benchmarks and tests QUIC stacks generally concentrates on the performance of the transport stack as a whole. In general, because most stacks re-implement standard congestion control algorithms, congestion control in QUIC was a largely neglected area before our work in 2022 [4]. In §5, we discuss in detail about how this is a significant research gap and propose ways to detect and correct for speciation in QUIC implementations of standard congestion control algorithms.

Chapter 3

# The Great Internet TCP Congestion Control Census

Given how big a departure BBR represents from the design of traditional loss-based CCAs, we wanted to take stock of the Internet's congestion control landscape 3 years since its first deployment in 2016. Since the last measurement study before ours was conducted before BBR was deployed on the Internet, past measurement methodologies do not work well for identifying non-window-based CCAs like BBR on the Internet. To this end, we had to develop a new measurement tool, called Gordon, to measure and classify the CCAs run by the Alexa Top 20,000 websites on the Internet. In this chapter, we present the motivation behind this measurement study, Gordon's design, and a snapshot of the Internet's CCA landscape in 2019.

## 3.1 Background

Over the past 40 years, TCP congestion control has evolved to adapt to the changing needs of the users and to exploit improvements in the underlying network. Most recently, in 2016, Google proposed and deployed a new TCP variant called BBR [22] (**B**ottleneck

**B**andwidth and **R**ound-trip propagation time). BBR represents a major departure from traditional congestion-window-based congestion control. Instead of using packet loss as a congestion signal, BBR uses estimates of the bandwidth and round-trip delays to regulate its sending rate. BBR has since been introduced in the Linux kernel [70] and deployed by Google across its data centers. As the TCP ecosystem has changed significantly since the last study [46] done in 2011, it is timely to conduct a new census to understand the latest distribution of TCP variants on the Internet.

The goals of our TCP census are relatively modest. We aim to (i) understand how the distribution of previously identified variants has changed since 2011; (ii) develop a method to identify BBR in existing websites; and (iii) determine the proportion of undocumented TCP variants if any. The final goal of our approach represents a significant departure from previous studies, which assumed a fixed set of known TCP variants and attempted to classify all the measured websites as one of the known variants.

To this end, we designed *Gordon*, a tool that allows us to measure the exact congestion window (`cwnd`) corresponding to each successive RTT in the TCP connection response of a congestion control algorithm "in the wild." While rate-based TCP variants do not maintain a congestion window, they typically maintain a maximum allowable number of packets in flight [22], which we can measure as the *effective* congestion window for each RTT. To compare this response to that of known variants, we created a localized bottleneck where we introduced a variety of network changes: loss events, bandwidth change, and increased delay. We also normalize all measurements by RTT. An offline classifier is then used to identify the TCP variant based on the `cwnd` trace over time. By decoupling measurement from classification unlike prior studies [19, 20, 21], our approach allows us to not only identify known TCP variants but also detect new undocumented variants. Our approach also makes it possible to improve the accuracy of the classifier without repeating the relatively expensive measurements, if new network profiles are not required for the improved classifier.

We used Gordon to measure the 20,000 most popular websites according to the Alexa rankings [71]. The following are our key findings:

1. Our results suggest that, as expected, CUBIC is currently the dominant TCP variant on the Internet and is deployed at 36% of all the classified websites, which is an increase from what was reported in the last study in 2011 (§3.3.5).

2. The rate of BBR adoption over the past 3 years since its release has been phenomenal. BBR (together with its Google variant) is currently the second most popular TCP variant deployed at 22% of the classified websites (§3.3.5).

3. While BBR has a share of only 22% by website count, we estimate that its present share of total Internet traffic volume already exceeds 40%. This proportion will almost certainly exceed 50% if Netflix and Akamai also decide to adopt BBR (§3.3.3).

4. The assumption that TCP variants "in the wild" will come from a known set is not true anymore. In particular, we found that Akamai has deployed a unique loss-agnostic rate-based TCP variant on some 6% of the Alexa Top 20,000 websites (§3.3.4).

Since our key design principle is to look for generic characteristics such as reaction to bandwidth change, delay and different types of loss, Gordon can be extended to identify new future variants that are not known today. Given that we expect the TCP congestion control landscape to undergo rapid and significant change soon, we do not think that the previous approach of taking a snapshot every 10 years is good enough. We are in the process of enhancing and automating Gordon into a web-service that can capture a continuous view of the Internet's ongoing transition to a new era of rate-based congestion control. We hope that the current shift in congestion control philosophy and our work in uncovering new undocumented rate-based variants would draw attention towards studying the interaction between `cwnd`-based and rate-based protocols at scale.

The rest of the chapter is organized as follows: In §3.2, we describe the design and implementation of Gordon's measurement and classification techniques. In §3.3, we first evaluate the measurement accuracy of Gordon and then present detailed results of using Gordon to identify TCP variants for the Alexa Top 20,000 websites [71] on the Internet. In §3.4, we describe the practical difficulties we faced, the current limitations of Gordon, and future directions to improve Gordon for understanding the long-term evolution of Internet congestion control. Finally, we conclude in §3.5.

## 3.2 Methodology

Gordon emulates a local bottleneck and tracks the evolution of the effective congestion window (`cwnd`) (see §3.2.1) of the probed TCP variant while changing the available bandwidth, increasing the delay and introducing packet losses in a controlled manner (see §3.2.2). In the case of rate-based protocols that do not use a `cwnd` for rate regulation, we track the unacknowledged packets in flight as the `cwnd` of the protocol. The key insight behind our design is that any congestion control protocol must ultimately react to changing networking conditions. We then try to identify the TCP variant from the observed `cwnd` response via offline processing (see §3.2.3).

Gordon targets identifying congestion control variants that have been deployed in the Windows and Linux kernels. However, since it operates as an interceptor, it is not limited to measuring only TCP behavior and can be used to measure UDP traffic as well. In this work, we concentrate on making measurements on TCP web traffic since TCP supports an overwhelmingly large proportion of Internet traffic [10].

### 3.2.1 Measuring `cwnd` over time

At a high level, we want to determine the evolution of a target congestion control algorithm's `cwnd`. We note that the `cwnd` is essentially the maximum number of unac-

knowledged packets in flight as allowed by the sender's algorithm. Therefore, a simple way to measure the evolution of the `cwnd` is to withhold acknowledgments from a TCP receiver (after the handshake) and count the number of packets received until an RTO is triggered. We refer to this first congestion window as $C_1$. Next, we restart a new connection and this time, we will send $C_1$ acknowledgments and stop. The total number of packets received before a re-transmission would be the total number of packets for the first 2 RTTs, or $C_1 + C_2$. In principle, by repeating this process and progressively measuring $C_1 + C_2 + \cdots + C_n$, we can determine the `cwnd` for the $n^{th}$ RTT and systematically track the evolution of `cwnd` over time. It should be noted here that this effectively normalizes our `cwnd` measurements by RTT. We employ this packet counting methodology with TCP SACK disabled. We resort to restarting connections because we found that previous approaches that do similar `cwnd`-based measurements using delayed acknowledgments do not work for rate-based variants like BBR. These previous techniques typically use the bloated RTTs caused by the delayed ACKs as 'separators' to help them differentiate between different `cwnd` measurements for different RTT's in a single connection. This is not possible with rate-based variants like BBR that fill the entire network pipeline, and thus render this delayed ACK approach to measuring `cwnd` untenable.

Unfortunately, we found that a naïve packet counting strategy does not work well on the real Internet for two reasons. First, most of the available web pages are relatively small and we would not be able to plot any meaningful evolution of the `cwnd`. Second, the naïve approach is very sensitive to random packet losses.

**MTU sizing and crawling for large web-pages.** Since we measure `cwnd` in packets, a straightforward way to obtain more packets from an HTTP/HTTPS page download is to reduce the MTU size of the connection. IPv4 [72] specifies a minimum MTU size of 68 bytes. However, we found that setting an MTU size of 68 bytes often resulted in some connections failing without reason. Through repeated trials for all the

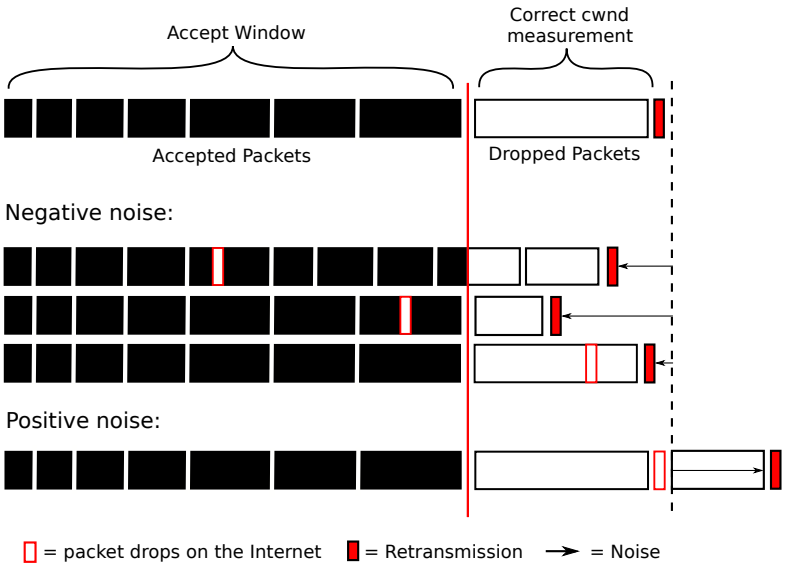



**Figure 3.1:** Possible scenarios for random losses.

websites in the Alexa Top 20,000 list, we found that while an MTU of 68 bytes works for most websites, some accept only connections with larger MTU sizes. To address this issue, Gordon uses binary search to determine the minimum MTU size for a website and performs the measurement using this MTU size. This acceptable MTU size search is done before every measurement since the minimum acceptable MTU size could vary depending on the underlying Internet path, which could change over time.

However, reducing the MTU size was often not enough to yield a sufficiently long trace to identify the TCP variant. Thus, we first used a crawler to determine the available pages for each website (to the best of our ability) and used the largest of these pages to perform our measurements. Using our final network profile (see §3.2.2), Gordon needs about 80 packets for 30 RTTs to be able to accurately plot `cwnd` evolution graphs for more complicated algorithms like CUBIC. With most websites accepting 68-byte MTUs, this would mean an ideal web-page size for Gordon would be at least 165 KB.

**Handling Random Packet Losses.** In Fig. 3.1, we present the various scenarios when we observe packet loss in our measurements. We note that most packet losses

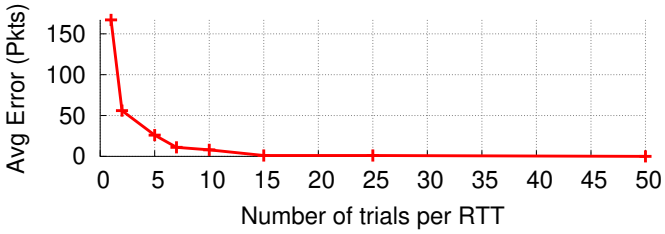



**Figure 3.2:** Sensitivity analysis for repeated measurements.

result in a lower estimate (*negative* noise). It is only when the first re-transmitted packet is lost that we end up counting the entire re-transmitted window twice and have *positive* noise. The latter is easily eliminated if we stop counting packets when we see the re-transmission of any packet in the current `cwnd` measurement window.

We eliminate negative noise caused by random losses by repeating the measurement for each congestion window several times and taking the maximum window measurement as the `cwnd`. In Fig. 3.2, we plot the measurement noise from random losses while measuring various web-servers on the Internet (both real hosts on the Internet and controlled servers set up on AWS) for different number of trials. We see that 15 trials per `cwnd` measurement are sufficient to eliminate negative noise. Here, by 'noise' we mean the cumulative sum of the difference between the measured and ground truth `cwnd` values. In this experiment, the ground truth was taken to be the measurements made over 50 trials. In addition to this, all our experiments were done over wired links to minimize the possibility of random packet losses.

In Fig. 3.3, we plot the window measurements for `reddit.com` using 15 trials per `cwnd` measurement. The red points are the individual window measurements. We see that taking the maximum over 15 trials per window measurement are sufficient to provide us with a relatively smooth `cwnd` evolution curve. The small `cwnd` during the first 5 RTTs is the result of the SSL certificate exchange protocol.

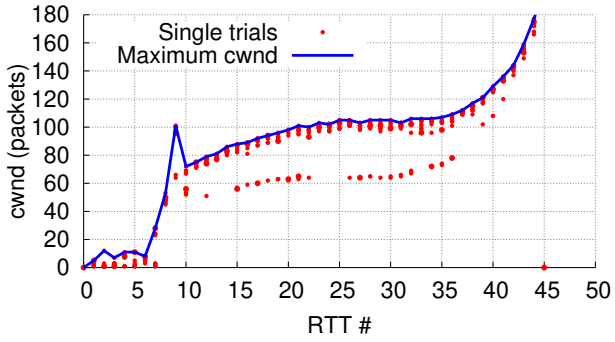



**Figure 3.3:** `cwnd` measurement for `reddit.com`.

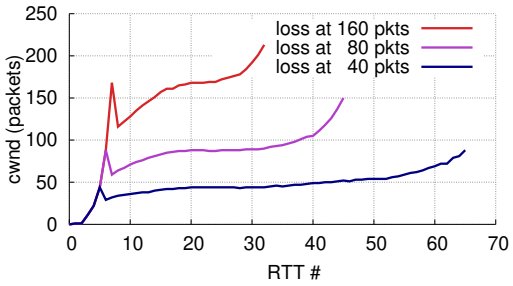



**Figure 3.4:** Evolution of CUBIC `cwnd` for different packet drops.

### 3.2.2 Designing a Network Profile

Our goal is to identify TCP variants from the evolution of their `cwnd` over time. Conceptually, we described a way to do this measurement in §3.2.1. However, we need a way to normalize the measurements so that they can be compared to base measurements of known TCP variants. Since we have full control over the network bottleneck, we can impose a common network profile on all the websites. In particular, we introduce a packet loss event and a bandwidth change event at the network bottleneck and observe the response of the probed TCP algorithm.

**Packet Loss**. Most congestion control algorithms enter their Congestion Avoidance phase when they see a packet loss. The general assumption is that packet losses signal congestion due to buffer overflow. Since we control the network bottleneck, we can decide exactly when a packet loss should happen.

Through measurements, we found that most connections have a starting window size of 10 packets, as suggested by Chu et al. [73, 74]. This means that for a typical Slow Start, we can expect the first few congestion windows to be 10, 20, 40, 80, etc. In Fig. 3.4, we plot the evolution of `cwnd` for a controlled web server running CUBIC while Gordon emulates a drop at different stages of a connection - namely when the measured `cwnd` first reaches more than 40, 80 and 160 packets. We evaluate CUBIC since it has relatively complex `cwnd` evolution in the Congestion Avoidance phase. Fig. 3.4 shows that if the packet drop occurs too early, the subsequent `cwnd` is relatively small and it might be hard to discern between the curve shapes after the packet drop. On the other hand, if the packet drop is too late, the window size becomes very large and we need very large flows (large web pages) to make a measurement that captures the entire CUBIC curve. We found that inflicting a packet loss after the `cwnd` reaches 80 packets achieves a good trade-off between these two concerns. We call this value the *Packet Drop Threshold*. Except for this inflicted packet drop meant to "force" `cwnd`-based TCP variants into *Congestion Avoidance* phase, no other packets are explicitly dropped by Gordon during the measurement. Our buffer is big enough to avoid buffer overflows.

**Regulating the Bottleneck Bandwidth.** Recent rate-based congestion control algorithms like BBR do not back off when they encounter a packet loss. Even so, these algorithms still cap the maximum number of packets in flight. In particular, BBR limits the number of packets in flight to twice the estimated bandwidth-delay product (BDP). To characterize such algorithms, we vary the bottleneck bandwidth and observe how the measured `cwnd` changes when the bottleneck bandwidth changes.

Since our methodology requires us to limit the sender's `cwnd` to about 100 packets to make the flows last long enough, we emulate a BDP of 50 packets. We achieve this BDP by maintaining an RTT of 100 ms between the sender and the receiver and limiting the initial bottleneck bandwidth to 500 packets/s for the first 1,500 packets received. This rate is reduced to 334 packets/s for the next 1,500 packets before the bandwidth

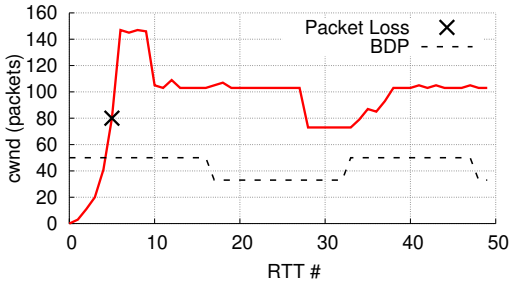



**Figure 3.5:** How BBR reacts to the bandwidth changes.

is restored to 500 packets/s. This behavior can be seen in Fig. 3.5, where we show the available bandwidth in terms of the BDP for the flow (since the delay is a constant). We can see that the `cwnd` for a controlled web server running BBR tracks the available bandwidth at twice the BDP emulated by Gordon after a measurement delay of 10 RTTs. We decided on changing the BDP every 1,500 packets because it would result in a period of 15 to 20 RTTs and works for the general file sizes in our sampled websites. This change in bandwidth also allows us to identify other rate-based variants that may react to a change in bottleneck bandwidth but track the emulated BDP differently.

**Final Network Profile.** In summary, we inflict a packet drop for the first window where the number of packets received is strictly larger than 80. The available bandwidth of the bottleneck alternates between 500 packets/s and 334 packets/s after every 1,500 packets received. In Figures 6.4, we plot the responses for some common congestion control algorithms as measured by Gordon while applying the final network profile. We note that except three pairs of congestion control algorithms (Veno/Vegas, New Reno/HSTCP and CTCP/Illinois) we are generally able to identify the TCP variant from the shape of the curve within the first 30 RTTs. These shapes are deterministic and Gordon is consistently able to record traces like the ones in Figures 6.4 over multiple runs. These shapes show slight deviations when measured over the Internet, and their impact on our classification accuracy is discussed in § 3.3.1.

In the future, if there are deployments of other congestion control variants, additional

network profiles can easily be added to Gordon to identify them. In this work, we limit ourselves to using a single network profile because of the cost associated with measuring each website.

### 3.2.3 Classification

The output from Gordon is a plot of estimated `cwnd` versus time (RTT #) of the target host in response to our final network profile. It remains for us to determine the TCP variant from the shape of the graphs. For measurements that are sufficiently long and yield enough data, we expect the shapes to be similar to those in our control tests.

We use a simple decision-tree-based approach to identifying variants over the Internet (see §3.3.1). One of the benefits of our approach of decoupling measurement and classification is that other researchers are free to swap our classifier with a different classifier. We have made the source code for Gordon and our measurement traces publicly available (§3.6).

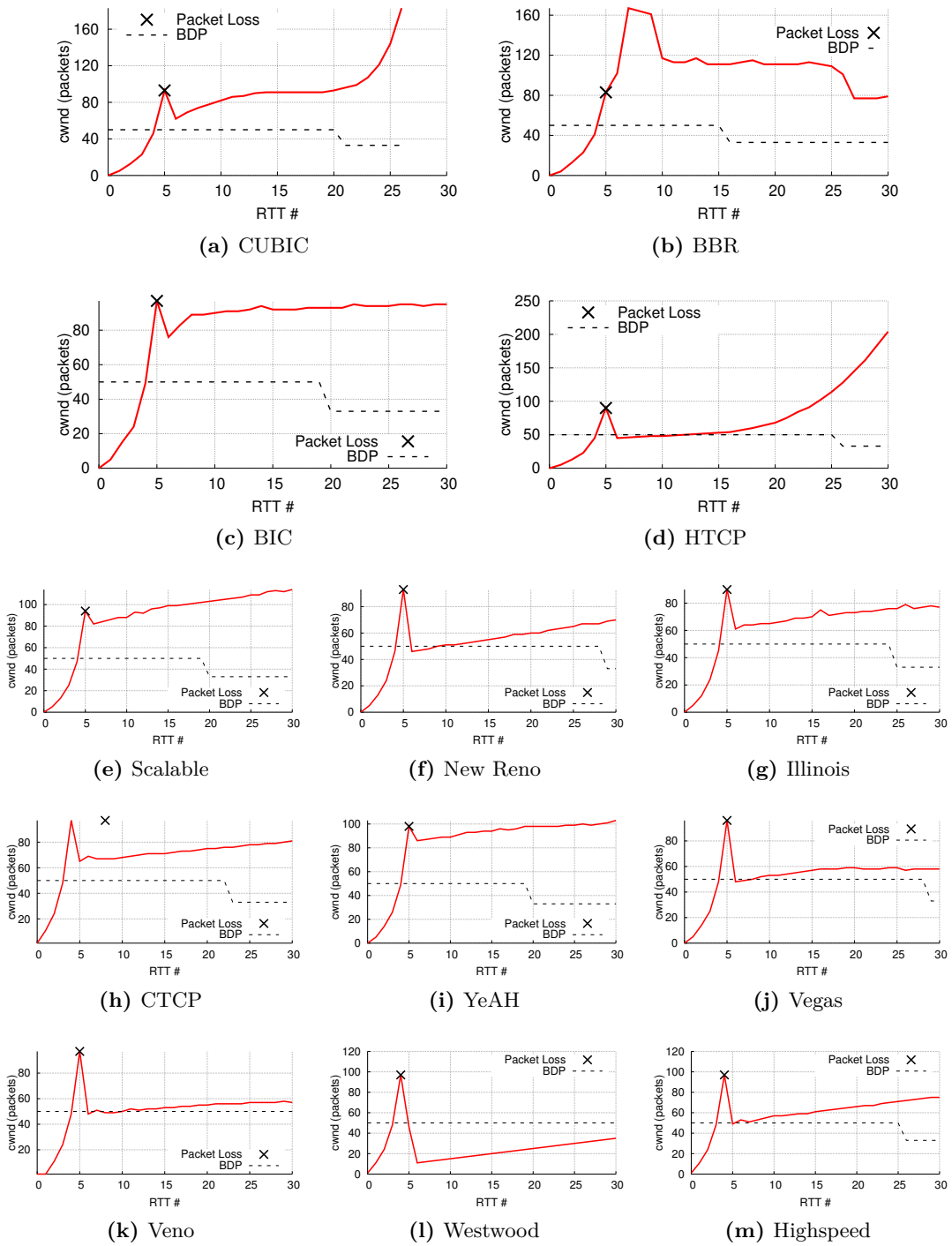
To compute the shape, we first identify the back-off points in the trace that signify the end of Slow Start and the beginning of the Congestion Avoidance phase. Then the traces are treated differently based on the emulated network stimulus that caused this back-off.

**Case 1: Back-off After Packet Loss**. We divide the resulting Congestion Avoidance phase into 3 regions (as shown in Fig. 3.7).

1. Catch-Up: This region corresponds to the region right after the algorithm backs off to a lower `cwnd` after encountering a packet loss.

2. Steady: This is the region where `cwnd` demonstrates linear or no growth.

3. Probe: This is the region when the algorithm tries to probe for more available bandwidth by increasing the `cwnd`.




**(a)** CUBIC **(b)** BBR

**(c)** BIC **(d)** HTCP

**(e)** Scalable **(f)** New Reno **(g)** Illinois

**(h)** CTCP **(i)** YeAH **(j)** Vegas

**(k)** Veno **(l)** Westwood **(m)** Highspeed

**Figure 3.6:** `cwnd` evolution of CCAs in the Linux kernel in response to the final network profile.

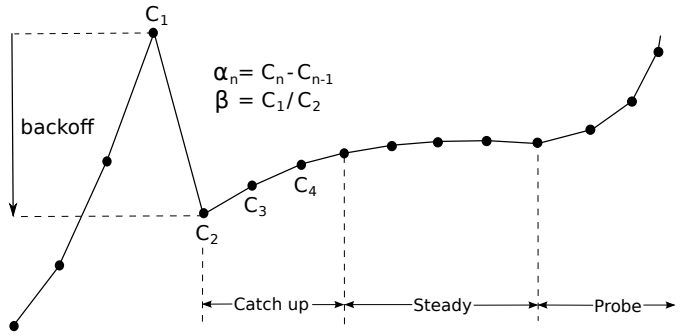



**Figure 3.7:** Calculating $\alpha$ and $\beta$ from the 3 regions.

In addition to this, we also calculate two features common to most loss-based congestion control algorithms – $\alpha$ and $\beta$. Where $C_i$ is the `cwnd` value at the $i^{th}$ RTT of the Congestion Avoidance phase,

1. $\alpha_n = C_n - C_{n-1}, n \geq 3$ is the increase in `cwnd` between 2 successive measurements by Gordon for all RTTs after back off (see Fig. 3.7).

2. $\beta = \frac{C_2}{C_1}$, is the proportion of back-off after packet loss.

Based on the division of the Congestion Avoidance phase into 3 regions, we found that the curves for the known `cwnd`-based TCP variants would take one of the 4 shapes shown in Figures 3.8a - 3.8d. We can computationally classify a curve into one of the 4 shapes based on the change in gradient ($\frac{d\alpha}{dt}$) for each region and by determining whether the steady region exists, as shown in Table 3.1.

Once we have the shape and the values of $\alpha_i$ and $\beta$, we can determine the variant from Table 3.2 by computing $\bar{\alpha}$, the mean of $\alpha_i$. Many of the values in Table 3.2 were obtained from the papers [14, 15, 16, 17, 22, 43, 45, 75, 76] describing the various algorithms. However, we found some difference when we measured the references traces obtained in our network testbed. Some adjustments were then made to ensure that the values of $\beta$ and $\bar{\alpha}$ reflected what we observed in our traces. Algorithms that react to loss, but cannot be classified into one of these shapes are classified as *Unknown*. We note that CUBIC [16], BIC [15] and HTCP [75] can be identified by shape alone. Scalable [76], Illinois [45],

**(a)** CUBIC shape **(b)** BIC shape **(c)** HTCP shape **(d)** Linear shape

**Figure 3.8:** Shapes identified by *Gordon*'s classifier.

**Table 3.1:** Shape Classification.

| Shape | Regions | | |
|---|---|---|---|
| | Catch-up | Steady | Probe |
| CUBIC | $\frac{d\alpha}{dt} < 0$ | ✓ | $\frac{d\alpha}{dt} > 0$ |
| BIC | $\frac{d\alpha}{dt} < 0$ | ✓ | - |
| HTCP | - | - | $\frac{d\alpha}{dt} > 0$ |
| Linear | - | ✓ | - |

CTCP [17], YeAH [42], New Reno [14], Veno [44], Westwood [77] and Vegas [43] all increase their `cwnd` linearly during Congestion Avoidance and are very similar in shape.

While most variants can be differentiated by their values of $\beta$ and $\bar{\alpha}$ (slope of the `cwnd` graph in the Congestion Avoidance phase), Gordon is not able to differentiate between three pairs of algorithms - CTCP and Illinois, New Reno and HSTCP and between Vegas and Veno. In the Congestion Avoidance phase, both Vegas and Veno initially increase their congestion window by 1 every RTT ($\alpha = 1$) before having more or less constant `cwnd` and are therefore indistinguishable when they interact with our network profile. Similarly, both CTCP and Illinois evolve their `cwnd` values using similar functions after seeing a packet loss. HSTCP and New Reno both back-off to half their `cwnd` on seeing a packet loss and increment their `cwnd` by 1 every RTT in Congestion Avoidance mode. Therefore, Gordon classifies them together as 'Vegas/Veno', 'CTCP/Illinois', and 'New Reno/HSTCP,' respectively. It remains as future work to introduce a second stage to disambiguate between these pairs (§3.4).

**Case 2: No Back-off**. For variants that do not back-off after a packet loss, we try

**Table 3.2:** Known TCP Variant Classification.

| Shape | $\beta$ | $\bar{\alpha}$ | Variant |
|---|---|---|---|
| CUBIC | $> 0.66$ | - | CUBIC |
| BIC | $> 0.66$ | - | BIC |
| HTCP | $> 0.5$ | - | HTCP |
| Linear | $> 0.8$ | $= 1.01$ | Scalable |
| | $> 0.8$ | [1, 1.01] | YeAH |
| | $> 0.5$ | N.A. | CTCP/Illinois |
| | $(0.2, 0.5]$ | $< 1$ | Vegas/Veno |
| | $(0.2, 0.5]$ | $= 1$ | New Reno/HSTCP |
| | $\leq 0.2$ | $= 1$ | Westwood |
| Stable regions = 2×BDP | | | BBR |

to either classify them as BBR or an unknown variant. Even though BBR is a rate-based algorithm, it maintains a `cwnd` that is equal to twice the BDP. Also, since BBR uses the maximum receive rate in the past 10 RTTs for calculating it's BDP [78], we expect to see a drop in `cwnd` corresponding to our network profile's drop in bandwidth delayed by 10 RTTs.

Therefore, to identify if these unique behaviors are present in a measurement, the classifier starts by identifying stable regions that show little change in `cwnd` as shown in Fig. 3.9. This is because since our emulated BDP is a step function, we expect BBR's cwnd to trace this step function as well. We then compare these `cwnd` stable regions with the emulated BDP. If the `cwnd` is twice the emulated BDP and the website reduces its `cwnd` 10 RTTs after a bandwidth change was emulated, the algorithm is classified as BBR. If not, it is classified as *Unknown*.

### 3.2.4 Implementation

In Fig. 3.10, we present an overview of Gordon's system design. To inflate the RTT between our measurement server and the remote host (as discussed in §3.2.2), Gordon is run inside a Mahimahi delay shell [79]. We use `wget` [80] to emulate a browser making

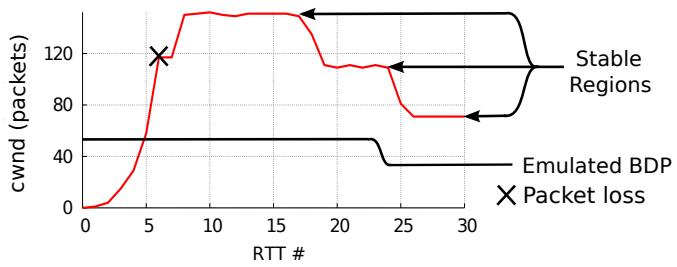



**Figure 3.9:** Identifying stable regions for loss-agnostic flows.

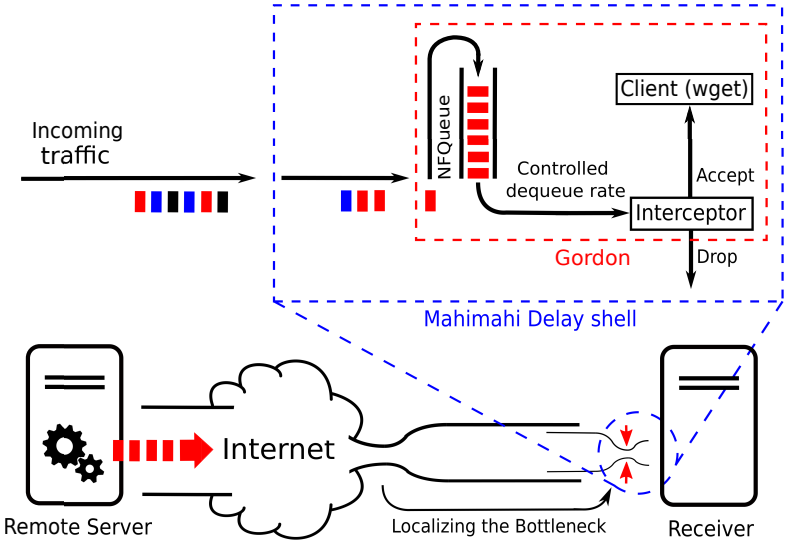



**Figure 3.10:** Gordon Design.

an HTTP/HTTPS GET request to the target web server. The incoming HTTP response packets are redirected to an NFQueue [81] using a Linux Netfilter redirect rule.

The interceptor module of Gordon dequeues packets from the NFQueue and selectively delivers them to `wget` or drops them. Gordon controls the rate at which packets are dequeued from the NFQueue to localize the bottleneck of the connection and to regulate the bottleneck bandwidth (as described in §3.2.2). The interceptor module is implemented in about 350 lines of C code. The final output consists of a trace of the maximum `cwnd` size observed for each RTT period, which is processed offline by a classifier written in 440 lines of Python code. For each website in the Alexa Top 20,000 list [71], we used a web crawler written in about 300 lines of Python code to obtain

URLs to the largest web pages/objects that it could find on the website.

Because of the scale of our measurements, Gordon was also extended into a web service. This web service consisted of a single centralized server responsible for aggregating measurements made by 250 clients (workers) distributed across 5 regions (viewpoints) - Ohio, Sao Paulo, Paris, Mumbai, and Singapore. These workers requested jobs at the granularity of a single `cwnd` measurement for a website, allowing us to spread our connections over time and seem less aggressive to a website. Each host was measured five times (once from each viewpoint) while the centralized server tracked these five individual measurements separately. This web service was implemented in about 2,050 lines of TypeScript code.

At the moment, we have made Gordon and our measurements available on GitHub (see §3.6). We are still working to make the web service available as a live dashboard of the TCP variant distribution on the Internet.

## 3.3 Results

We measured and classified the top 20,000 websites on the Internet based on their Alexa ranking [71]. These measurements were made between 11 July 2019 and 17 October 2019 (unless specified). The distribution of the file sizes obtained using a crawler (see §3.2.1) for the measurements is shown in Fig. 3.11. We can see that about 18% of the websites return pages smaller than the ideal page size of 165 KB (§ 3.2.1). We were able to classify the variants for some of these websites with page sizes smaller than 165 KB. We refer to the remaining websites that cannot be classified as 'Short flows'.

We also found that about 1,302 websites in the Alexa Top 20,000 list did not respond to `wget` requests. These websites had DDoS protection from Cloudflare or Google's ReCaptcha, and therefore did not respond to repeated `wget` requests. A small fraction of the websites also had invalid URLs that did not even open on a web browser. Upon

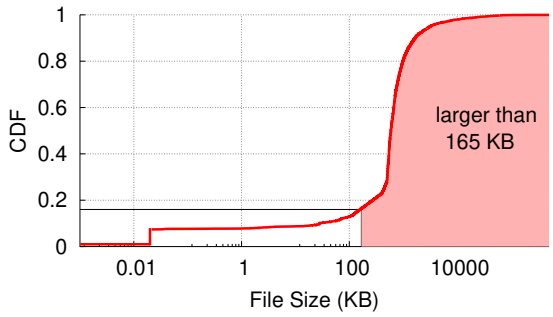



**Figure 3.11:** CDF of file sizes used in measurements.

further investigation, we found that these URLs were links to phishing websites that had been visited so often that they had made it to the Alexa Top 20,000 list. Collectively, we consider these websites to be 'Unresponsive'.

### 3.3.1 Verification of Measurement Accuracy

First, we validate the accuracy of our approach by setting up a physical test web server in Singapore and performing measurements from AWS EC2 instances in 9 locations (viewpoints): Paris, London, Ireland, Sydney, Seoul, Mumbai, Virginia, Oregon, and Ohio. The RTTs for the measurements ranged from 59 ms to 255 ms. To provide the ground truth, the test server runs one of the known TCP variants, which was then measured 5 times from each viewpoint, to give a total 45 measurements for each variant. Later, the configuration was reversed, with the AWS instances running a known variant and acting as web servers while a local server made measurements. In Table 3.3, we present the confusion matrix for these 90 measurements (per algorithm). The key takeaway is that for known variants, the accuracy is high and false positives are relatively rare. Note that the figures in Table 3.3 reflect the accuracy of single measurements. If we take the majority result across the five measurements from an individual viewpoint, we can achieve 100% classification success. The errors are caused by noisy measurements arising from Internet traffic.

**Table 3.3:** Classification accuracy.

| | Classified as | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | BBR | CUBIC | BIC | HTCP | Scalable | YeAH | Vegas/Veno | New Reno/HSTCP | CTCP/Illinois | Westwood | Unknown |
| BBR | **98%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 2% |
| CUBIC | 0% | **95%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 5% |
| BIC | 0% | 9% | **91%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| HTCP | 0% | 0% | 0% | **95%** | 0% | 0% | 0% | 0% | 0% | 0% | 5% |
| Scalable | 0% | 0% | 0% | 0% | **98%** | 0% | 0% | 0% | 0% | 0% | 2% |
| YeAH | 0% | 0% | 2% | 0% | 0% | **98%** | 0% | 0% | 0% | 0% | 0% |
| Vegas/Veno | 0% | 0% | 0% | 0% | 0% | 0% | **94%** | 6% | 0% | 0% | 0% |
| New Reno/HSTCP | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **96%** | 0% | 0% | 4% |
| CTCP/Illinois | 0% | 0% | 3% | 0% | 0% | 0% | 0% | 0% | **94%** | 0% | 3% |
| Westwood | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 2% | 0% | **98%** | 0% |

### 3.3.2 TCP variants on the Internet

Each target website from Alexa Top 20,000 was measured from AWS EC2 instances in the US (Ohio), Europe (Paris), South America (Sao Paulo) and Asia (Mumbai and Singapore). Our measurements were made from different viewpoints to help us get the best view of a website's congestion control behavior (since all websites are not hosted by CDNs). In addition, we kept re-measuring websites that we were not able to classify as a known variant. These iterative measurements were stopped only when a re-run did not further improve the number of classified websites.

Table 3.4 shows the distribution of TCP variants on the Internet as measured from these viewpoints. As expected, we found that for certain websites, some viewpoints gave less noisy measurements compared to others. This is the only reason for the slight discrepancies between numbers reported from different viewpoints. Out of the 20,000 target websites, a total of 13,739 websites were classified similarly from all viewpoints. Out of the remaining 6,261 websites, 1,424 websites were successfully classified from

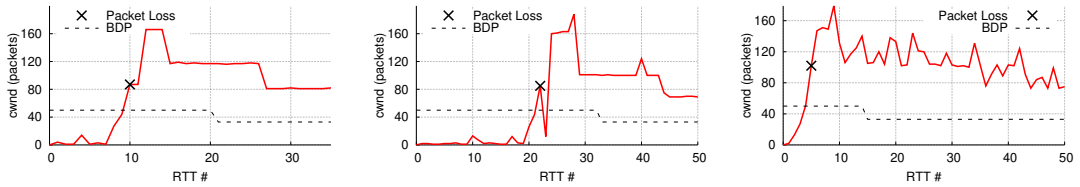
**Table 3.4:** Distribution of variants as measured from different viewpoints on the Internet.

| Variant | Ohio | | Paris | | Mumbai | | Singapore | | Sao Paulo | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Sites | Share | Sites | Share | Sites | Share | Sites | Share | Sites | Share |
| CUBIC | 5,966 | 29.83% | 5,893 | 29.47% | 5,950 | 29.75% | 5,930 | 29.65% | 5,966 | 29.83% |
| BBR | 3,297 | 16.49% | 3,414 | 17.07% | 3,378 | 16.89% | 3,386 | 16.93% | 3,393 | 16.96% |
| BBR G1.1 | 167 | 0.84% | 167 | 0.84% | 167 | 0.84% | 167 | 0.84% | 167 | 0.84% |
| YeAH | 1,102 | 5.51% | 1,092 | 5.46% | 1,081 | 5.40% | 1,115 | 5.57% | 1,112 | 5.56% |
| CTCP/Illinois | 1,085 | 5.42% | 1,054 | 5.27% | 1,092 | 5.46% | 1,082 | 5.41% | 1,097 | 5.48% |
| Vegas/Veno | 556 | 2.78% | 557 | 2.78% | 556 | 2.78% | 551 | 2.75% | 548 | 2.74% |
| HTCP | 543 | 2.71% | 551 | 2.75% | 544 | 2.72% | 541 | 2.70% | 500 | 2.50% |
| BIC | 169 | 0.85% | 166 | 0.83% | 161 | 0.80% | 165 | 0.83% | 165 | 0.83% |
| N.Reno/HSTCP | 154 | 0.77% | 151 | 0.75% | 154 | 0.77% | 147 | 0.73% | 151 | 0.75% |
| Scalable | 36 | 0.18% | 37 | 0.18% | 37 | 0.18% | 37 | 0.18% | 36 | 0.18% |
| Westwood | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| Unknown | 4,143 | 20.71% | 4,132 | 20.66% | 4,096 | 20.48% | 4,105 | 20.52% | 4,074 | 20.37% |
| Short-flows | 1,480 | 7.40% | 1,484 | 7.42% | 1,482 | 7.41% | 1,472 | 7.36% | 1,489 | 7.44% |
| Unresponsive | 1,302 | 6.51% | 1,302 | 6.51% | 1,302 | 6.51% | 1,302 | 6.51% | 1,302 | 6.51% |
| Total | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% |

some viewpoint and 3,535 websites could not be classified from any viewpoints.

The distribution of variants as measured from these viewpoints shows the same general trend of CUBIC [16] being the dominant congestion control variant in terms of website count, with BBR [24] coming in second. In Table 3.5, we show the consolidated numbers for all websites following the rule that if a website has been identified to be using some known congestion control variant in any of the regions, it is considered to be running that congestion control variant. There were no classification conflicts between different viewpoints for these 1,424 successfully classified websites. In other words, we found no evidence for websites deploying different congestion control algorithms in different regions.

**Google's custom version of BBR.** Gordon discovered that some Google-owned domains (167, including YouTube) were using a modified version of BBR that reacted differently to packet loss compared to vanilla BBR (see Fig. 3.12b). This difference was first observed in February 2019. Before that, we had observed traces resembling vanilla BBR (Fig. 3.12a). While we initially suspected that this new variant was BBRv2,




**(a)** `youtube.com` Dec '18. **(b)** `youtube.com` Feb '19. **(c)** BBRv2, Alpha release, July '19.

**Figure 3.12:** The evolution of BBR.

we checked the `cwnd` evolution of BBRv2 that was recently released in July 2019 (see Fig. 3.12c) and found that it was not. We thus refer to this variant as BBR G1.1 in Tables 3.4 and 3.5. It should be noted here that this anomalous behavior was only observed for Google websites. None of the other websites identified to be using BBR showed this anomalous behavior even after repeated measurements. They all deployed vanilla BBR.

We have confirmed our findings about BBR with Google. In particular, Google is frequently running experiments and testing refinements to BBR. Google currently runs a slightly modified version of BBRv1 that has a gentler reaction to packet loss than the open-source BBRv1. This experimental variant (BBR G1.1) was meant as an incremental step toward BBRv2. However, BBR G1.1 was deployed in late 2017, which does not explain our observation of a trace resembling vanilla BBR from Google websites in December 2018. We have thus been measuring Google sites repeatedly and found that we still see traces with the shape shown in Fig. 3.12a occasionally. Hence, it is possible that Gordon occasionally fails to detect the drop in `cwnd` for BBR G1.1 immediately after a packet loss event from time to time. At some level, this is not surprising since BBR does not actively maintain a `cwnd` like traditional `cwnd`-based TCP variants.

**Table 3.5:** Distribution of variants.

| Variant | Websites | Proportion |
|---|---|---|
| CUBIC [16] | 6,139 | 30.70% |
| BBR [22] | 3,550 | 17.75% |
| BBR G1.1 | 167 | 0.84% |
| YeAH [42] | 1,162 | 5.81% |
| CTCP [17]/Illinois[45] | 1,148 | 5.74% |
| Vegas [43]/Veno [44] | 564 | 2.82% |
| HTCP [75] | 560 | 2.80% |
| BIC [15] | 181 | 0.90% |
| New Reno [14]/HSTCP [82] | 160 | 0.80% |
| Scalable [76] | 39 | 0.20% |
| Westwood [77] | 0 | 0.00% |
| Unknown | 3,535 | 17.67% |
| Short flows | 1,493 | 7.46% |
| Unresponsive websites | 1,302 | 6.51% |
| Total | 20,000 | 100% |

### 3.3.3 Traffic Volume & Popularity

We believe that the distribution of TCP variants by pure website count in Table 3.5 does not present the full picture.

**Understanding Traffic by Volume**. In Table 3.6, we present Internet traffic volume data by Sandvine [10]. Based on the reported Internet traffic volume, we expect BBR variants to already contribute at least 40% of the global Internet traffic. During our measurements, we found that Netflix had switched from CUBIC to BBR in early March 2019, only to switch back to CUBIC in April 2019. We note that Google recently announced that Netflix is currently experimenting with BBR [26]. We also contacted Netflix and were told that the Netflix website was hosted on AWS. Netflix however uses different protocols depending on the context, and that most of their video streaming traffic is delivered via their Open Connect Appliances running FreeBSD's New Reno with RACK [83] extensions. The reason for choosing New Reno over CUBIC was

**Table 3.6:** Excerpt of website traffic share (source: Sandvine [10]).

| Site | Downstream traffic share | Variant* |
|---|---|---|
| Amazon Prime | 3.69% | CUBIC |
| Netflix | 15% | CUBIC |
| Netflix Video | | New Reno+ |
| YouTube | 11.35% | BBR G1.1 |
| Other Google sites | 28% | BBR G1.1 |
| Steam downloads | 2.84% | BBR |

* as measured on servers serving static HTTP/HTTPS pages.
+ as informed by Netflix, not measured by Gordon.

that the Netflix team felt that the New Reno stack was more mature and that improving loss-detection/loss-recovery heuristics from RACK would be more helpful for their chunked-delivery use case. We were informed by the Akamai team that Akamai would be deploying BBR G1.1 on more of their hosted sites in the near future. If Netflix and Akamai does do the switch to BBR, BBR and its variants' traffic share on the Internet would increase to well above 50%.

**Understanding Traffic by Popularity**. Similar trends can also be observed if we consider the popularity of the websites. In Fig. 3.13, we plot the distribution of the identified variants for the top-k sites. We see that BBR is the most widely deployed variant among the top 250 websites, accounting for 25.2% of all hosts. Another interesting observation was that BBR was the most common TCP variant for adult entertainment websites. All in all, our results suggest that BBR is rapidly catching up with CUBIC in popularity and some variant of BBR is poised to overtake CUBIC as the dominant TCP variant.

### 3.3.4 Whithering the Unknown Variants

One of the benefits of our methodology is that Gordon can provide us with insights on a congestion control variant even if we are not able to identify it. Given that a larger number of websites (5,028 in total) were classified as 'Unknown' or 'Short flows' (to-

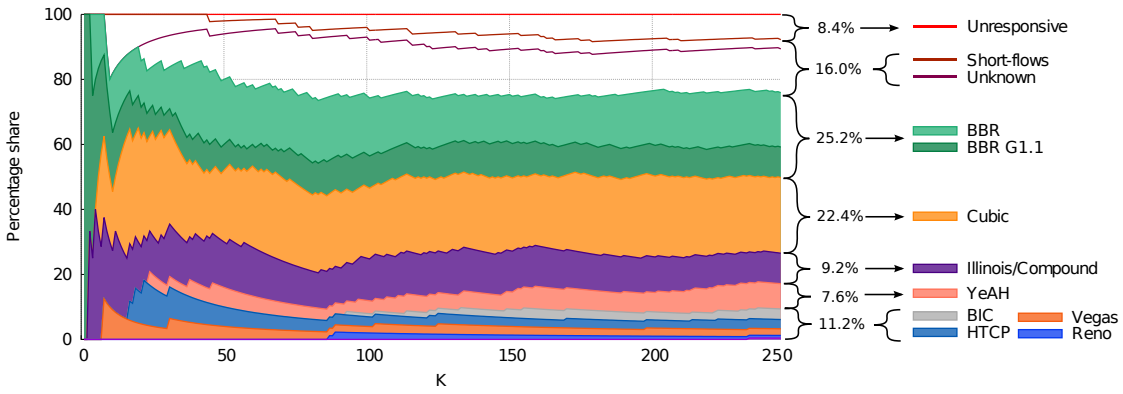



**Figure 3.13:** Distribution of variants among the Alexa Top-$k$ sites.

gether referred to as 'Uncategorized' hosts henceforth), we ran a variety of *new* network profiles to investigate their behavior under different conditions. These network profiles were designed with different combinations of emulated BDPs, delays and Packet Drop Thresholds. We hypothesize that the same TCP variant would exhibit the same behaviors for all network profiles, while different TCP variants may exhibit the same behavior for some profiles, but different behavior for others, to allow us to tell them apart. Our goal is to identify large clusters of traces that could suggest the presence of a new major, but hitherto unknown, variant.

Given that Gordon can modify these three network parameters, we came up with eight custom network profiles (shown in Table 3.7) that are distributed over the range of these network parameters. Each of these network profiles emulates a fixed RTT and BDP for an experiment run and introduces a packet drop when the `cwnd` size goes above the Packet Drop Threshold for the first time.

**Reaction to Loss.** We found that among the 5,028 (25.14%) websites with unknown variants, only 3,275 (16.38%) of them reacted to packet loss. Out of these 3,275 websites, 1,493 (7.47%) are short flows and the remaining 1,782 (8.91%) websites gave inconsistent measurements after reacting to a packet loss. In other words, repeating our measurements yielded different traces each time. We hypothesize that these are

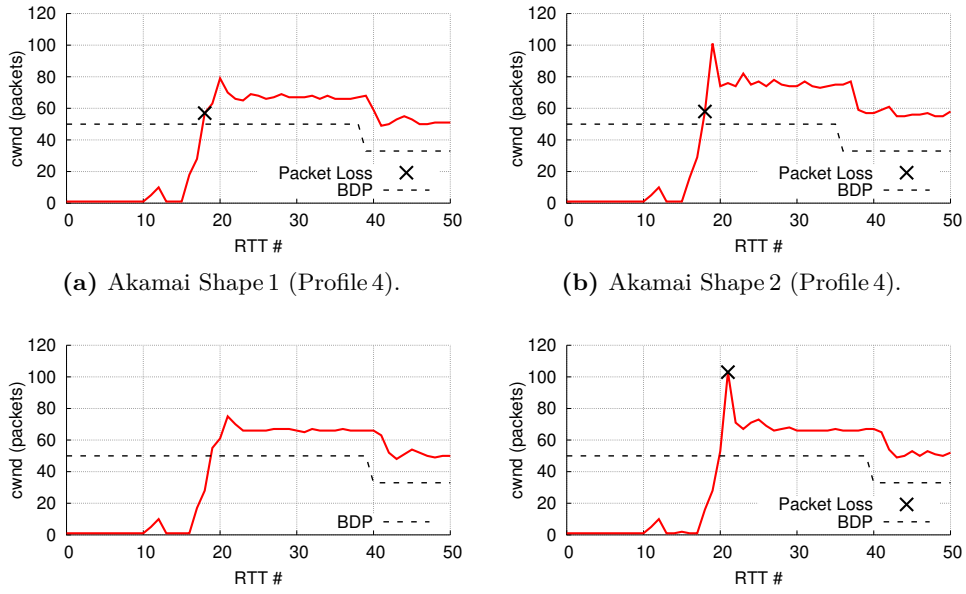
**Table 3.7:** Custom network profiles to investigate uncategorized hosts.

| Profile | Packet Drop Threshold (packets) | RTT (ms) | BDP (packets) |
|---|---|---|---|
| 1 | 80 | 100 | 50 |
| 2 | 80 | 100 | 25 |
| 3 | 80 | 50 | 25 |
| 4 | 40 | 100 | 50 |
| 5 | 40 | 100 | 25 |
| 6 | 40 | 50 | 25 |
| 7 | 40 | 200 | 100 |
| 8 | 40 | 100 | 100 |

likely `cwnd`-based TCP variants that we are not able to classify because of noise. It was surprising that this noisiness was observed for all 5 viewpoints.

Among the remaining 1,753 (8.77%) websites with variants that did not react to the packet loss, we found that a large class of 1,103 (5.52%) websites reacted to changes in the BDP. For the remaining 650 (3.25%) websites that did not react to the packet loss, we could not determine if they reacted to changes in the BDP because of noisy measurements. Again, repeating measurements yielded different traces each time.

**Akamai Congestion Control Variant.** We found that all 1,103 (5.52%) websites that reacted to changes in the BDP but not to packet loss were all hosted by the Akamai CDN. These websites typically maintained the `cwnd` at a fixed multiple of the BDP, ranging from 1.2 to 1.5. Typical shapes for these websites are shown in Figures 3.14a - 3.14d. We found that the traces for the AkamaiCC websites in response to our custom network profiles tend to take one of 2 shapes shown in Figs. 3.14a and 3.14b. As shown in Figs. 3.14c and 3.14d, these shapes remain consistent across different network profiles. While this behavior matches no known TCP implementation in the Windows or the Linux kernel, we hypothesized that it was the result of TCP optimizations developed at Akamai [84] or the deployment of FAST TCP [85]. We refer henceforth to this variant as *AkamaiCC*. Some notable websites identified to use AkamaiCC include `microsoft.com`, `apple.com`, and `hulu.com`.




**(a)** Akamai Shape 1 (Profile 4).

**(b)** Akamai Shape 2 (Profile 4).

**(c)** Akamai Shape 1 (Profile 1).

**(d)** Akamai Shape 2 (Profile 4).

**Figure 3.14:** Sample traces for websites hosted by Akamai.

We found that a total of 1,260 (6.30%) websites among the Alexa Top 20,000 websites were hosted by Akamai, but not all of them show the behavior illustrated in Figures 3.14a - 3.14d. All the remaining 157 (0.79%) Akamai-hosted websites did not react to loss and yielded noisy measurements and so are categorized as 'Unknown.' It is plausible that these websites are also running AkamaiCC, but we are not able to see the AkamaiCC shape in their traces because of noise.

We contacted Akamai to verify these results and they have confirmed that AkamaiCC was likely FAST TCP. That said, the Akamai team also highlighted that Akamai does not typically deploy a specific TCP variant for a specific website (though there were cases where they might). It is plausible that our findings were an artifact of our experimental setup and the pages that we had chosen to download from the Alexa Top 20,000 websites. Akamai currently deploys a variety of Congestion Control variants including FAST TCP, a modified version of Reno, vanilla BBR, a modified version of BBR, QDK (a proprietary Congestion Control algorithm), and CUBIC. In addition, under some network conditions,

**Table 3.8:** Summary of websites not classified as known congestion control variants.

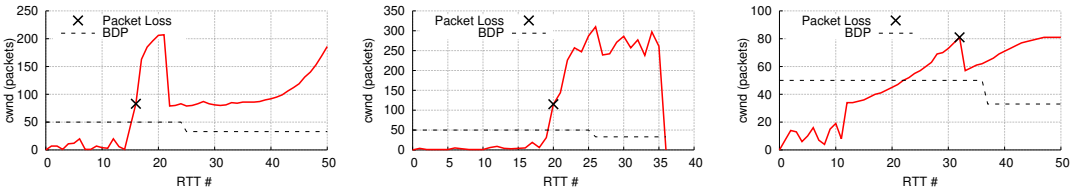
| Type | React to Packet Loss? | React to BDP? | Websites (share) |
|---|---|---|---|
| AkamaiCC | ✗ | ✓ | 1,103 (5.52%) |
| Unknown Akamai | ✗ | ? | 157 (0.79%) |
| Unknown | ✗ | ? | 493 (2.47%) |
| | ✓ | ? | 1,782 (8.91%) |
| Short flows | ✓ | ? | 1,493 (7.47%) |
| Unresponsive | ? | ? | 1,302 (6.51%) |
| Total | | | 6,330 (31.65%) |

Akamai servers could switch between these algorithms in the middle of a connection. This would be a plausible explanation for the noisy and unrecognizable traces observed for some of the 157 Akamai-hosted websites that we could not classify.

In summary, as shown in Table 3.8, a large number of the websites that Gordon was not able to identify as known variants can be attributed to the 1,103 (5.51%) websites running AkamaiCC. In other words, Gordon can classify 14,773 (73.87%) of the Alexa Top 20,000 websites as some variant. Among the remaining 5,227 (26.14 %) websites, 1,302 (6.51%) were found to be unresponsive, 1,493 (7.47%) had web pages that were too small to yield a long enough trace for classification, and 2,432 (12.16%) could not be classified because most of them yielded noisy and inconsistent traces.

**The best of the rest.** We know from our results in §3.3.1 that some of the websites classified as one of the 2,432 "Unknown" websites would be known variants that Gordon is not able to identify because of noise. However, it is likely that there also new and undocumented variants because of the diverse behaviors that we observed. We reproduce 3 of the more interesting traces in Figure 3.15:

1. `amazon.com`: In Fig. 3.15a, we see that Amazon has deployed a TCP variant which resembles HTCP in its Congestion Avoidance phase. However, the variant either does not back-off on seeing a Gordon-induced packet loss or has a significant multi-RTT delay in its response to loss. The reduction at `cwnd` = 200 is not due




(a) `amazon.com` (b) `zhihu.com` (c) `yahoo.co.jp`

**Figure 3.15:** The weird and wonderful world of TCP *in the wild*.

to buffer overflow. The buffer for Gordon can hold significantly more packets than that without suffering overflow.

2. `zhihu.com`: The behavior observed for the trace in Fig. 3.15b showed an unrestricted growth in the sender's `cwnd`. It seems like the deployed TCP variant is oblivious to packet losses and bandwidth changes, and simply maintains a very high and constant `cwnd`.

3. `yahoo.co.jp`: The behavior in Fig. 3.15c suggests that whatever congestion control variant Yahoo has deployed is exiting Slow Start prematurely, and conservatively increases its `cwnd` until it sees a packet loss.

We believe that these examples serve to illustrate the diversity in the unknown variants and would convince the reader that it was not embarrassing that we have not been able to identify these variants in the first instance. At some level, these results are also a wake-up call. In academia, we often assumed that we would be the ones to invent new TCP variants and the industry would subsequently, pick the winner. The development of BBR has been led by Google, and it seems that companies such as Akamai, Amazon, and Netflix, are not too far behind.

### 3.3.5 TCP Evolution over the past Two Decades

To have an overview of TCP evolution on the Internet over the years, we compare our results to previous studies conducted by Padhye et al. [19], Medina et al. [20] and Yang et

**Table 3.9:** Evolution of TCP variants on the Internet over the past 2 decades.

| | 2001 [19] | | 2004 [20] | | 2011 [21] | | 2019 | |
|---|---|---|---|---|---|---|---|---|
| Loss-based AIMD | New Reno | 35% (1,571) | New Reno | 25% (21,266) | | | New Reno[h] | 0.80% (160) |
| | Reno | 21% (945) | Reno | 5% (4,115) | AIMD | 12.46% (623) | Reno[d] | - |
| | Tahoe | 26% (1,211) | Tahoe | 3% (2,164) | | | Tahoe[d] | - |
| Loss-based MIMD | - | - | - | - | CUBIC | 22.30% (1,115) | CUBIC | 30.70% (6,139) |
| | | | | | BIC | 10.62% (531) | BIC | 0.90% (181) |
| | | | | | HSTCP | 7.38% (369) | HSTCP[h] | |
| | | | | | Scalable | 1.38% (69) | Scalable | 0.20% (39) |
| Delay-based AIMD | - | - | - | - | Vegas | 1.16% (58) | Vegas[v] | 2.82% (564) |
| | | | | | Westwood | 2.08% (104) | Westwood | 0% (0) |
| Delay-based MIMD | - | - | - | - | CTCP | 6.68% (334) | CTCP | 5.74% (1,148) |
| | | | | | Illinois | 0.56% (28) | Illinois | |
| | | | | | Veno | 0.90% (45) | Veno[v] | |
| | | | | | YeAH | 1.44% (72) | YeAH | 5.81% (1,162) |
| | | | | | HTCP | 0.36% (18) | HTCP | 2.80% (560) |
| Rate-based | - | - | - | - | - | - | BBR | 17.75% (3,550) |
| | - | - | - | - | - | - | BBR G1.1 | 0.84% (167) |
| | - | - | - | - | - | - | AkamaiCC | 5.51% (1,103) |
| Unknown | | 17.30% (792) | | 53% (44,950) | Unknown | 3.96% (198) | | 12.16% (2,432) |
| | | | | | Abnormal SS* | 2.88% (144) | | |
| Short flows | | - | | - | | 26% (1,300) | | 7.47% (1,493) |
| Unresponsive | | 0.7% (30) | | 14% (11,529) | | - | | 6.51% (1,302) |
| Total hosts | | 100% (4,550) | | 100% (84,394) | | 100% (5,000) | | 100% (20,000) |

[d] These implementations have been deprecated.
[h] HTCP and New Reno have been classified together.
[v] Veno and Vegas have been classified together.
[*] websites identified by CAAI having abnormal Slow Starts.

al. [21] in Table 3.9. Since the total hosts measured over the various studies vary widely and there is also a large variance in terms of success rates, we normalize the results over the total reported successful classifications for each study in Table 3.10 to obtain estimated proportions of the various variants. New Reno had rapidly surpassed Reno as the dominant TCP variant in early 2000's. The next 10 years saw the rise of loss-based MIMD protocols such as CUBIC and BIC which dominated the overall adoption decreasing the share of loss-based AIMD protocols. By 2011, HSTCP and Microsoft's CTCP also held significant shares of 10% and 9% respectively.

Five years later in 2019, traditional loss-based AIMD schemes have become near-extinct (at least in terms of pure website count). On the other hand, the adoption of delay-based Vegas has almost doubled. CUBIC has remained the most popular TCP variant and has increased its share among the top 20,000 hosts to some 36% compared

to 30% in 2011, while the share of BIC and HSTCP has reduced significantly.

While delay-based variants seem to have become slightly more popular over the past decade (increasing in share from 15% in 2011 to 20% in 2019), CTCP seems to have slightly decreased in popularity in recent years. We suspect that this is likely due to Microsoft's addition of CUBIC as an option in Windows Server 2016 and making CUBIC the default congestion control algorithm in Windows 10 (2019 builds) and Windows Server 2019 [86].

Finally, the most significant development between 2011 and 2019 is the emergence of rate-based variants like BBR and AkamaiCC. BBR and it's variant BBR G1.1 now have an approximate 22% share while AkamaiCC has a 6% share. Overall, CUBIC seems to have increased in popularity at the expense of traditional loss-based AIMD variants, but this lead will likely be under pressure from rate-based variants in the near future.

## 3.4 Discussion

Over the course of our measurements, it became clear that the Internet was a constantly evolving entity and a moving target. While the main results reported in this study were from the measurements done between July and October in 2019, findings like Google's change in its BBR deployment and the existence of rate-based variants other than BBR on the Internet shows that we are currently in midst of a shift from the traditional congestion control paradigm.

The Gordon project started as a measurement study to understand the latest distribution of TCP variants on the Internet since the last study was done 8 years ago. We decided to start with a simple approach of measuring `cwnd` one RTT at a time. While the modifications we adopted (§3.2.1) might seem straightforward, in hindsight, it took us a while to get the details right and collect all the data. It turns out that our chosen approach made the mitigation of random losses much easier and works well for the vast

**Table 3.10:** Share of TCP variants normalized over all successful classifications.

| | 2001 [19] | | 2004 [20] | | 2011 [21] | | 2019 | |
|---|---|---|---|---|---|---|---|---|
| Loss-based AIMD | New Reno | 35% | New Reno | 29% | | | New Reno[h] | <1% |
| | Reno | 21% | Reno | 6% | AIMD | 17% | Reno[d] | - |
| | Tahoe | 26% | Tahoe | 3% | | | Tahoe[d] | - |
| Loss-based MIMD | - | - | - | - | CUBIC | 30% | CUBIC | 36% |
| | | | | | BIC | 14% | BIC | 1% |
| | | | | | HSTCP | 10% | HSTCP[h] | |
| | | | | | Scalable | 2% | Scalable | <1% |
| Delay-based AIMD | - | - | - | - | Vegas | 2% | Vegas[v] | 3% |
| | | | | | Westwood | 3% | Westwood | 0% |
| Delay-based MIMD | - | - | - | - | CTCP | 9% | CTCP | 7% |
| | | | | | Illinois | <1% | Illinois | |
| | | | | | Veno | 1% | Veno[v] | |
| | | | | | YeAH | 2% | YeAH | 7% |
| | | | | | HTCP | <1% | HTCP | 3% |
| Rate-based | - | - | - | - | - | - | BBR | 21% |
| | - | - | - | - | - | - | BBR G1.1 | 1% |
| | - | - | - | - | - | - | AkamaiCC | 6% |
| Unknown | | 18% | | 62% | Unknown | 5% | | 14% |
| | | | | | Abnormal SS* | 4% | | |
| Total Measurable hosts | | 100% | | 100% | | 100% | | 100% |

[d] These implementations have been deprecated.
[h] HTCP and New Reno have been classified together.
[v] Veno and Vegas have been classified together.
* websites identified by CAAI having abnormal Slow Starts.

majority of websites surveyed.

**Bringing Uncooperative Sites On-board**. We suspect that our approach in making a large number of abrupt connections can be improved. In particular, we observed that many of the websites for which we were not able to identify the TCP variant were in the banking and government sectors (§3.3.2). We are not entirely surprised that we were often throttled or blocked in the midst of a measurement run since our connections would seem to be misbehaving to the TCP sender. We have classified these hosts as 'Unresponsive' in Table 3.5. In fact, as we demonstrate in Chapter 6, Gordon's aggressive measurement strategy does not work on most websites anymore. In response to this, we have since developed a more future-proof passive measurement strategy.

**Easy Extensions**. While we had hoped to design "one tool to measure them all," we have subsequently realized that there is a limitation to our approach. Because we normalize a host's sending behavior by RTT, behavioral differences that exist at a granularity smaller than one RTT cannot be observed. This would also explain why we are often unable to observe a drop in the `cwnd` after a packet loss event for the currently deployed Google G1.1 BBR variant (see §3.3.2). That said, the variants that Gordon is not able to differentiate are relatively small and insignificant in terms of popularity, so we did not invest more time to work on them. In principle, we can classify them by adding a second stage to the classification process after Gordon is done with a high-level classification.

**Understanding Long-Term TCP Evolution**. Our results suggest an active push by large Internet companies towards rate-based TCP variants. However, as common cloud service providers like Google Cloud are now enabling BBR [87] and the Akamai CDN is running AkamaiCC, small entities using these services might be making the switch to rate-based variants without knowing it. This suggests that the landscape of TCP congestion control is undergoing rapid and significant change, possibly led by the CDNs. Therefore, we do not think that taking a snapshot every 10 years is good enough and more frequent measurements need to be made.

## 3.5 Summary

In summary , we used Gordon to identify the TCP variants for the top 20,000 websites based on their Alexa rankings [71]. Our results suggest that CUBIC is currently still the dominant TCP variant on the Internet and is deployed at 36% of the Alexa Top 20,000 websites that we successfully classified. Rate-based TCP variants like BBR have the next largest share. While BBR and its variant BBR G1.1 have a share of only 22% in terms of website count, their present share of total Internet traffic volume is likely to

be larger than 40% [10]. This proportion will almost certainly exceed 50% if Netflix also decides to adopt BBR.

Since it is natural for the Internet to evolve, this is not the first time that we are seeing a dominant TCP variant in the process of being replaced by an alternative. However, we believe that the current change represents a fundamental shift in the underlying Internet. In previous transitions, all the TCP variants were `cwnd`-based and the interactions between AIMD/MIMD protocols have been well-understood. BBR represents a fundamental departure in our approach to congestion control. While BBR has been studied and issues have been highlighted [24, 88], to the best of our knowledge, the interactions between BBR and CUBIC at scale are not fully understood. While nothing seems to have broken thus far even as BBR has gained traction, we do not yet know for sure that nothing will necessarily go wrong. We believe that our results suggest the need for more in-depth study in the interactions between BBR and CUBIC to ensure the future stability and success of the Internet.

## 3.6 Resources

Our measurement tool, along with the `cwnd` traces for the Alexa Top 20,000 websites is available on GitHub (`https://github.com/NUS-SNL/Gordon`).

Chapter 4

# Are we heading towards a BBR-dominant Internet?

The rapid adoption rates of BBR in three short years since its introduction (as described in Chapter 3) inspire one to ask an obvious question: *Are we heading towards a BBR-dominant Internet?* In this chapter, we explore how we can answer this question.

Predicting how widely we can expect BBR to be adopted on the Internet is an important question because the stability of the Internet depends on the competing flows interacting well with one another. We have not experienced a *congestion collapse* [7] for many years likely because the vast majority of flows have been well-understood AIMD/MIMD-window-based TCP flows [18]. The last major change in the Internet congestion landscape happened when CUBIC replaced New Reno [19, 21]. That transition was however relatively incremental because both CUBIC and New Reno are loss-based and `cwnd`-based. Therefore, all existing in-network solutions, policing algorithms, and AQMs already deployed on the Internet could largely remain unchanged.

On the other hand, if BBR were to replace CUBIC as the dominant congestion control algorithm for the Internet, it represents a fundamental paradigm shift. Many classic networking questions that have supposedly been settled would have to be re-

evaluated. For example, it was said that router buffers ought to be sized inversely proportional to $\sqrt{N}$, where $N$ is the number of flows [89]. Later, it was shown that even tiny buffers might suffice under certain conditions [90]. However, these rules of thumb assumed that flows were loss-based. BBR is loss-agnostic [91]. In other words, a BBR-dominant future Internet [1] could have potentially wide-ranging consequences and even fundamental issues like buffer sizing will need to be revisited [40].

The first step toward predicting the future composition of the Internet's congestion control landscape is to understand the incentive(s) for switching to BBR. Companies like Dropbox [35], YouTube [92], and Spotify [36] that have adopted BBR have cited better throughput as the most common reason for making the switch. To determine if switching to BBR would continue to yield better throughput, we need to understand how BBR flows interact with CUBIC flows. While a model was earlier proposed by Ware et al [25], we found that some of the assumptions made were not realistic and verified experimentally that their model does not make accurate predictions. To address this gap, we developed a new model that is able to accurately model BBR's performance to within 5% error for most realistic buffer sizes.

With an accurate model of competing CUBIC and BBR flows, we formulate the interactions between CUBIC and BBR flows as a *normal form game*. A *normal form game* is a standard representation of a strategic game between a finite number of players where each player has the opportunity to maximize its utility by selecting some strategy from a finite set of strategies. By doing so, we can abstract the setting in which websites chose their congestion control algorithms as a simple normal form game in which some finite number of players (websites) try to maximize a utility (throughput) by selecting some pre-defined strategies (i.e. either running CUBIC or BBR). We can then apply standard game-theoretic analysis to determine if a Nash Equilibrium exists. A Nash Equilibrium (NE) is a strategy distribution in which none of the players stand to gain anything by switching strategies given that the strategies of all the other players remain

the same. In the context of our problem, a Nash Equilibrium is a stable distribution of CUBIC and BBR flows such that none of the flows have any incentive to switch congestion control algorithms.

By solving the normal form game, we show that a NE distribution of CUBIC and BBR flows will exist in most networks with realistic configurations. Our model is also able to predict these NE distributions and we verify empirically with a large number of experiments that these predictions are accurate. Our contributions in this work can be summarized as follows:

1. We present a new mathematical model for predicting the bandwidth shares of competing CUBIC and BBR flows. We verify our model with extensive experiments to show it is demonstrably more accurate at predicting bandwidth shares than the current state-of-art model by Ware et al. [25].

2. With our model, we are able to predict that BBR's throughput gains over CUBIC steadily reduce as the proportion of BBR flows increases.

3. We adapt our model to determine this Nash Equilibrium (NE) distribution for multiple flows with the same RTT and show (both mathematically and empirically) that this NE is at a mixed distribution of CUBIC and BBR flows. We also show that our analysis seems to hold for BBRv2, an upgraded version of BBR that is currently being developed at Google [26].

In summary, we present a mathematical model for understanding how CUBIC and BBR compete with each other and use its results to predict the future of the Internet's congestion control landscape. We predict that while BBR, or perhaps BBRv2, is likely to become more popular, *it is not likely that the majority of the Internet will fully switch from CUBIC to BBR if better throughput is the key consideration.* We observed that there are diminishing returns in BBR's throughput advantage over CUBIC as the proportion of BBR flows increases. As more and more flows switch to BBR on the

Internet, CUBIC is likely to become more competitive until the point where there is no longer an incentive to switch between the two.

While a limitation of our model is that we assume that all flows have the same RTTs, we argue that this assumption is plausible, since a majority of today's Internet traffic is served via CDNs [10]. This means that it is quite plausible for the flows at local bottlenecks to have similar RTTs. We augment our model to predict a Nash Equilibrium distribution of CUBIC and BBR flows in a network and show that the ultimate mix of CUBIC and BBR on the Internet will mainly depend on the bottleneck buffer size and RTTs of the competing flows.

## 4.1 Modelling Interactions between BBR and CUBIC

In this section, we first describe a basic model that will allow us to predict the throughput shares when a CUBIC flow competes with a BBR flow at a common bottleneck. Next, we extend this model to a setting with multiple CUBIC and BBR flows. For simplicity, we will assume that all the competing flows have the same base RTT. We note that the majority of today's Internet traffic is served via CDNs [10], so it would not be uncommon for the majority of flows at local bottlenecks to have similar RTTs. In this light, we argue that this assumption is likely applicable in many contexts.

### 4.1.1 Background

**BBR Overview**. BBR (**B**ottleneck **B**andwidth and **R**ound-trip propagation time) is a rate-based congestion control algorithm proposed by Google in 2016 [22]. BBR estimates its share of the bottleneck bandwidth and the minimum round-trip time (RTT) of the path to regulate the TCP send rate. While doing so, BBR maintains a cap on its in-flight data at twice the bandwidth-delay product (BDP). BBR is implemented as a state machine with the following 4 states to make periodic and sequential measurements to

keep its estimates up-to-date:

1. **Startup**: To quickly learn the bottleneck bandwidth, BBR performs an exponential search by doubling its sending round every iteration. By doing so, it is able to find bottleneck bandwidth in $O(\log_2(\text{BDP}))$ round trips. BBR transitions to the Drain phase once it detects that the pipe is full by looking for a plateau in bandwidth estimates.

2. **Drain**: BBR drains packets it has accumulated in the queue during the aggressive Startup phase by reducing its in-flight packets to 1 BDP. Once it estimates that the queue is fully drained, but the pipe remains full, BBR enters the ProbeBW phase.

3. **ProbeBW**: BBR spends a majority of time in the ProbeBW state, and probes for bottleneck bandwidth using a technique called gain cycling. BBR undergoes a cycle of 8 RTTs: it first sends packets at 1.25 times the maximum receive rate to probe for extra bandwidth at the bottleneck. To compensate for this aggression. it sends packets at 0.75 times for the next RTT, and for the remaining 6 RTTs, it maintains its sending rate at the maximum receive rate.

4. **ProbeRTT**: BBR needs to empty the bottleneck buffer in order to accurately estimate the minimum RTT ($\text{RTT}_{min}$). BBR enters the ProbeRTT phase once every 10 seconds, reducing its in-flight packets to 4 in an attempt to drain the buffer.

**CUBIC Overview**. TCP CUBIC [16] is a loss-based algorithm, which means that when it encounters a packet loss, it shrinks its congestion window (cwnd) by a factor of 0.7. Otherwise, it increases cwnd using Equation (4.1).

$$\text{cwnd}(t) = C \times (t - K)^3 + W_{max} \tag{4.1}$$

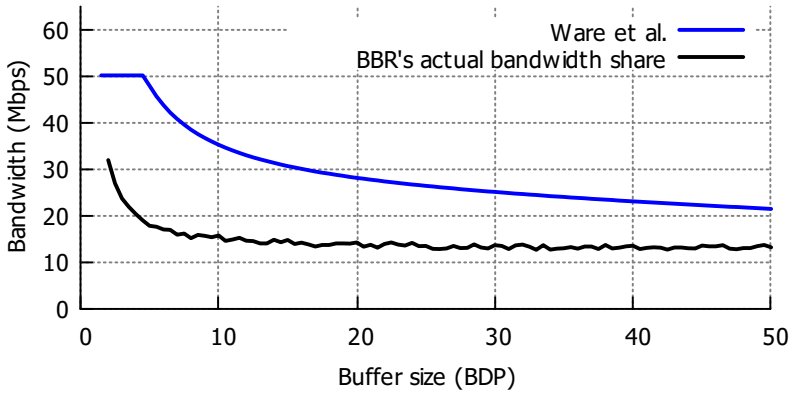



**Figure 4.1:** BBR bandwidth share for 50-Mbps bottleneck link at 40 ms RTT.

where $W_{max}$ is the window size just before the window is reduced and

$$K = \sqrt[3]{W_{max} \times (1 - \beta_{cubic})/C}$$

CUBIC's implementation in the Linux kernel sets $C = 0.4$, $\beta_{cubic} = 0.3$. For the purposes of our model, the key aspect of a CUBIC flow is that it reduces to 0.7 times $W_{max}$ after it experiences a packet loss.

### 4.1.2 Issues with Model by Ware et al.

To the best of our knowledge, the current best state-of-art model for the interactions between CUBIC and BBR is the model by Ware et al. [25]. Their model predicts the aggregate bandwidth of the competing BBR flows as

$$BBR_{frac} = (1 - p)\left(\frac{d - Probe_{time}}{d}\right) \tag{4.2}$$

where

$$p = \frac{1}{2} - \frac{1}{2X} - \frac{4N}{q} \tag{4.3}$$

$$Probe_{time} = \left(\frac{q}{c} + .2 + l\right)\left(\frac{d}{10}\right) \tag{4.4}$$

where $p$ represents the competing CUBIC flows' aggregate fraction of the bottleneck bandwidth $c$. $N$ is the number of competing BBR flows, $q$ is the average queuing delay in the bottleneck buffer and $X$ is the size of the bottleneck buffer in BDP. $l$ is the base RTT of all the flows and $d$ is the duration of the time the flows compete.

Their model predicts that BBR flows get a fixed share of the bottleneck bandwidth regardless of the number of competing CUBIC flows. While qualitatively this model does make some interesting observations (for example, Ware et al. were the first to highlight that BBR's in-flight cap is key in determining how it competes with other CUBIC flows), we have found their model to deviate significantly from actual BBR performance in experiments. We can see this from the results in Figure 4.1 for a simple experiment with a CUBIC flow competing with a BBR flow at a 50-Mbps bottleneck link, with each flow lasting for 2 minutes and having a base RTT of 40 ms.

With some analysis, we found that the inaccuracies in Ware et al.'s model [25] are due to the following assumptions:

1. The most problematic assumption is that the buffer is always full. This assumption was most likely made in the interest of simplicity since even the experiments in [25] demonstrate that this assumption is not true.

2. The second assumption is an over-simplification that BBR's RTT is the base congestion-free RTT plus $p \times q$, where $p$ is CUBIC's share of the link capacity and $q$ is the size of the bottleneck buffer. Since a flow's throughput is directly proportional to its average buffer occupancy, this calculation implies that CUBIC's *average* buffer occupancy is responsible for bloating BBR's $RTT_{min}$ estimate. However, since BBR measures the *minimum* RTT during the ProbeRTT phase, it stands to reason that this bloating of the $RTT_{min}$ should be affected by CUBIC's *minimum* buffer occupancy, and this is not the average buffer occupancy. This problem is further exacerbated when compounded with their first assumption

that the buffer is always full. This effectively fixes CUBIC's buffer occupancy and uses what is in reality CUBIC's *maximum* buffer occupancy to calculate BBR's bloated $RTT_{min}$ estimate.

We make none of these assumptions in our new model. When there are multiple CUBIC flows involved, we derive a confidence interval instead of assuming CUBIC's share of the link capacity to be fixed. This interval not only accurately predicts the actual average bandwidth of the competing flows, but also captures the stochasticity of these interactions caused by the varying degrees of synchronization between the CUBIC flows across trials and network conditions.

### 4.1.3 Basic 2-Flow Model

In this section, we describe a simple model that can predict the bandwidth shares of two competing CUBIC and BBR flows passing through a simple drop-tail queue. In this model, both the flows have the same base/minimum $RTT$ and compete at a bottleneck with link capacity $C$ and buffer size $B$.

**Assumptions**. Our model makes the following assumptions:

1. **The link is always fully utilized.** Since our analysis is centered around the bottleneck of the connection, we assume that the link is always utilized and there are always a non-zero number of packets in the buffer. To allow this assumption to hold in the presence of loss-based flows like CUBIC, we also assume the buffer is sufficiently sized [40] (at least 1 BDP) and that the CUBIC flows do not suffer any premature packet loss.

2. **The BBR flows always maintain 2 BDP packets in flight.** This assumption is in line with the observations made by Ware et al. [25], where they showed that BBR becomes `cwnd`-bound when it competes with CUBIC. The standard implementation of BBR has a `cwnd` twice its estimated BDP. To allow this assumption

**Table 4.1:** Model Notation

| Symbol | Meaning |
|---|---|
| $C$ | Bottleneck link capacity |
| $B$ | Bottleneck buffer size |
| $RTT$ | Base RTT (propagation delay) |
| $RTT^+$ | BBR's over-estimate of the RTT |
| $b_c$ | CUBIC's average buffer occupancy |
| $b_b$ | BBR's average buffer occupancy |
| $Q_d$ | Queuing delay |
| $b_{cmin}$ | CUBIC's minimum buffer occupancy |
| $b_{cmax}$ | CUBIC's maximum buffer occupancy |
| $\lambda_b$ | BBR flow's bandwidth |
| $\lambda_c$ | CUBIC flow's bandwidth |
| $\lambda_{cmin}$ | CUBIC's smallest bandwidth share |
| $\lambda_{cmax}$ | CUBIC's largest bandwidth share |
| $W_{max}$ | CUBIC's largest cwnd |

to hold, we consider buffers that are at least 1 BDP in size.

3. The packets from the two flows are uniformly distributed in the buffer.

4. The BBR flows are mostly loss-agnostic (This is true for BBRv1 [22])

5. **The reduction in BBR's bandwidth during the ProbeRTT phase is negligible.** We make this assumption because BBR's ProbeRTT phase lasts only for around 200 ms, which is negligible compared to its 10 s long ProbeBW phase.

6. All flows have the same base/minimum RTT.

The notation used for our model is listed in Table 4.1 for convenient reference.

**Modeling throughput**. Consider the bottleneck illustrated in Figure 4.2. At any given point in time, the throughput achieved by a flow is going to be the number of bytes it has in flight divided by its round-trip time. The in-flight bytes for both the CUBIC and BBR flows will be their respective buffer shares $b_b$ and $b_c$ plus the amount of data they have on the wire. Since both the flows see the same bottleneck queuing

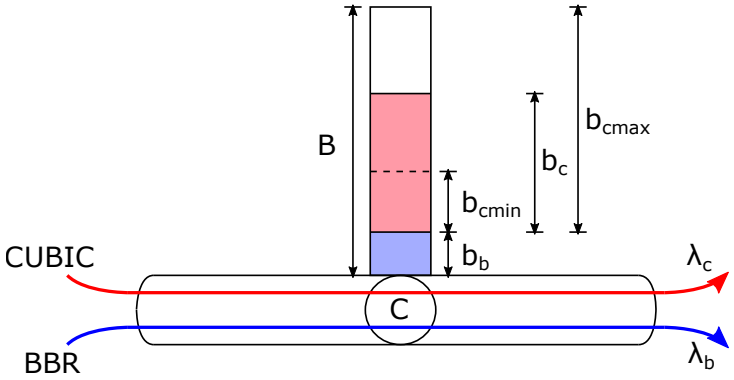



**Figure 4.2:** Network model.

delay because they share the same bottleneck, we can write their bandwidths as follows:

$$\lambda_c \leftarrow \frac{\lambda_c RTT + b_c}{RTT + Q_d} \tag{4.5}$$

$$\lambda_b \leftarrow \frac{\lambda_b RTT + b_b}{RTT + Q_d} \tag{4.6}$$

However, since we know that BBR is limited by its `cwnd` (which is capped by 2 × BDP) when competing with a CUBIC flow [25], we can rewrite Equation (4.6) as follows:

$$\lambda_b \leftarrow \frac{2\lambda_b RTT^+}{RTT + Q_d} \tag{4.7}$$

where $RTT^+$ is BBR's overestimate of the minimum RTT.

The queuing delay $Q_d$ is the total number of bytes both the flows have in the buffer divided by the rate at which these bytes are drained (link capacity $C$)

$$Q_d = \frac{b_b + b_c}{C} \tag{4.8}$$

**Relating $RTT^+$ to CUBIC's buffer occupancy.** BBR flows see a bloated $RTT$ because during its ProbeRTT phase, the bottleneck buffer is not completely empty and

there are still some CUBIC packets that have not drained. Then, $RTT^+$ can be written as:

$$RTT^+ = RTT + \frac{b_{cmin}}{C} \tag{4.9}$$

here, $b_{cmin}$ is the smallest number of packets a CUBIC flow has in the buffer during BBR's ProbeRTT phase. We will assume this to be CUBIC's buffer share when it backs off after a packet loss. Combining Equations (4.7) and (4.9) and simplifying, we get:

$$b_b + b_c = 2b_{cmin} + C \cdot RTT \tag{4.10}$$

where $b_b + b_c$ is effectively the average buffer occupancy. This is assuming the buffer size is greater than 1 BDP (or $C \times RTT$). We can use this result together with Equation (4.8) to rewrite Equation (4.5) as follows:

$$\lambda_c \leftarrow \frac{\lambda_c RTT + b_c}{2RTT + \frac{2b_{cmin}}{C}} \tag{4.11}$$

**CUBIC Minimum Buffer Occupancy**. $b_{cmin}$ is CUBIC's minimum buffer occupancy, which occurs when the CUBIC flow backs off after a packet loss. Since we know that CUBIC backs off to 0.7 times its maximum buffer occupancy after a packet loss, we can calculate $b_{cmin}$ as follows:

$$b_{cmin} = (0.7W_{max}) - (\lambda_{cmin} RTT) \tag{4.12}$$

where $\lambda_{cmin}$ represents the share of the bottleneck bandwidth the CUBIC flow receives during backoff, and $(\lambda_{cmin} \times RTT)$ is the number of bytes on the pipe after this backoff. Since the relevant buffer shares of BBR and CUBIC are an indicator of how much bottleneck bandwidth they are receiving at any point in time, we can write $\lambda_{cmin}$ as:

$$\lambda_{cmin} = \frac{b_{cmin}}{b_{cmin} + b_b} C \tag{4.13}$$

To calculate $b_{cmin}$ in Equation (4.12), we need to calculate the largest window size $W_{max}$ for the competing CUBIC flow. We estimate $W_{max}$ as follows:

$$W_{max} = b_{cmax} + \lambda_{cmax} RTT \tag{4.14}$$

where $b_{cmax}$ is simply the buffer occupancy a CUBIC flow has when it completely fills the bottleneck buffer:

$$b_{cmax} = B - b_b \tag{4.15}$$

and $\lambda_{cmax}$ is the bandwidth CUBIC gets when it puts $b_{cmax}$ packets in the buffer:

$$\lambda_{cmax} = \frac{b_{cmax}}{b_{cmax} + b_b} C \tag{4.16}$$

From the results from Equations (4.13) to (4.15), we can expand Equation (4.12) to calculate $b_{cmin}$ as follows:

$$b_{cmin} + \frac{b_{cmin}}{b_{cmin} + b_b} C \cdot RTT = 0.7 \times (B - b_b + \frac{B - b_b}{B} C \cdot RTT) \tag{4.17}$$

**Putting it all together**. Using the relation from Equation (4.10) and approximating $b_b + b_c = B$, we can write Equation (4.17) as:

$$\frac{B - C \cdot RTT}{2} + \frac{\frac{B - C \cdot RTT}{2}}{\frac{B - C \cdot RTT}{2} + b_b} C \cdot RTT = 0.7 \times (B - b_b + \frac{B - b_b}{B} C \cdot RTT) \tag{4.18}$$

Since $B$, $C$, and $RTT$ are known quantities, we can solve Equation (4.18) to get a BBR flow's buffer occupancy when competing with another CUBIC flow. This $b_b$ value can then be plugged into a simplified version of Equation (4.11) to solve for $\lambda_c$ and $\lambda_b$:

$$\lambda_c \left( RTT + \frac{2 b_{cmin}}{C} \right) = 2 b_{cmin} + C \cdot RTT - B_b \tag{4.19}$$

$$\lambda_b = C - \lambda_c \tag{4.20}$$

### 4.1.4 Modelling Multiple Flows

For a network with multiple CUBIC and BBR flows with the *same* RTT, we make the following observations:

1. First, the 2-flow model described in §4.1.3, only needs to take into account a CUBIC flow's maximum and minimum buffer occupancy. Hence, for a bottleneck with a total $N$ flows with $N_c$ of them being CUBIC flows and $N_b$ of them being BBR flows, we can model all the CUBIC flows as one *aggregate* CUBIC flow with a combined bandwidth $\hat{\lambda}_c$.

2. Similarly, we model all the BBR flows as another *aggregate* BBR flow with the bandwidth $\hat{\lambda}_b$. This is because we assume that the behavior of the aggregate BBR flow is practically identical to a single BBR flow when all the participating BBR flows will be `cwnd` bound. This is because we expect the BBR flows to be synchronized even while competing with other CUBIC flows and be fair to each other because of having similar RTTs [22].

3. Next, while $\hat{b}_{cmax}$ remains largely unchanged (since CUBIC flows always attempt to fill the buffer, regardless of whether there is one flow or many flows), $\hat{b}_cmin$ can vary for the *aggregate* CUBIC flow. This is because with multiple flows, depending on the loss pattern and start times of the competing flows, they can have varying levels of synchronization between the multiple CUBIC flows. We consider the maximum and minimum levels of synchronization separately.

In other words, to model a network with multiple CUBIC and BBR flows, we use the same model described in §4.1.3 but replace $\lambda_b$ and $\lambda_c$ with $\hat{\lambda}_b$ and $\hat{\lambda}_c$, respectively. The one key difference is that instead of using Equation (4.12), we consider 2 boundary cases:

1. **CUBIC Synchronized**. If all the CUBIC flows are synchronized, the lower bound for $\hat{b}_c min$ would be given by:

$$\hat{b}_{cmin} = (0.7\hat{W}_{max}) - (\hat{\lambda}_{cmin} RTT) \tag{4.21}$$

2. **CUBIC De-Synchronized**. On the other hand, if only one of $N_c$ CUBIC flows back-off at any time, i.e. all the flows are perfectly de-synchronized, the upper bound for $\hat{b}_{cmin}$ would be given by:

$$\hat{b}_{cmin} = \left(\frac{(N_c - 0.3)}{N_c}\hat{W}_{max}\right) - (\hat{\lambda}_{cmin} RTT) \tag{4.22}$$

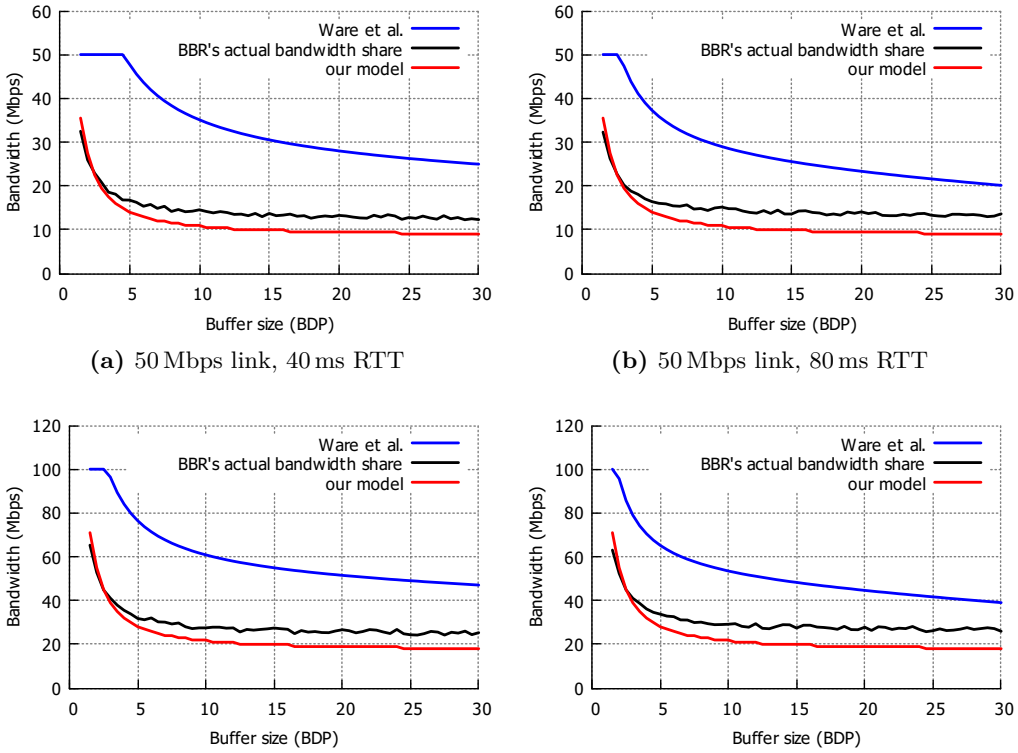
Solving the model for these 2 scenarios will provide us with a good estimate for the bandwidth shares of the BBR and CUBIC flows. In practice, we find that the empirical results are generally much closer to the case where CUBIC flows are synchronized (i.e. Equation (4.21)). The average bandwidths of the individual flows can be obtained as follows:

$$\lambda_c = \frac{\hat{\lambda}_c}{N_c} \tag{4.23}$$

$$\lambda_b = \frac{\hat{\lambda}_b}{N_b} \tag{4.24}$$

## 4.2 Model Validation

In this section, we validate our models in §4.1.3 and §4.1.4 using real experiments. Since all the flows in our model have the same RTTs, we normalize the buffer size to the bandwidth-delay product (BDP) in the graphs in this section to make it easy to compare across different network conditions.




**(a)** 50 Mbps link, 40 ms RTT **(b)** 50 Mbps link, 80 ms RTT

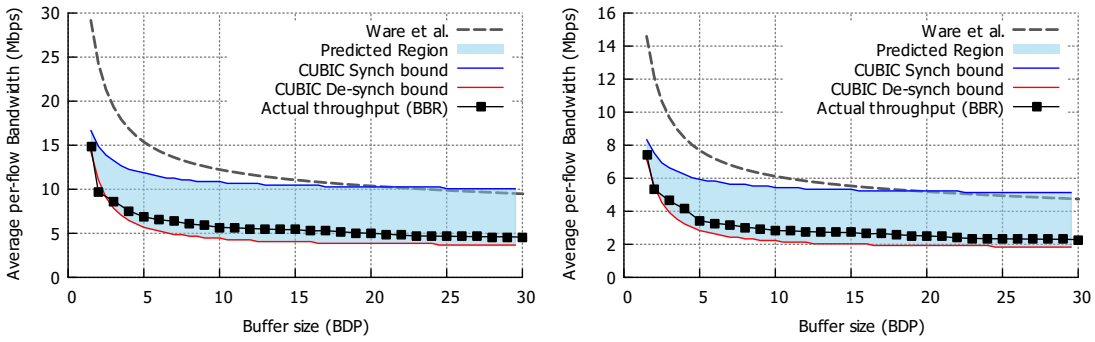
**(c)** 100 Mbps link, 40 ms RTT **(d)** 100 Mbps link, 80 ms RTT

**Figure 4.3:** Predicted throughput vs. actual throughput when a CUBIC flow competes with BBR.

### 4.2.1 Basic 2-Flow Model

We first evaluate the accuracy of our simple model that predicts the bandwidth shares of two competing CUBIC and BBR flows. To this end, we launched a CUBIC and BBR flow through a 50 Mbps bottleneck link. The buffer size was varied from 1 BDP all the way up till 30 BDP in steps of 0.5 BDP. We repeated the same experiment with a 100 Mbps bottleneck link.

In Figures 4.3, we plot the observed throughput share of the BBR flow against buffer size and compared it to the values predicted by Ware et al. [25]. Over a large range of buffer sizes, our model can predict the throughput achieved by a BBR flow competing with a CUBIC flow within 5% of the actual value. In contrast, Ware et al.'s model has




**(a)** 5 CUBIC vs 5 BBR **(b)** 10 CUBIC vs 10 BBR

**Figure 4.4:** Predicted vs. actual throughput of BBR and CUBIC flows when they compete at 100 Mbps bottleneck with 40 ms RTT.

an error of at least 30% error, and this is for shallower buffers. As discussed in §4.1.2, this is because they made assumptions that do not hold in shallow to moderately sized bottleneck buffers.

An interesting observation from Figures 4.3 is that both the predictions of our model as well as Ware et al.'s model are relatively stable across different link speeds and RTTs, i.e. the plots for other link speeds and RTTs have a similar shape and error.

### 4.2.2 Multiple Flows

To evaluate the accuracy of our model for multiple flows, we launched 10 flows (5 CUBIC flows vs. 5 BBR flows) through a 100 Mbps bottleneck link with all the flows having a base RTT of 40 ms. The buffer size was varied from 1 BDP to 30 BDP in steps of 1 BDP. We repeated the same experiments with 20 flows (10 CUBIC flows vs. 10 BBR flows). All the flows in these experiments were started simultaneously and lasted for 2 minutes.

In Figures 4.4, we plot the per-flow average throughput against the confidence interval predicted by our multi-flow model. We see from our results that the per-flow average throughput for BBR falls within the confidence interval predicted by our model. In fact,

we found the measured per-flow average throughput to be very close to the boundary where the CUBIC flows are de-synchronized. We checked the traces of our experiments and verified that the CUBIC flows were indeed generally not found to be synchronized in these experiments. It should be noted here that while it *looks* like the model by Ware et al. matches our model's 'Synch' bound in deeper buffers, it is not because their model assumes that the competing CUBIC flows are perfectly synchronized. In fact, they assume that the buffer occupancy of the competing loss-based flows does not vary at all - therefore CUBIC flows being either synchronized or not has no impact on the assumptions of their model.

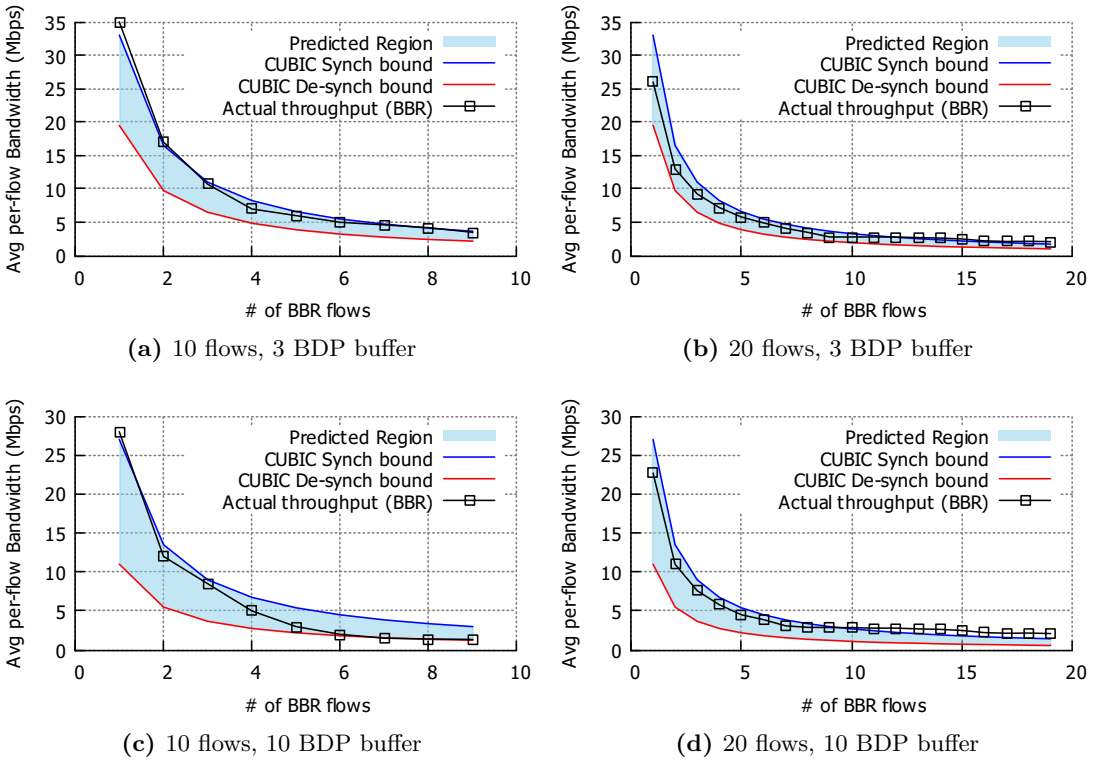
### 4.2.3 Varying the Proportion of Flows

Since our goal is to understand the evolution of the Internet's congestion control landscape, it is also important to understand how BBR's average per-flow bandwidth share will change as the share of BBR flows at the bottleneck increases.

To this end, we launched 10 flows through a 100 Mbps bottleneck for buffer sizes of 3 and 10 BDP. Each of these flow were either CUBIC or BBR. Over multiple runs, we increased the share of the flows running BBR and measured the average bandwidth achieved by BBR flows over a duration of 2 minutes. All the flows were launched simultaneously. This experiment was then repeated for 20 flows.

In Figures 4.5, we plot the average per-flow throughput against the number of BBR flows (out of 10 or 20). We see that the measured average per-flow throughputs indeed fall within the upper and lower bounds predicted by our multi-flow model. We note that in some cases, the measured values are closer to the boundary where the CUBIC flows are synchronized and in other cases where they are not. Again, we checked the traces of our experiments and verified that the behavior of the CUBIC flow did correspond to the closer line in the experiments.

The most important takeaway from these experiments is that BBR's average per-flow




**(a)** 10 flows, 3 BDP buffer

**(b)** 20 flows, 3 BDP buffer

**(c)** 10 flows, 10 BDP buffer

**(d)** 20 flows, 10 BDP buffer

**Figure 4.5:** The diminishing throughput returns for BBR as its share at the bottleneck grows.

bandwidth reduces as the proportion of flows running BBR at the bottleneck increases. This suggests that as more and more users on the Internet start using BBR as their congestion control algorithm, the throughput advantage currently enjoyed by BBR over CUBIC will be reduced. At some stage, the average throughput for BBR could fall below that of CUBIC! We will use this observation in §4.3 to show that a Nash Equilibrium distribution of CUBIC and BBR must exist when multiple flows with the same base RTT compete at a common bottleneck link. This trend also suggests that as more BBR flows join the bottleneck, their collective buffer occupancy increases only sub-linearly.

## 4.3 Applying Game Theory

In this section, we apply game theory using our model in §4.1.4 to predict how the Internet might evolve in the near future.

Performance is multi-faceted and context-dependent. For simplicity, we focus on throughput and assume the flows will choose the congestion control algorithm that offers them the highest throughput, since better throughput is often cited as the reason for switching congestion control algorithm [35, 36, 92].

We model users (websites) as agents that currently choose either CUBIC or BBR as their congestion control (CC) algorithm. If a user can enjoy higher throughput by switching to the other CC algorithm, then there would be an incentive to switch. A Nash Equilibrium (NE) distribution of CUBIC and BBR flows occurs when none of the users have *any* incentive to switch, either because doing so will not result in increased throughput, or worse, will result in lower throughput.

We show that *a NE must exist when flows with similar RTTs compete at a bottleneck* and discuss how this analysis applies to more complicated setting with other (non-BBR) congestion control algorithms and more complex utility functions. Networks with flows with different RTTs remain future work.

### 4.3.1 NE for flows with similar RTTs

Consider a network with $n$ flows sharing a common bottleneck, each running either CUBIC or BBR as their congestion control algorithm. In this network, we define a given distribution of CUBIC and BBR flows to be a NE, if none of the flows have any incentive to switch from CUBIC to BBR or vice versa to achieve better performance. Since all the flows have the same RTT and are essentially symmetric, there are a total of $n + 1$ possible distributions for $n$ flows. For each of these distributions, we can measure the average bottleneck bandwidth received by all the BBR flows and plot them on a graph

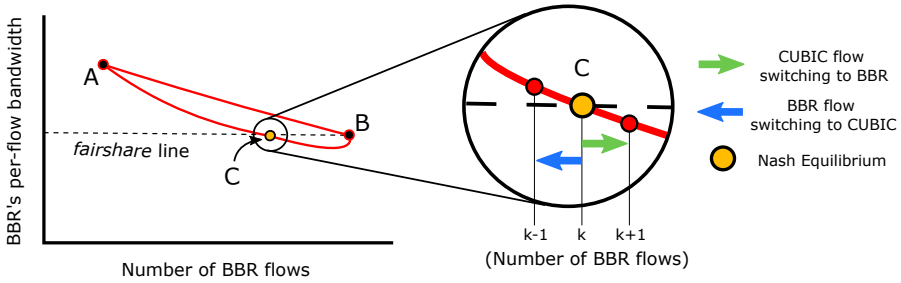



**Figure 4.6:** Nash Equilibrium for flows with similar RTTs.

as shown in Figure 4.6. The dotted line on this graph represents the *fair-share* line, where the average bandwidth of the BBR flows is equal to the fair-share bandwidth (i.e. link capacity divided by the total number of flows).

It has been observed that when a small number of BBR flows compete with a large number of CUBIC flows, they are able to get a disproportionately large share of the bottleneck bandwidth [25]. Given this result, we know that there exists a distribution (with a small number of BBR flows) that lies above the *fair-share* line in Figure 4.6, which we label as *A*. We also know that when *all* the flows at the bottleneck run BBR, they will take up all the bottleneck bandwidth. The average bandwidth for the BBR flows will then, by definition, be the fair-share bandwidth. We can use this observation to plot point B in Figure 4.6. We expect all the distributions between these two distributions to lie on a line connecting points A and B.

Next, we consider the distributions along the line from A to B. There are one of 2 possibilities: (i) either the line AB is always lies above the fair-share line, or (ii) the line AB intersects the fair-share line at some point C. We note that for the points above the fair-share line, BBR flows will have on average higher throughput; for the points below this line, the CUBIC flows will have on average higher throughput. What this also implies is that for the points above the fair-share line, some CUBIC flow will have an incentive to switch to BBR; the converse would be true for the points below the fair-share line.

**Case 1: AB is above the fair-share line**. For any point between A and B, some CUBIC flow would be incentivized to switch to BBR. As more flows switch from CUBIC to BBR, we move along the AB line until we reach B. Point B, where all flows are BBR, is then the Nash Equilibrium distribution. This is because no flows have the incentive to switch to CUBIC because it would result in a loss of throughput.

**Case 2: AB intersects the fair-share line at C**. We claim that C is a Nash Equilibrium distribution. To understand why we can zoom in and examine what happens at C (Figure 4.6). There are 2 possibilities:

1. **A CUBIC flow can switch to BBR**. This would correspond to the current state of the network moving to the right of C. This means the average throughput of the BBR flows will drop below that of the CUBIC flows. Therefore, a CUBIC flow will never switch to BBR.

2. **A BBR flow can switch to CUBIC**. On the other hand, a BBR flow switching to CUBIC would move the current state of the network to the left of C, which will result in the average throughput of the CUBIC flow dropping below that of the BBR flows. So again, this switch is also not tenable.

Since it is not tenable to move either left or right, point C is a stable Nash Equilibrium distribution.

**Estimating the Nash Equilibrium distribution**. Following the observations in §4.1.4, the Nash Equilibrium distribution exists when the combined bandwidth of all the BBR flows intersects with the *fair-share* line. In other words, the Nash Equilibrium distribution is the value for $N_b$ at which:

$$\lambda_b = \frac{\hat{\lambda}_b}{N_b} = \frac{C}{N} \tag{4.25}$$

We can use Equation (4.25) in conjunction with Equations (4.20) and (4.24) of our throughput model to predict the Nash Equilibrium distribution of CUBIC and BBR in

any given fixed capacity network where all the flows have the same base RTT!

### 4.3.2 Other Congestion Control Algorithms

The results for CUBIC and BBR in §4.3.1 are based on *only* two assumptions: (i) BBR is able to obtain a disproportionately large share of the bottleneck bandwidth for at least one distribution; and (ii) when all the flows are BBR, the BBR flows will take up the available bottleneck bandwidth. The latter is self-evident if we replace BBR with another congestion control algorithm. If we can show that the former property is also true for another congestion control algorithm X, then an NE distribution of CUBIC and X flows *must* also exist when flows with similar RTTs compete at a bottleneck.

To verify if the former property holds for the following congestion control algorithms that were proposed after BBR: (i) BBRv2 [26], (ii) Copa [93], and (iii) PCC Vivace [55], we launched an experiment with 10 flows in a network with a 100 Mbps bottleneck and a 2 BDP bottleneck buffer for each algorithm X. All flows in this network ran either CUBIC or X. We measured the per-flow average throughputs for all 11 possible distributions of CUBIC and X flows.

We plot the average per-flow throughput against the number of non-CUBIC (X) flows in Figure 4.7. We found that PCC-Vivace [55], BBR [22] and BBRv2 [26] are able to get a disproportionately large share of the bottleneck bandwidth when there are a small number of flows. On the other hand, Copa [93] obtains lower average throughput for all congestion control algorithm distributions. Therefore, we expect a Nash Equilibrium distribution to exist for BBRv2 and PCC Vivace as well, but perhaps not for Copa.

We note that our result is valid only for the case of two competing algorithms competing at a common bottleneck. Scenarios where more than two CC algorithms compete at a common bottleneck remain future work.

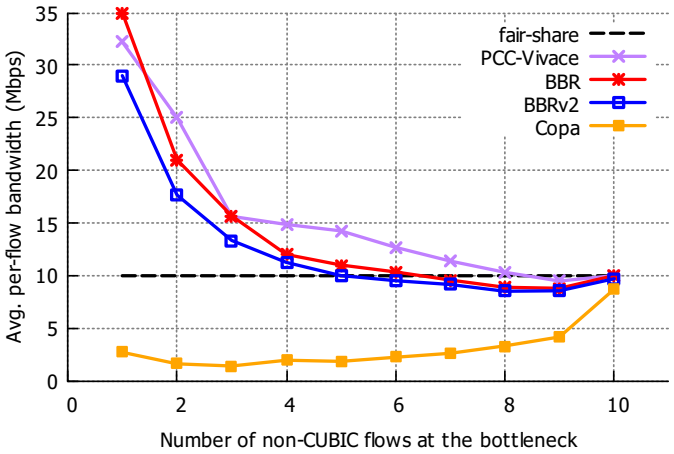



**Figure 4.7:** Combined bandwidth vs. number of flows for various congestion control algorithms.

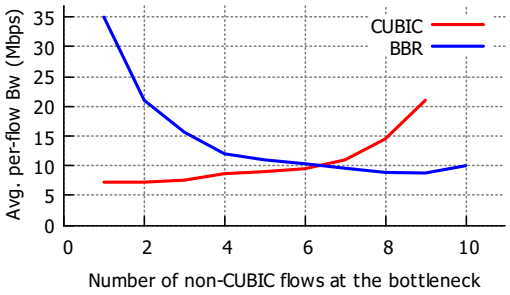



**Figure 4.8:** Average per-flow throughput.

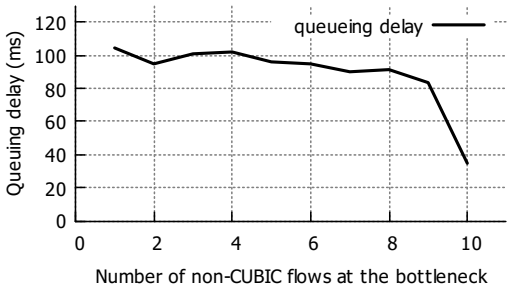
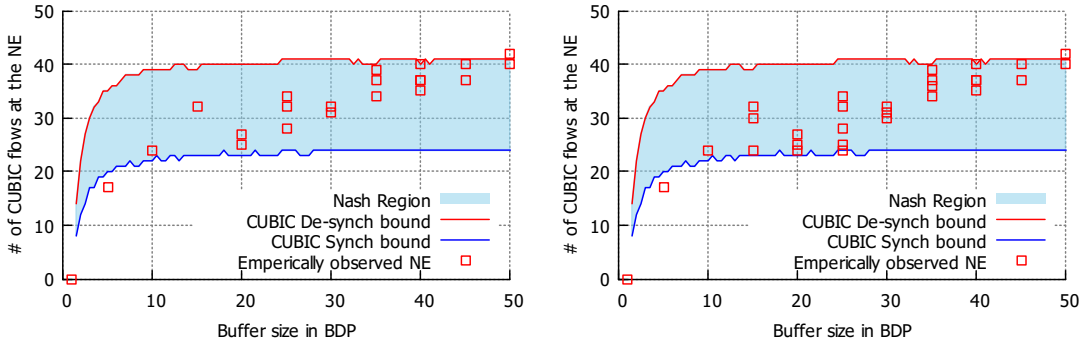



**Figure 4.9:** Average queuing delay

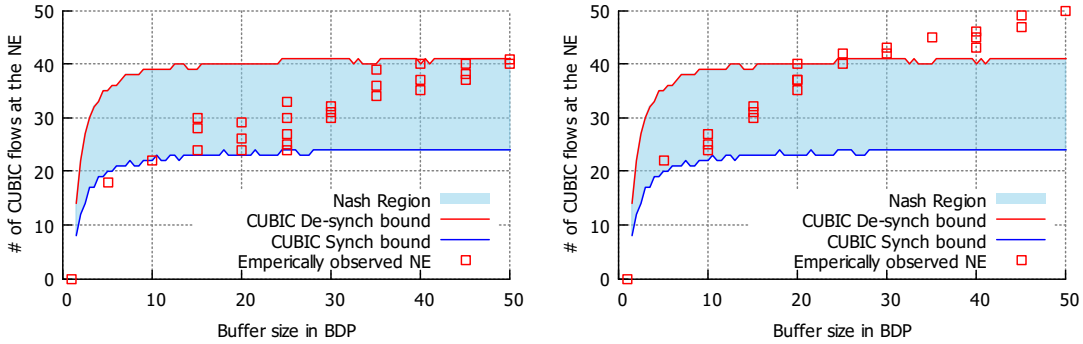
### 4.3.3 Complex Utility Functions

In the real world, it is likely there are senders that care not only about throughput but also delay. For video streaming, the metric of import would become even more complicated. However, we argue that even in such scenarios, throughput will likely still be the metric that drives senders to switch between CUBIC and BBR.

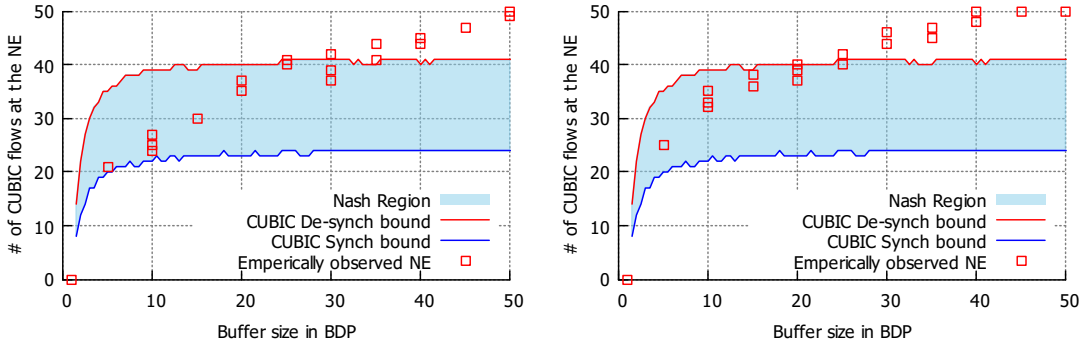
To illustrate this, we plot the average throughput per algorithm and average queuing delay of a 10 flow evolution experiment discussed in §4.3.2. The 10 flows in this experiment pass through a bottleneck link of 100 Mbps, with a 2 BDP buffer and 40 ms base RTT. The two lines in Figure 4.8 represent the average throughput and CUBIC and BBR




**(a)** 50 Mbps link, 20 ms link **(b)** 100 Mbps link, 20 ms link




**(c)** 50 Mbps link, 40 ms link **(d)** 100 Mbps link, 40 ms link




**(e)** 100 Mbps link, 40 ms link **(f)** 100 Mbps link, 80 ms link

**Figure 4.10:** Predicted Nash Equilibrium vs. observed Nash Equilibria points for a bottleneck with 50 flows.

flows receive in a given trial. The single line in Figure 4.9 represents the average queuing delay, which is a metric shared between the flows regardless of the congestion control algorithm they run in that trial. What these graphs illustrate is that even though both throughput and delay are dependent on the congestion control algorithm distribution at the bottleneck, throughput is likely the only metric that is asymmetric enough to drive flows to switch between CUBIC and BBR. For a flow that cares about queuing delay, a switch between CUBIC and BBR likely leads to a marginal gain in utility (since it is clear from Figure 4.9 that increasing the proportion of BBR flows has hardly any effect on queueing delay unless all flows were BBR.)

Therefore, we conjecture that under simple utility functions that are linear combinations of throughput and delay, a Nash Equilibrium distribution will still exist. This is because we expect the decision to select between different congestion control algorithms to still be dominated by the throughput gains in such settings. That said, it is still unclear how the flows will react where all the participating flows have drastically different utility functions. Investigating whether Nash Equilibria will exist and what the distributions would look like for complex utility remains future work.

### 4.3.4 Experimental Verification

In this section, we present the results of our testbed experiments to validate the accuracy of the NE distributions predicted by our results in §4.3.1.

**Methodology**. For each network setting, we run 10 trials of all the $n + 1$ possible combinations of the $n$ senders running either CUBIC or BBR. In each trial, the senders send data for 2 minutes and we record their average per-flow throughput. To identify the NE, we enumerate all the combinations and check if there is any combination such that no individual flow in that combination can achieve higher throughput if it switches to the other TCP variant (all other flows remaining fixed). It is common for multiple distributions to satisfy this condition.

**Nash Equilibria Found**. We plot the results of NE found for 50 competing flows in Figures 4.10a - 4.10f. The bottleneck bandwidth was set to 100 Mbps and 50 Mbps with the buffer size varying from 0.5 to 50 times the BDP. All these flows had the same base RTT which were 20, 40, and 80 ms across different trials. All the NE found empirically were in the interval predicted by our model, except those at high BDPs. BBR is not `cwnd`-limited in these regions and hence our model does not work well, which explains why the actual NE deviates from our predictions.

Aside from the trend that there tend to be more CUBIC flows at the NE in deeper buffers as compared to shallower buffers, the results in Figures 4.10a - 4.10f present two more interesting trends. The first is that we found *multiple* NE over different iterations of the same experiment. This is down to the throughput gains from switching between CUBIC and BBR being marginal around the Nash Equilibrium distribution. Therefore, any stochasticity across the trials can result in the NE shifting to neighboring distributions. We observe this phenomenon as multiple NE distributions across multiple trials.

The other trend is that when the buffer size is normalized by the BDP, the region predicted by the model is exactly the same regardless of the base RTT or the bottleneck link capacity. This is evident in Figures 4.10, where the predicted regions in all the six different network settings tested have the exact same shape. The empirically observed NE also follow this trend, with all of them roughly following the same curve across experiments with different base RTTs and link speeds. This would suggest that where the NE lies does not independently depend on either the link capacity $C$ or the $RTT$, but the BDP. This makes sense, since the key indicator in what bandwidth competing BBR and CUBIC flows get in §4.1 is the extra packets that BBR keeps in the buffer, which depends on the BDP ($C \times RTT$). We saw similar trends when we repeated these experiments for bottlenecks with 25 flows.

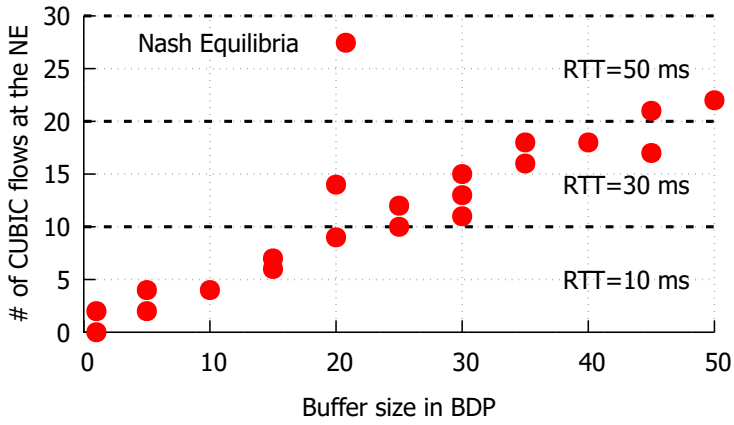



**Figure 4.11:** Nash Equilibrium distributions between CUBIC and BBR flows with different RTTs.

### 4.3.5 Flows with different RTTs

While our model assumes that all the flows have similar RTTs, the RTT distribution on the Internet can be quite diverse [94]. Even though our model cannot easily be extended to a multi-RTT setting, we conducted experiments to investigate the NE for flows that had different RTTs. In particular, we simultaneously launched 30 flows comprising of three groups of 10 flows with RTT of 10 ms, 30 ms, and 50 ms respectively. These flows shared a 100 Mbps bottleneck link with buffer sizes varying as multiples of the BDP (bandwidth-delay product) of the flow with the shortest RTT. We ran all possible combinations of CUBIC and BBR flows for these flows for three trials and then computed the Nash Equilibrium distributions just like in our previous experiments. The purpose of this experiment is not to quantitatively verify the predictions of our model, but to verify that Nash Equilibrium distributions of CUBIC and BBR can exist in multi-RTT networks as well. We plot the results in Figure 4.11. We noticed two key trends in NE between flows with different RTTs:

1. **Existence of NE.** For all the buffer sizes tested, we were able to find at least one Nash Equilibrium distribution of CUBIC and BBR flows. In many instances, there

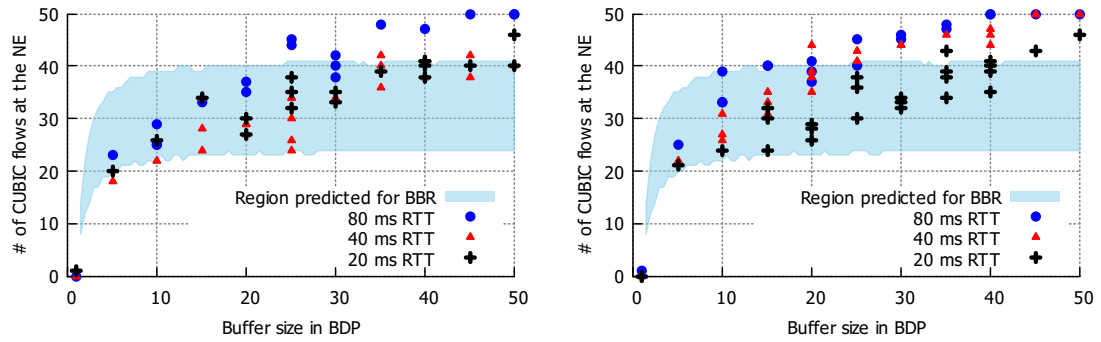
were multiple NE distributions across trials, but all these distributions roughly had the same percentage of flows running CUBIC.

2. **Nature of the NE.** In all Nash Equilibrium distributions, all the flows choosing to run CUBIC were the flows with the shortest RTTs. In other words, if a Nash Equilibrium distribution had 15 out of 30 flows running CUBIC, these 15 flows would comprise of all the ten flows with 10 ms RTT, and five 30 ms RTT flows.

Our results suggest that flows with larger RTTs benefited by switching to BBR more than flows with shorter RTTs; the reverse is true for CUBIC. This is expected from our understanding of RTT-fairness for CUBIC and BBR. Loss-based congestion control algorithms like CUBIC in general tend to favor flows with shorter RTTs [95], because flows with shorter RTTs are able to get quicker feedback and probe for bandwidth more frequently. With BBR, the opposite is true, i.e., flows with larger RTTs obtain a larger share of the bottleneck bandwidth than flows with smaller RTTs [96], because BBR flows become `cwnd`-limited and maintain a buffer share directly proportional to their RTT. When CUBIC and BBR flows with different RTTs compete, it is only natural that these two opposing trends would complement each other to give rise to NE distributions with shorter-RTT CUBIC flows and longer-RTT BBR flows.

### 4.3.6 BBR Predictions applied to BBRv2

Google is working to replace BBR with BBRv2 in the near future [26]. BBR has been found to be unfair to CUBIC flows in smaller buffers and can take up to half the total available bandwidth at the bottleneck regardless of flow-wise share [25]. To mitigate this issue, BBRv2 is designed to be a less aggressive alternative to BBR. At a high level, BBRv2 behaves like BBR, but because it has a variable `cwnd`, it is able to react to packet loss. We repeated the experiments in §4.3.4 for BBRv2 to determine if Nash Equilibrium distributions exist, and if so, how they would compare to our predictions for BBR.




**(a)** 50 flows, 50 Mbps **(b)** 50 flows, 100 Mbps

**Figure 4.12:** Nash Equilibrium distributions between competing CUBIC and BBRv2 flows.

Our results in Figures 4.12 suggest that multiple Nash Equilibria also exist when CUBIC and BBRv2 flows compete at a common bottleneck. This is in line with our observations in §4.3.2. Because BBRv2 is less aggressive than BBR, the Nash Equilibria for BBRv2 generally had a higher share of CUBIC flows for the same buffer size when compared to BBR (see Figure 4.7). Our results also suggest that our current model for BBR works well for BBRv2 when the RTT is relatively small. Augmenting the model to improve throughput predictions for BBRv2 remains future work.

## 4.4 Discussion

**Nash Equilibrium for networks with different RTTs**. Nash Equilibria were earlier observed in settings where the competing CUBIC and BBR flows have different base RTTs [2]. However, one limitation of our model is that the analysis presented in this chapter does not apply to networks that have flows with different RTTs. This is because with different RTTs, flows will no longer be symmetric and the state space with all the possible distributions will grow exponentially. Our proof in §4.3.1 requires that we linearize the state space of all the possible CCA distributions in a way that the two conditions discussed in §4.3.2 are met. We have not found a way to do so for a network

where flows have different base RTTs. While we have results that suggest that Nash Equilibria generally exist for networks with different RTTs (see §4.3.5), we have not been able to extend the proof in §4.3.1 to this setting.

**Implications on Internet Buffer Sizing**. Router buffer sizing is a long-standing problem [40, 89, 90]. Rules of thumb have been derived over the years and trends have been moving towards "tiny" buffers [90], to avoid Buffer Bloat [97]. However, given that BBR keeps 2×BDP packets in flight, CUBIC flows may face starvation if BBR becomes the dominant TCP variant on the Internet.

**Model Performance for large numbers of flows.** While our experiments have validated our model for up to 50 concurrent flows, 50 is still orders of magnitude smaller than the number of concurrent flows passing through the bottleneck links on the Internet. It remains to be seen how our predictions will scale to the Internet. However, we see no reason why qualitatively our predictions would not apply to larger networks with hundreds of concurrent flows.

**More diverse workloads and more complicated metrics.** One gap in the evaluation of our model is that we have only run experiments for long flows. Real Internet workloads are more diverse, and consist of not only long flows, but also chunky video traffic, short flows generated by ad services, and latency-sensitive traffic from live streaming and video calling. Different application traffic is likely to care about more complex metrics than just throughput. It is also unlikely that the mathematical model presented here (which models the steady state behavior of CUBIC and BBR) will be able to accurately replicate the interactions between short flows. Improving our model and evaluation to handle more diverse and realistic workloads remains future work.

**Forced synchronization among CUBIC flows**. We observed that the actual bandwidth share for the NE found is often closer to the CUBIC-Synched bound of our model (see Figures 4.5). This suggests that for multiple competing CUBIC and BBR flows, the CUBIC flows can get synchronized. We believe that this is likely because

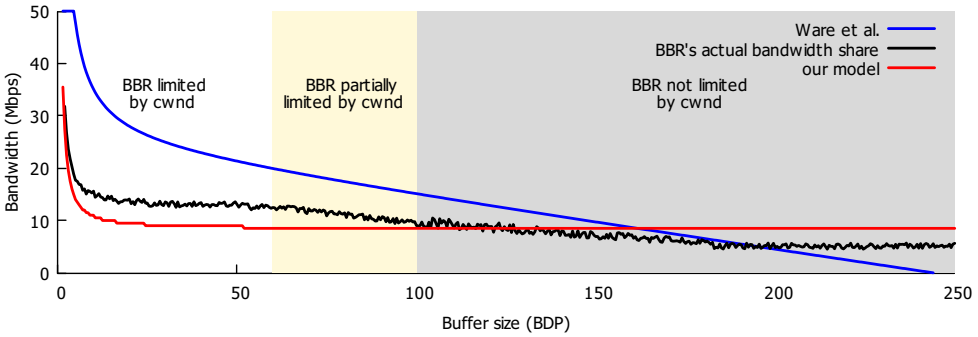



**Figure 4.13:** Performance of the model in ultra-deep (>100*BDP) buffers.

when all the BBR flows transition (together) from ProbeRTT to ProbeBW, they collectively add many packets to the buffer causing the buffer to overflow and the majority of the CUBIC flows to experience packet losses at the same time, and they end up synchronizing.

**Poor Performance for ultra-deep buffers**. We do not expect the model described in §4.1 to be applicable in very deep buffers (more than 100 times the BDP). This is because in such buffers, BBR is not always `cwnd` limited. When BBR exits its ProbeRTT phase, it starts with keeping only 1 BDP packets in flight. However, as the flow scavenges more and more bandwidth during its periodic ProbeBW cycles, it slowly grows the number of packets it has in flight till it is capped by the `cwnd`. Since these ProbeBW cycles happen every 8 RTTs, and in deep buffers these RTTs are much larger than in shallow buffers, BBR becomes `cwnd` limited at a much slower rate [25]. Therefore, our model would always overestimate how well BBR would perform against CUBIC flows in deep buffers.

To demonstrate this, we conducted a simple experiment with a single CUBIC flow competing with a single BBR flow at a 50-Mbps bottleneck link, with each flow lasting for 2 minutes and having a base RTT of 40 ms. We then plotted the bandwidth received by the BBR flow in buffer sizes ranging from just 1 BDP up to 250 times the BDP. We can see the extent of our model's over-estimation in ultra-deep buffers in Figure 4.13. It

is clear that BBR's average throughput gradually decreases as the buffer size increases beyond 60 times the BDP. The actual throughput will dip below our model's predicted value when the buffers are deeper than 100 times the BDP. We verified that the BBR flow was not `cwnd`-limited at these buffer ranges in these experiments.

**Assumption of 2 BDP packets in flight**. Our model in §4.1 assumes that BBR flows always maintain 2 BDP worth of packets in flight. In practice, the actual number of packets would vary between 1 and 2 BDP, since each ProbeBW phase starts with approximately 1 BDP of packets. As the ProbeBW phase progresses, the overestimation of the minimum RTT will cause BBR to increase the number of packets in flight, and the rate of increase depends on the RTT. For higher RTT values, the number of bandwidth probes in the 10-second ProbeBW phase will be smaller and so the average number of packets in flight will be closer to 1 BDP. Our assumption of 2 BDP packets in flight allows us to achieve good accuracy while keeping the model simple. Nevertheless, it is likely that it is possible to improve our model by estimating the number of packets in flight during the ProbeBW phase more accurately.

**Tightening the bounds.** Currently, our model provides bounds on a the bandwidth share competing CUBIC and BBR flows can get when they share a bottleneck. There is scope for making these bounds tighter than they currently are if we can definitively predict the degree of synchronization between the CUBIC flows running through the bottleneck. We have verified that when the NE lies close to the CUBIC Synch and CUBIC De-synch bounds, it is because in those experiments, the CUBIC flows did in fact synchronize or de-synchronize. Unfortunately, it is hard to predict how well CUBIC flows will synchronize, which makes tightening the bounds of our current model a challenge. That said, we have empirically observed that CUBIC flows tend to de-synchronize more in deeper buffers compared to shallower buffers.

## 4.5 Summary

In summary, we showed that BBR sees diminishing returns in its throughput advantage over CUBIC as the proportion of BBR flows increases. As BBR flows become more numerous, the average per-flow bandwidth of the BBR flows will drop. This dynamic suggests that for most realistic network scenarios, there will likely always be a Nash Equilibrium distribution of CUBIC and BBR flows, where no flows have any incentive to switch. We thus make a bold prediction that it is unlikely that BBR will completely replace the CUBIC flows on the Internet in the near future. Even as BBR (or BBRv2) continues to grow in dominance, we believe that some flows always continue to be CUBIC for some time to come.

Chapter 5

# Containing the Cambrian Explosion in QUIC Congestion Control

As discussed in Chapter 1, the adoption of QUIC represents a major development in Internet Congestion Control. This is because since it is implemented in the user space, it represents an easy way for developers to test and deploy new and modified CCAs. Being cognizant of this development, we inspect how developers are implementing CCAs in the current crop of QUIC stacks, and how they compare with the existing TCP CCA implementations that they try to replicate. In this chapter, we investigate how QUIC CCA implementations risk making the Internet's CCA landscape even more heterogeneous than before.

QUIC was first introduced by Google in 2015 to address the limitations of TCP and to add security enhancements to HTTP [60]. It has since seen rapid adoption and has been designated as the default transport stack for HTTP3. QUIC is estimated to already constitute some 30% of downstream traffic in EMEA (Europe, Middle East, and Africa) and 16% of downstream traffic in North America [98].

While the original motivation behind QUIC was to improve security, QUIC's protocol designers also used this opportunity to redesign many legacy aspects of TCP, like

the handshake, and to introduce multi-streaming. However, for congestion control, most QUIC developers chose to be conservative and re-implemented standard congestion control algorithms (CCAs) like CUBIC [99], Reno, and BBR [100]. This is not surprising since these algorithms are well understood and predictable, thanks to many years of deployment on the Internet. Their stability properties are especially important since QUIC already takes up a significant share of today's Internet traffic [98].

In general, for compatibility with existing CCAs, we expect these QUIC CCA implementations to have the following two properties:

1. **Behave like their kernel counterparts.** Since the entire motivation behind re-implementing standard congestion control algorithms is to achieve predictability and stability, we want QUIC implementations to resemble their corresponding kernel implementations. In particular, we expect QUIC CCAs to not only achieve delays and throughputs similar to their kernel counterparts but also to interact with existing CCAs qualitatively in the same way.

2. **TCP friendliness.** We want new QUIC congestion control implementations to co-exist well with existing Internet traffic and be friendly towards other QUIC and TCP kernel implementations. In particular, it would be disastrous if new implementations cause significant degradation or starvation of existing flows.

In this chapter, we investigated how closely the CCA implementations for the 11 popular open-source QUIC stacks listed in Table 5.1 adhere to these expectations. These stacks were selected because they are all open source, stable, and deployed on the Internet. While it is easy to evaluate TCP-friendliness and qualitative interactions between different implementations, it is much harder to define the *ideal* behavior of a congestion control algorithm. Even if we use a given implementation as the standard reference for a congestion control algorithm, it is not entirely straightforward how we can quantify how well a new implementation conforms to this reference implementation. For example, a

**Table 5.1:** List of QUIC/TCP stacks studied and their available CCAs.

| Organization | Stack | Version/Commit Hash | CUBIC | BBR | Reno |
|---|---|---|---|---|---|
| Linux kernel | TCP | Linux 5.13.0-44-generic | ✓ | ✓ | ✓ |
| Facebook | `mvfst` [101] | 65a9c066e742620becacc99b7c0ca86200e6a4c4 | ✓ | ✓ | ✓ |
| Google | `chromium` [102] | 82a3c71cf5bf2502d5ad90489fe20ce8f8cb3fab | ✓ | ✓ | ✗ |
| Microsoft | `msquic` [103] | e6110b62cd8e0d84e6436bde2504e6bc0702921a | ✓ | ✗ | ✗ |
| Cloudflare | `quiche` [104] | 9dfeaafb625b08760218def7beb8db133e3f50cb | ✓ | ✗ | ✓ |
| LiteSpeed | `lsquic` [105] | 108c4e7629a8c10b9a73e3d95be0a1652e620fb9 | ✓ | ✓ | ✗ |
| Go | `quicgo` [106] | 424a66389c01d10678bfb980cfe6faa8524b42b6 | ✓ | ✗ | ✓ |
| H2O | `quicly` [107] | d44cc8b21ed0d27ab6d209d0775c3961b2f89f38 | ✓ | ✗ | ✓ |
| Rust | `quinn` [108] | f86dd7596d4df31370b294c35501cec8da48b393 | ✓ | ✗ | ✓ |
| Amazon Web Services | `s2n-quic` [109] | 17826d9df1c59903beadd1733bbe79ed7d67647e | ✓ | ✗ | ✗ |
| Alibaba | `xquic` [110] | 00f622885d91e02c879f8531bc04af7a584faed4 | ✓ | ✓ | ✓ |
| Mozilla | `neqo` [111] | 07c2019988a8f0a37f87cbd90f95e906e7b53258 | ✓ | ✗ | ✓ |

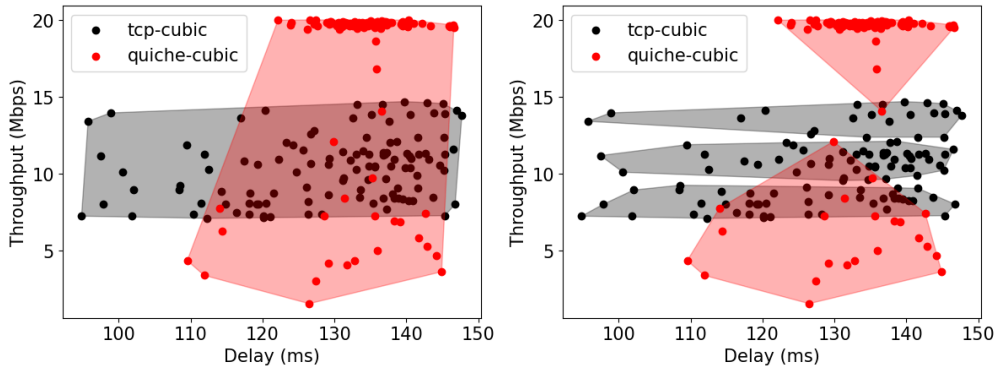
simple fairness-based measure of conformance would not be desirable as it would fail to capture the algorithmic nuances between different CCA implementations. An ideal metric for capturing the similarity between two CCA implementations should capture the following two properties (relative to a reference implementation):

1. **Replaceability.** How easily can a third-party observer tell if we replaced the reference implementation with the new implementation?

2. **Inherent performance trade-offs.** Different congestion control algorithms represent different trade-offs in a network. Does the new implementation operate in the same trade-off space as the standard reference implementation?

To this end, we propose a metric called the *Performance Envelope* (PE) that was built on these two ideas. To investigate a new QUIC CCA implementation, we sample the delay ($d$) and throughput ($T$) of the new implementation while it competed with the reference (kernel) implementation in a controlled network environment. These $(d, T)$ pairs were then plotted on a delay-throughput plane to visualize the delay-throughput trade-off space for the new QUIC CCA implementation. The region defined by the convex hull of this point cloud was referred to as the PE of the new QUIC CCA implementation.

To measure the replaceability of a QUIC implementation, the PE of the QUIC im-




**(a)** A single-cluster PE **(b)** A multi-cluster PE

**Figure 5.1:** A single convex hull for the PE does not fully capture low conformance in `quiche` CUBIC.

plementation competing with a reference implementation was compared with the PE of a reference implementation competing with itself. The area of the overlap between these two PEs was then used as a measure of similarity (called *conformance*) between the two implementations (see Figure 5.1a).

However, a naïve definition of the PE (as described in our early work [4]) has its limitations. For example, we found that plotting the performance envelope with a single convex hull does not provide us with sufficient granularity to identify low conformance in some cases. One such example is illustrated in Figure 5.1.

The maximum possible value for *conformance*, the overlap between the PEs of the reference and QUIC implementations, is 1. We can clearly see that using a single convex hull for the PE does not fully capture low conformance in the `quiche` implementation of CUBIC in Figure 5.1a. This is because while the overlap between the PEs in Figure 5.1 is relatively large, most of it comprises empty space without any data points. This will inadvertently cause us to overestimate the similarity between these implementations, even though the two implementations may exhibit very different behaviors. We address this issue by clustering the delay-throughput pairs before constructing the final PEs (see Figure 5.1b).

Furthermore, we need a way to provide hints on how the conformance of a QUIC implementation could be improved. We found several QUIC implementations where the conformance can be improved by merely tuning their intentionally misconfigured `cwnd` and sending rate parameters (§5.3). In such cases, the PE of the reference and tested QUIC implementations generally have the same shape and the same number of clusters but are translated to a different region in the delay-throughput plane.

To identify such implementations, we propose an additional metric *Conformance post-Translation* (or *Conformance-T*), that is the maximum conformance that can be achieved by translating the PE of a QUIC CCA implementation. Generally, a high Conformance-T would indicate that an implementation's conformance can be improved significantly with simple parameter tuning. We show that this parameter tuning can be guided by the translation vector ($\Delta$-throughput, $\Delta$-delay).

In summary, our work advances the state of the art in our understanding of speciation [4, 69] in QUIC CCAs with the following contributions:

1. We present a comprehensive study of 11 QUIC stacks using this enhanced definition of the PE in both controlled testbeds and on the Internet. Our results show that while most QUIC CCA implementations are conformant in shallow buffers, they become less conformant in deep buffers (§5.2.1). In the process, we identified *seven* QUIC implementations that had low conformance;

2. We propose new metrics *Conformance*, *Conformance-T*, $\Delta$-throughput, and $\Delta$-delay, that can provide hints on the root cause of low conformance for a QUIC implementation (§5.1.2);

3. We demonstrate how low-conformance implementations can cause unfairness (§5.2.3) and subvert our expectations of how we expect different CCAs to interact (§5.2.4); and

4. We identify implementation-level differences that led to the low conformance and

propose modifications to improve conformance for three QUIC CCA implementations (§5.3). We were also able to identify instances where low conformance arises when features that are a part of the TCP stack and not the CCA itself (like Hystart (RFC 9406) [112]) are not implemented in QUIC stacks.

The traces and source code for all our experiments is available at [33].

Given the large number of QUIC stacks, it seems inevitable that there will be an increasing number of non-conformant QUIC CCA implementations. Our proposed metrics will provide developers with a means to understand how their CCA implementations deviate from standard kernel implementations and with hints to make the required modifications to ensure that their CCA implementations are conformant, thus reducing the likelihood of mistakes that might cause instability and performance degradation to the Internet.

While we had initially set out to study how we can ensure that future QUIC CCA implementations are conformant to the standard kernel implementations, we have come to realize that the standard kernel implementations are also moving targets that will evolve with time. While not covered in this thesis, the question of how QUIC CCA implementations can keep up with new RFCs and evolving kernel implementations is also an important concern.

## 5.1 Methodology

Our goal is to quantify the conformance of QUIC implementations of CUBIC, BBR, and Reno for the QUIC stacks in Table 5.1. In this section, we describe the PE as well as the Conformance, Conformance-T, $\Delta$-throughput, and $\Delta$-delay metrics.

### 5.1.1 Measuring similarity between implementations

To understand how well the CCAs implemented in a QUIC stack compare to the standard reference implementations, we needed a way to quantify and visualize the difference. We also want to be able to do so in a code-agnostic way, i.e. we should not have to read the code to pick out any deviations.

One straightforward way to identify deviations would be to compare the `cwnd` evolution to a reference implementation running in identical network conditions [**?** ]. However, we argue that this is impractical in the context of QUIC. Given that QUIC is implemented in the user space, it is unreasonable to expect these implementations to accurately replicate the complex waveforms in algorithms like CUBIC exactly. User space implementations of these algorithms generally have a fast path and a slow path that serves to approximate the behaviour of these algorithms, but they do not exactly match them. If the developers set out to exactly match the `cwnd` evolution of these algorithms, they would not be able to realistically do so in the user space without significant impact on performance.

An alternative is to define a more *coarse-grained* definition for conformance in the context of fairness. If a new implementation of a congestion control algorithm is as fair/unfair to a reference flow as the standard implementation, we can say that it is behaving in a manner *conformant* to the reference implementation. However, we are of the view that such an approach will not adequately capture the finer differences between different congestion control algorithms.

Our key insight is that a good measure of conformance should be based on a metric that captures the different trade-offs of different congestion control algorithms. To this end, we propose the *Performance Envelope* (PE), a multi-dimensional metric for comparing the relative behaviour of different implementations of different congestion control algorithms. The Performance Envelope is a visual representation of the different

trade-offs made by different congestion control algorithms.

In this chapter, we will evaluate the various implementations of standard congestion control algorithms in QUIC by looking at their throughput and delay trade-offs. We decided to choose these two metrics, because the trade-off between throughput and delay is the key consideration in the design of most modern congestion control algorithms [14, 16, 22, 55, 93]. Therefore, even if an implementation is not completely accurate, it should at least offer the same throughput-delay tradeoffs as the reference (kernel) implementation. However, depending on the application, the performance envelope can be adapted to capture the trade-offs between other network metrics as well.

### 5.1.2 Defining the Performance Envelope

To determine the PE for a test implementation, a flow running the test implementation is launched alongside a competing flow that runs the corresponding reference (Linux kernel TCP) implementation. Both flows are set to the same RTT and run through a bottleneck with a constant link capacity and a fixed-size droptail buffer. The flows run for 120 seconds to ensure that they have sufficient time to converge to steady state. The start and end of the flow traces are also truncated by 10% to remove the transient behaviors. The throughput and delay time series data (computed offline via packet trace) of the test implementation is then sampled every 10 RTTs and plotted pair-wise $(d, T)$ on a delay-throughput plane as a point cloud. The region defined by the convex hull of this point cloud is the PE for the implementation. Empirically, we have found that sampling the time-series throughput and delay data at this rate is sufficient to capture an implementation's PE. In other words, sampling more frequently does not substantially affect the shape of the PE for a CCA implementation.

**Handling outliers.** Since the data points that are used to determine the PE are instantaneous values, there will be outliers. Earlier, we removed these outliers by eliminating 5% of the points with the largest Euclidean distance from the centroid of the PE.

However, we found that there is no guarantee that the points furthest from the centroid are necessarily outliers and by doing so, we risk artificially reducing the variance in the PE. Ideally, we want to remove the outliers that are points that arise from natural network variation across trials and not artifacts of the implementation itself. Therefore, we decided that a more principled way to remove outliers was to capture $(d, T)$ pairs over multiple trials and use the intersection of the convex hulls produced by all these trials to be the final PE. In practice, it turns out that our approach also removes roughly 5% of the points on average for our experiments.

**One convex hull is not enough**. From Figure 5.1, we can see that the distribution of the points in the point cloud is often not uniform. Hence, if we use only a single convex hull, it is plausible that we may include large regions of empty space that do not contain any points. In other words, using only a single convex hull will often result in the overestimation of an implementation's conformance. To address this issue, instead of a single convex hull, we use a clustering algorithm to group data points into clusters and then calculate convex hulls for each individual cluster. The final PE is then the set of all convex hulls.

**How many clusters is enough?** We use the standard $k$-means clustering algorithm [113] to compute clusters from our set of data points in the throughput-delay plot. Usually, the number of clusters for the $k$-means algorithm is determined using the elbow method, which selects the inflection point for the mean squared error of the resulting clusters. However, in our case, we found that the regular elbow method was not satisfactory because there was no obvious inflection point if we considered the mean squared error.

On the other hand, we can see in Figures 5.2 and 5.3 that a PE often has a "natural" number of clusters arising from the characteristics of the CCA. For BBR, this natural number is generally 2 (because of its distinct `ProbeBW` and `ProbeRTT` phases, see Figure 5.2). For CUBIC and Reno, natural clusters still exist, but there does not seem to

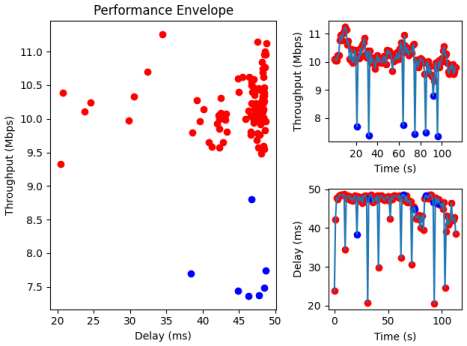
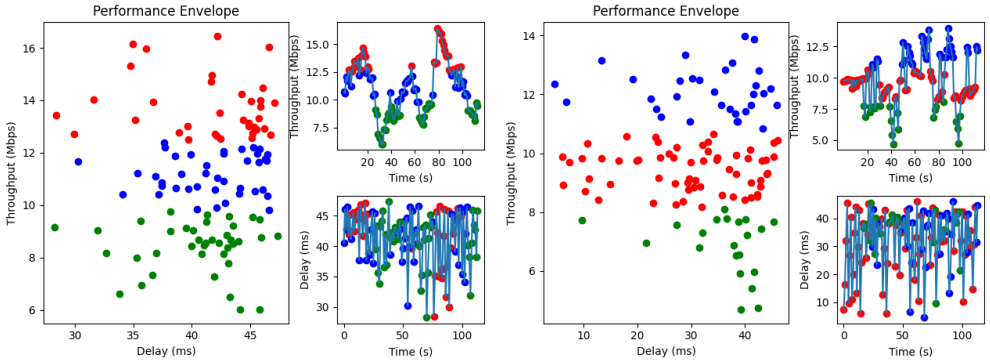



**Figure 5.2:** Two distinct clusters corresponding to TCP BBR's `ProbeBW` (red) and `ProbeRTT` (blue) phases.




**(a)** TCP CUBIC **(b)** TCP Reno

**Figure 5.3:** Clusters for CUBIC and Reno are less distinct and tend to form around different throughput levels.

be a fixed number (see Figure 5.3).

In order to determine this "natural" number of clusters algorithmically, we ran the $k$-means algorithm for all $k \geq 1$ for each trial. For each $k$, each trial will produce a PE with $k$ convex hulls. The final PE for each $k$ is then computed as the intersection for all the convex hulls over all the trials. For each of these PEs, we compute and plot the intersection over union (IOU) $R$, which we define as the proportion of the total data points for all the trials contained in the PE. This is effectively the amount of information retained in the PE for each value $k$.

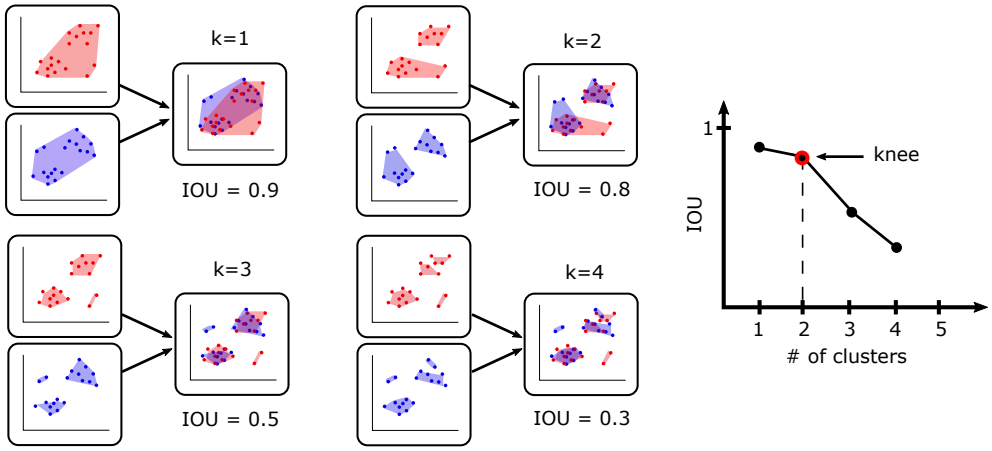



**Figure 5.4:** Determining $k$, the number of clusters for a Performance Envelope. IOU = Intersection over Union.

What we found was that $R$ is a strictly decreasing function of the number of clusters $k$, since as the number of clusters increases, the size of each cluster becomes smaller and the final intersection will contain fewer data points. Because it was indeed true that there was a "natural" number of clusters for each implementation, we found that $R$ drops most steeply at some $k$ for all the instances that we studied. We use the value of $k$ before the drop as the number of clusters for the final PE. We illustrate this iterative process in Figure 5.4, with the blue and red point clouds representing data points from two trials of the same measurement. We removed the outliers from each trial *before* we computed the number of clusters for a PE so that the outliers do not impact the number of clusters in the PE for a CCA implementation.

### 5.1.3 Quantifying similarity with *Conformance* and *Conformance-T*

The Performance Envelope, as described in §5.1.2, is not only a convenient way to visualize the throughput-delay trade-off a CCA implementation makes, but also allows us to compare two different CCA implementations. Given that the PE is a 2D region in the throughput-delay plane, we can use the overlap between the PEs of different implementations to estimate how *similar* they are. Since we are interested in understanding how

well a QUIC implementation conforms to its kernel counterparts, we formally define this overlap between two PE as *Conformance*.

**Conformance.** The *conformance* of an implementation is determined by calculating the overlap its PE has with the PE of the reference implementation. The overlap is weighted by the number of points present in the overlapping region. In particular,

$$\text{Conformance} = \frac{\text{\# of points in the overlapping region}}{\text{total \# of points in both PEs}}$$

Clearly, the maximum possible value of conformance is 1 (complete overlap) and the minimum value is 0 (no overlap).

Other than measuring the conformance of different CCA implementations more accurately, we also want a way to deduce how it might be possible to improve the conformance of an implementation via simple parameter tuning. Modern congestion control algorithms and QUIC stacks can be quite complex, and therefore manually checking and tuning all their parameters is not tractable. What we need are hints for the implementor on what tuning might be needed to improve a CCA implementation's conformance. We found many instances where this is possible. In such cases, the PE of an implementation generally has the same shape and number of clusters as the reference implementation but is merely translated to another region in the delay-throughput plane.

**Conformance-T.** To identify implementations that can likely be fixed with parameter tuning, we compute the translation that will maximize the intersection between the respective clusters of data points. We refer to this overlap post-translation as the *Conformance post-Translation* (or *Conformance-T*). To compute Conformance-T, we first derive a rough translation vector as the vector difference between the centroids of the data points for the implementation PE and for the reference PE, respectively. We then obtain the translation vector ($\Delta$-throughput, $\Delta$-delay) by searching the vicinity for a translation that achieves the maximal overlap between the two PEs. The local search

is performed within a bounding box that allows the PE to vary by up to 20% of the maximum throughput and delay. In most cases where the Conformance-T is high, the initial centroid-based translation is sufficient to achieve the maximum overlap. In general, a high Conformance-T value for a low-conformant implementation suggests that the implementation's conformance can be significantly improved with just simple parameter tuning.

To understand how Conformance-T works, consider the following experiment: we know that BBR multiplies its BDP estimate with a constant called `cwndgain` to determine its `cwnd`. By default, `cwndgain` is set to 2 in the Linux kernel. We modified the kernel version of BBR by changing its `cwndgain` and measured the Conformance and Conformance-T values for modified implementations with a range of `cwndgain` values from 1.0 to 4.0. We plot the resulting values for Conformance-T in Figure 5.5. Unsurprisingly, Conformance and Conformance-T are highest when `cwndgain` is 2.0. As the gap in `cwndgain` in the modified implementation increases, the Conformance drops as expected, even though the algorithmic behavior of the modified version is almost identical to that of the vanilla kernel BBR implementation. On the other hand, the Conformance-T remains relatively high. This suggests that Conformance-T is a relatively robust way to capture shifts in observed behavior arising from parameter tuning.

**Parameter tuning.** It turns out that the translation required to compute Conformance-T also provides us with hints on the systematic difference between a QUIC implementation and its corresponding reference implementation. We capture the 2 components in the required translation as the translation vector ($\Delta$-throughput, $\Delta$-delay).

Consider the two *knobs* a congestion control algorithm usually uses to regulate how aggressive it is: (i) its sending rate and (ii) its `cwnd`. For QUIC implementations showing low conformance but a high value of Conformance-T, we can deduce which of these knobs have been improperly tuned and correct for them. For example, if the `cwnd` of an implementation is more than it should be, it would typically have higher throughput and

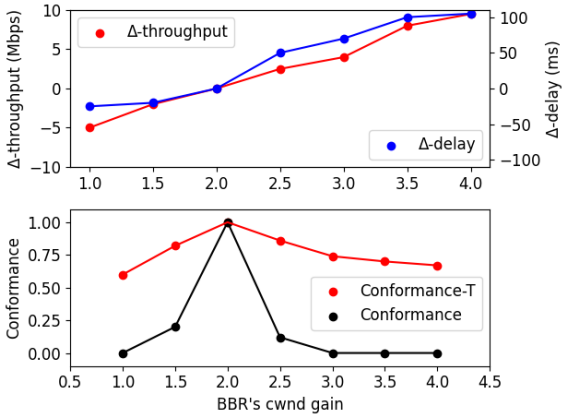



**Figure 5.5:** Conformance and Conformance-T values for modified versions of TCP BBR.

higher delay since it puts more packets in flight. This would show up as a large positive Δ-throughput and a large positive Δ-delay. We see this trend in Figure 5.5 where we see both Δ-throughput and Δ-delay increase as the `cwndgain` is increased.

On the other hand, if the implementation sets a correct `cwnd` but sends its packets at a larger sending rate than it should, we would see a large positive Δ-throughput but a negligible increase in Δ-delay. We see this behavior in `mvfst` BBR (See Figure 5.10) which we had earlier identified to have set its pacing gain to a value higher than the default [4].

### 5.1.4 Experiment Setup

Our experiments were conducted on a testbed with two Linux machines (Ubuntu 20.04, kernel version 5.13.0-44-generic) connected via a 1 Gbps Ethernet cable. To generate the QUIC flows, we installed the open-source QUIC stacks on both machines and used the test clients/servers provided. To generate the TCP flows, we used the iperf3 [114] tool.

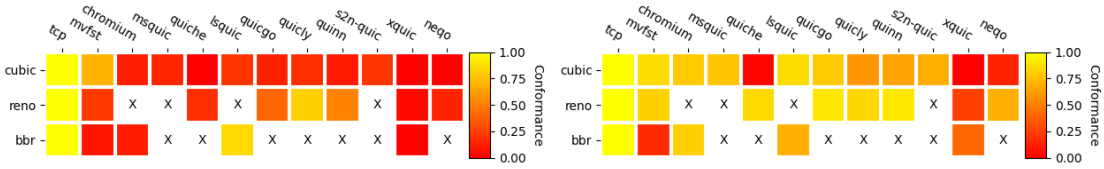
We used a modified version of our open-source tool QUIC Bench [33] to run our experiments and compute our new metrics Conformance-T, Δ-throughput, and Δ-delay.

**Table 5.2:** List of Known IETF QUIC/TCP stacks.

| Organization | Stack | Open Source? | Implements CCA? | Stable? | Deployed? | Evaluated? |
|---|---|---|---|---|---|---|
| Facebook | `mvfst` [101] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Google | `chromium` [102] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Microsoft | `msquic` [103] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Cloudflare | `quiche` [104] | ✓ | ✓ | ✓ | ✓ | ✓ |
| LiteSpeed | `lsquic` [105] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Go | `quicgo` [106] | ✓ | ✓ | ✓ | ✓ | ✓ |
| H2O | `quicly` [107] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Rust | `quinn` [108] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Amazon Web Services | `s2n-quic` [109] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Alibaba | `xquic` [110] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Mozilla | `neqo` [111] | ✓ | ✓ | ✓ | ✓ | ✓ |
| Akamai | `akamaiquic` [115] | ✗ | - | - | - | ✗ |
| Apple | `applequic` [116] | ✗ | - | - | - | ✗ |
| Apache | `ats` [117] | ✓ | ✓ | ✓ | ✗ | ✗ |
| F5 | `f5` [118] | ✓ | ✗ | ✗ | ✗ | ✗ |
| Haskell | `haskellquic` [117] | ✓ | ✗ | ✗ | ✗ | ✗ |
| Java | `kwik` [119] | ✓ | ✗ | ✗ | ✗ | ✗ |
| nghttp | `ngtcp2` [120] | ✓ | ✗ | ✗ | ✗ | ✗ |
| nginx | `nginx` [121] | ✓ | ✗ | ✗ | ✗ | ✗ |
| Pico | `picoquic` [122] | ✓ | ✓ | ✗ | ✗ | ✗ |
| Python | `aioquic` [123] | ✓ | ✗ | ✓ | ✓ | ✗ |
| Quant | `quant` [124] | ✓ | ✓ | ✗ | ✗ | ✗ |

We configured the socket buffer sizes for both UDP and TCP to be 12,582,912 bytes in order to have a fair comparison between TCP and QUIC.

As shown in Table 5.2, there are currently at least 22 modern QUIC stacks [28] available. However, in this chapter, we only evaluated 11 of them. We picked these 11 stacks because they are either used by major public companies, such as Google [102], Facebook [101], and Microsoft [103] or are the de-facto standard QUIC libraries in programming languages such as Go [106] or Rust [108]. The remaining stacks that we did not evaluate are either closed-sourced, had no stable version available, or did not implement congestion control.




**(a)** 5 BDP (deep) buffer

**(b)** 1 BDP (shallow) buffer

**Figure 5.6:** Conformance becomes significantly worse in 5 BDP (deep) buffers. (10 ms RTT, 20 Mbps)

## 5.2 Measurement Results

In this section, we present the results of our evaluation of the 11 QUIC stacks listed in Table 5.1. In §5.2.1, we compare the PEs of the QUIC implementations of CUBIC, Reno, and BBR to their corresponding Linux TCP implementations (hereafter referred to as the reference implementations) and calculate their conformance with respect to these reference implementations.

To investigate the general fairness between different implementations, we also performed a bandwidth-share-based analysis of all pairwise combinations of the 11 QUIC stacks. We present the results in §5.2.3. We also discuss how we can expect low-conformance QUIC implementations to subvert our expectations of how CUBIC and BBR interact in §5.2.4.

All evaluations were done under a variety of network conditions that were emulated by varying the network parameters as follows:

1. RTT (10 ms and 50 ms);
2. bottleneck bandwidth (20 Mbps and 100 Mbps); and
3. bottleneck buffer size (0.5, 1, 3, 5 times the BDP)

To ensure that buffer sizes are comparable across all the RTT and bottleneck bandwidth combinations, we normalize them as multiples of the Bandwidth-Delay Product (BDP).

**Table 5.3:** Summary of low-conformant implementations (1 BDP Buffer).

| Stack | Type | Conf | Conf-T | Δ-tput | Δ-delay |
|---|---|---|---|---|---|
| `chromium` | CUBIC | 0.6 | 0.74 | +3 Mbps | 0 ms |
| `neqo` | CUBIC | 0 | 0.62 | −6 Mbps | −5 ms |
| `quiche` | CUBIC | 0.08 | 0.55 | +5.5 Mbps | 0 ms |
| `xquic` | CUBIC | 0.55 | 0.64 | 0 Mbps | −5 ms |
| `mvfst` | BBR | 0 | 0.7 | +9 Mbps | 0 ms |
| `xquic` | BBR | 0.15 | 0.42 | +4 Mbps | 0 ms |
| `xquic` | Reno | 0.38 | 0.81 | −4 Mbps | −3 ms |

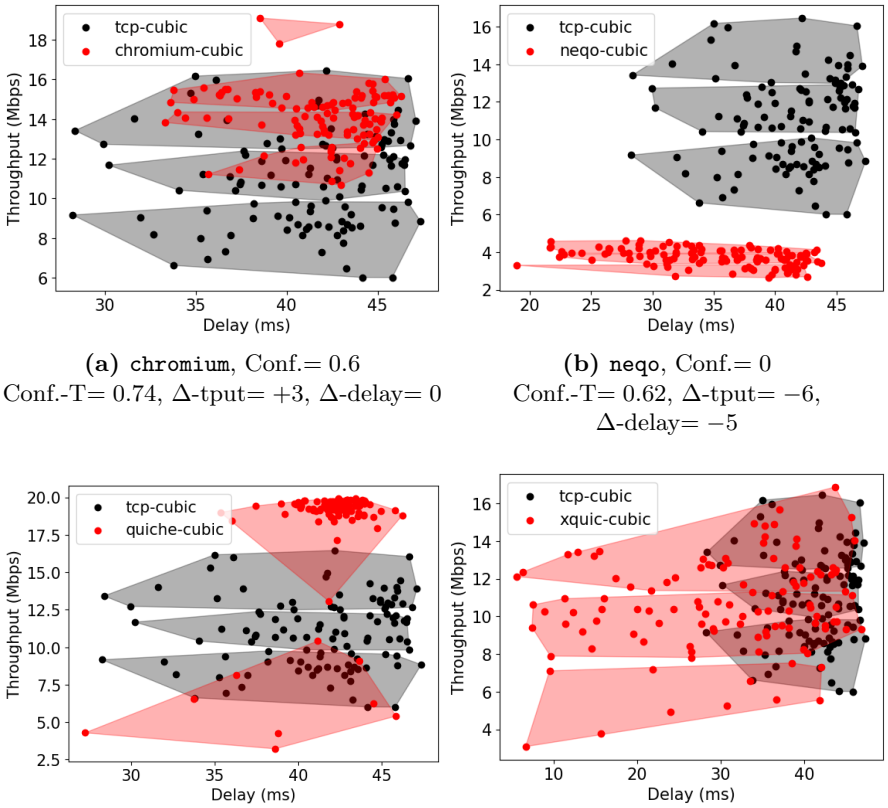
All network parameters were set using *tc* and Mahimahi [125] and each experiment was repeated 5 times.

We note here that all our Performance Envelope and conformance measurements are done over relatively stable network profiles with constant bottleneck bandwidths and simple droptail buffers. This is because while trying to understand how well QUIC implementations of standard CCAs resemble their kernel counterparts, we want any deviations to arise from the implementation and the QUIC stack itself, and not from the inherent variability of the network profile. It is for this reason that we evaluate the Performance Envelope with simple 2-flow experiments without any background traffic.

### 5.2.1 Conformance of CCA implementations of mainstream QUIC stacks

As we can see in Figure 5.6, the bottleneck buffer size has a significant impact on conformance. In particular, we can see from Figure 5.6a, that all implementations have poor conformance in 5 BDP (deep) buffers. Since this is a trend consistent across all stacks, it is plausible that the root cause of low conformance in deeper buffers is some artifact of the QUIC standard that becomes more pronounced when the buffers are larger. The investigation of this hypothesis remains as future work.

In general, the majority of the stacks are relatively conformant in shallow buffers as shown Figure 5.6b. There are several outliers to this trend, with some implementations




**(a)** `chromium`, Conf.= 0.6
Conf.-T= 0.74, Δ-tput= +3, Δ-delay= 0

**(b)** `neqo`, Conf.= 0
Conf.-T= 0.62, Δ-tput= −6,
Δ-delay= −5

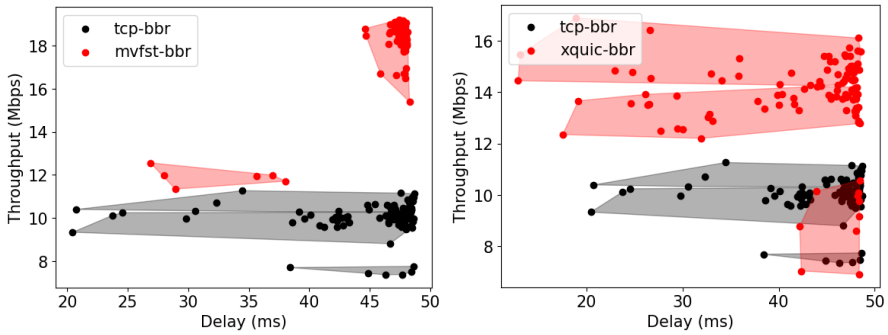
**(c)** `quiche`, Conf.= 0.08
Conf.-T= 0.55, Δ-tput= +5.5,
Δ-delay= 0

**(d)** `xquic`, Conf.= 0.55
Conf.-T= 0.64, Δ-tput= 0, Δ-delay= −5

**Figure 5.7:** QUIC CUBIC implementations with low conformance for 1 BDP buffers.

showing very low conformance (<0.5) even in shallow 1 BDP buffers. From Figure 5.6b, we can identify the 7 low-conformant stacks in red. The results for these stacks (in 1 BDP buffers) are summarized in Table 5.3. `xquic` CUBIC is included in the list, despite having conformance marginally greater than 0.5, because it was found to be extremely unfair to other implementations (§5.2.3).

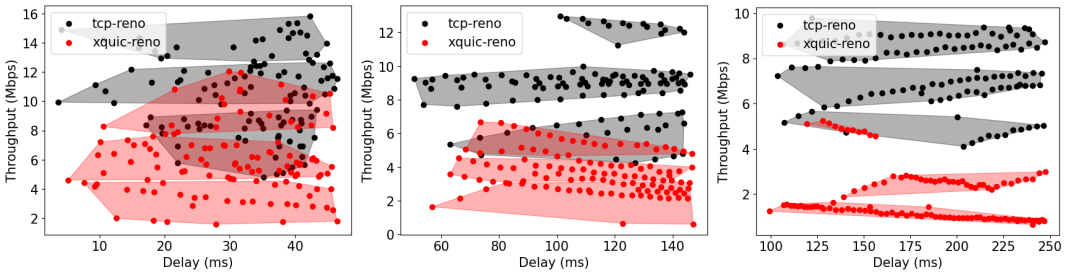
In §5.3, we describe modifications that can make `xquic` BBR and `quiche` CUBIC more conformant; for `xquic` CUBIC, we were able to identify the root cause for the low conformance. The PEs for non-compliant CUBIC and BBR QUIC implementations are shown in Figures 5.7 and 5.8 and the PEs for the sole non-compliant Reno implemen-




**(a)** `mvfst`, Conf.= 0
Conf.-T= 0.7, Δ-tput= +9, Δ-delay= 0

**(b)** `xquic`, Conf.= 0.15
Conf.-T= 0.42, Δ-tput= +4, Δ-delay= 0

**Figure 5.8:** QUIC BBR implementations with low conformance for 1 BDP buffers.



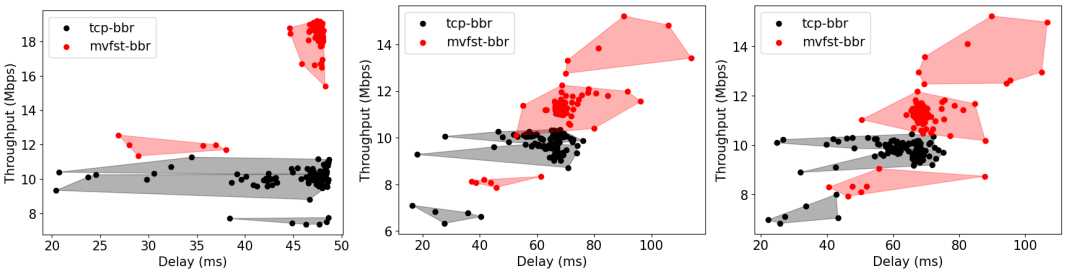

**(a)** 1 BDP buffer, Conf.= 0.38
Conf.-T= 0.81, Δ-tput= −4,
Δ-delay= −3

**(b)** 3 BDP buffer, Conf.= 0.14
Conf.-T= 0.67, Δ-tput= −5,
Δ-delay= 0

**(c)** 5 BDP buffer, Conf.= 0.08
Conf.-T= 0.6, Δ-tput= −6,
Δ-delay= 0

**Figure 5.9:** Performance envelopes for `xquic` Reno for different bottleneck buffer sizes.

tation are plotted in Figure 5.9. While we performed measurements over a large variety of network configurations as described in §5.2, link speed and RTT had a marginal impact on the results and so we only produce representative plots varying the buffer sizes unless stated otherwise. It is likely that link speed and RTT do not have a pronounced effect because we normalize our buffer sizes by the BDP in our relatively stable network profiles. This trend may not hold in networks with highly volatile bandwidth variations, like 5G networks.

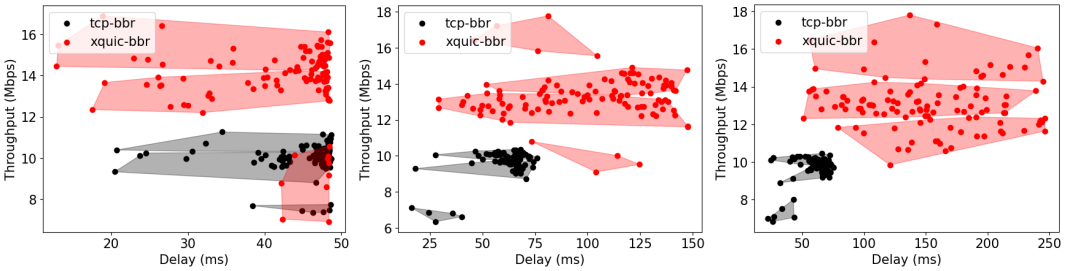



**(a)** 1 BDP buffer, Conf.= 0 Conf.-T= 0.7, Δ-tput= +9, Δ-delay= 0

**(b)** 3 BDP buffer, Conf.= 0.01 Conf.-T= 0.71, Δ-tput= +2, Δ-delay= −5

**(c)** 5 BDP buffer, Conf.= 0 Conf.-T= 0.68, Δ-tput= +2, Δ-delay= 0

**Figure 5.10:** Performance envelopes for `mvfst` BBR. (Conf.=Conformance)




**(a)** 1 BDP buffer, Conf.= 0.12 Conf.-T= 0.42, Δ-tput= +4, Δ-delay= 0

**(b)** 3 BDP buffer, Conf.= 0 Conf.-T= 0.53, Δ-tput= +3, Δ-delay= +25

**(c)** 5 BDP buffer, Conf.= 0 Conf.-T= 0.56, Δ-tput= +4, Δ-delay= +60

**Figure 5.11:** Performance envelopes for `xquic` BBR. (Conf.=Conformance)

#### 5.2.1.1 CUBIC

The implementations of CUBIC in `chromium`, `neqo`, `quiche`, and `xquic` had low conformance. We had earlier found the modifications needed to make `chromium` CUBIC more conformant [4]. To build on our earlier work, we describe how we can mitigate the low conformance for `quiche` CUBIC and `xquic` CUBIC in §5.3.

#### 5.2.1.2 BBR

We found BBR implementations in `mvfst` and `xquic` to have low conformance. The conformance for `mvfst` was better for deep buffers (see Figure 5.10), while for `xquic`,

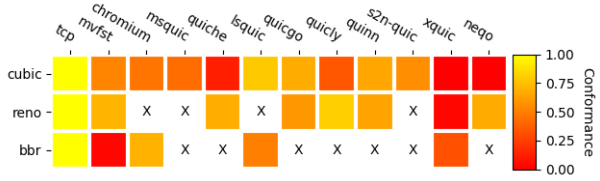
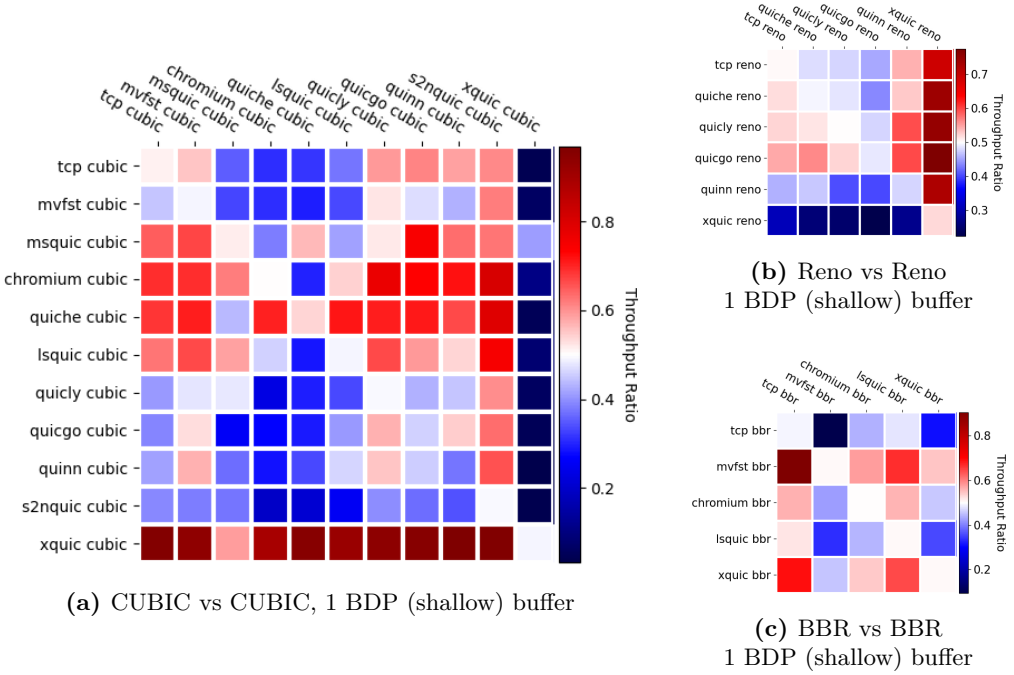



**Figure 5.12:** Conformance of various QUIC stacks when tested on AWS. Link speed was locally limited to 100 Mbps.

the lack of conformance became worse in deep buffers (see Figure 5.11). However, both these implementations show significantly high Conformance-T values. The positive Δ-throughput for both these implementations suggests that they might be reasonably conformant implementations of BBR with some parameter tuning. We had earlier highlighted that `mvfst` BBR multiplies its final sending rate by 120% in order to improve throughput [4]. We verified that reducing the send rate to 100%, `mvfst` BBR will become more conformant. We show in §5.3 that a similar modification can also improve the conformance of `xquic` BBR.

#### 5.2.1.3 Reno

QUIC Reno implementations are generally conformant for most QUIC stacks. The conformance in deeper buffers is also relatively better than CUBIC and BBR. This is likely because Reno is the simplest algorithm among the three CCAs investigated, and is thus easier to implement correctly. The only exception among them is `xquic`, as shown in Figure 5.9. The fact that even a simple CCA like Reno is non-conformant for `xquic` suggests that there might be a larger issue with the `xquic` stack itself, given that all its CCA implementations show poor conformance. Upon investigation, we could not find anything that was clearly wrong with the `xquic` CCA implementations, suggesting that the root cause was beyond algorithmic parameters and correctness of the CCA implementation. Determining the exact cause of the observed differences remains future work.




**(a)** CUBIC vs CUBIC, 1 BDP (shallow) buffer

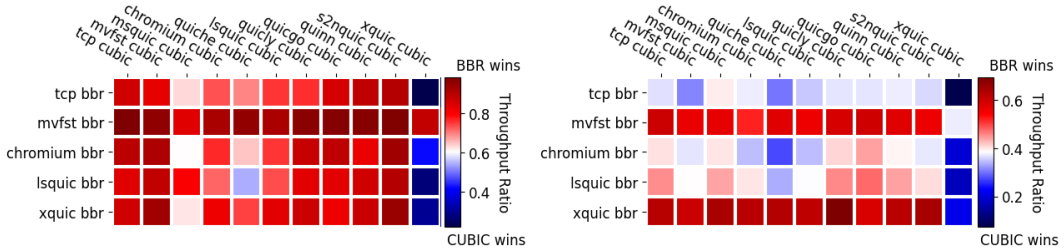
**(b)** Reno vs Reno
1 BDP (shallow) buffer

**(c)** BBR vs BBR
1 BDP (shallow) buffer

**Figure 5.13:** Throughput ratios for competing implementations on CUBIC, Reno, and BBR (20 Mbps, 50ms RTT).

### 5.2.2 Investigating Conformance "in the Wild"

We repeated our experiments in §5.2.1 on the Internet. In these measurements, the senders were run on `aws` instances and connected to receivers on physical servers in our lab. We limited the link speed to 100 Mbps at the server. We measured the ping latency before every experiment and added additional delay using Mahimahi [125] to keep the RTT constant at 50 ms across all trials. Like before, the results of these experiments are plotted as a heatmap in Figure 5.12. We found the conformance numbers to be similar to our results for 1 BDP buffer in our testbed (see Figure 5.6b). While we were tempted to conclude that this hints that the buffers on the Internet are shallow, we refrain from making any such claim since a CCA's performance on the Internet can be impacted by other network artifacts as well.




**(a)** 1 BDP (shallow) buffer. Expected to be red. **(b)** 5 BDP (deep) buffer. Expected to be blue.

**Figure 5.14:** Different implementations of CUBIC and BBR competing with each other (a throughput ratio of 1 means the BBR flow starves the CUBIC flow.)

### 5.2.3 Fairness between Implementations

The PE only compares the time series behavior of QUIC implementations to their kernel counterparts, which can provide us an estimate of how faithfully QUIC CCA implementations conform to the reference implementations. However, no matter how hard developers might try to reproduce the behaviors of the kernel implementations, there will be overheads, since the QUIC stack CCA implementations run in the user space. There will also likely be some inherent unfairness between algorithms. It is also inevitable that some organizations will want to modify and tune their congestion control algorithms for their own applications and therefore lead to low conformance by design.

For such cases, we would still be keen to ensure that new implementations can coexist with other kernel and QUIC CCA implementations without causing instability or significant degradation. To determine if new implementations are friendly to other implementations, we do a simple bandwidth-share-based analysis of the pairwise combinations of all 23 QUIC CCA implementations investigated in this chapter. Moreover, this serves as an extension to the conformance analysis presented in §5.2.1 and a sanity check that high conformance is highly correlated to fairness to other implementations.

In these experiments, the two competing flows share a 20 Mbps bottleneck link, 50 ms RTT, and a 1 BDP buffer. The bandwidth share is computed as $\frac{T_x}{T_x+T_y}$ and $\frac{T_y}{T_x+T_y}$ where

$T_x$ and $T_y$ are the throughputs of the two competing flows averaged across 5 trials. If the bandwidth share is greater than 0.5 for any flow, it implies that the flow has more than a fair share of the bandwidth.

Since it is well known that different congestion control algorithms can be unfair to each other [3, 25], we looked into fairness between implementations of the same CCAs. We plot the throughput ratios for competing implementations of CUBIC, BBR, and Reno in Figure 5.13. If we compare these results to the implementations earlier identified as being low-conformant in Table 5.3, it is not surprising to see which QUIC implementations of CUBIC, Reno, and BBR are overly aggressive. `chromium` CUBIC, `quiche` CUBIC, and `xquic` CUBIC (all previously identified low-conformant implementations of CUBIC) were unfair to all other implementations of CUBIC.

Similar trends exist for `xquic` BBR, `mvfst` BBR, and `xquic` Reno, which also show low conformance. We also note that `lsquic` CUBIC also shows some degree of unfairness despite having high conformance (0.76). This suggests that while low conformance is likely to lead to unfairness, high conformance does not necessarily result in fairness, so it is still important to do a bandwidth-share-based analysis for new QUIC CCA implementations.

### 5.2.4 Contradicting known trends in inter-CCA fairness

Given the current heterogeneous congestion control landscape on the Internet, it is important for interactions between different congestion control algorithms to be consistent and predictable. In particular, the interactions between CUBIC and BBR, the two most dominant congestion control algorithms on the Internet, has been a hot topic in congestion control research over the past few years [3, 25, 126].

In particular, it is well known that BBR will achieve higher bandwidth than CUBIC when they compete in shallow buffers due to CUBIC backing off frequently and BBR being largely loss-agnostic. Also, CUBIC is expected to achieve higher throughput than

BBR in deep buffers since CUBIC is a buffer-filler [3, 25]. In other words, Figure 5.14a is expected to be all red, and Figure 5.14b is expected to be all blue.

However, we see in Figure 5.14 that some QUIC implementations of CUBIC and BBR do not conform to these expectations. In particular, we see that in shallow buffers `xquic` CUBIC outperforms most BBR implementations (Figure 5.14a); in deep buffers, `xquic` BBR and `mvfst` BBR outperforms other CUBIC implementations. All three of these implementations were earlier identified as showing very low conformance (Table 5.3). This shows that in addition to introducing unfairness, low-conformant implementations can potentially subvert our expectations of how we expect standard congestion control algorithms to interact.

## 5.3 Fixing low-conformance implementations

QUIC CCA implementations might not behave like their Linux TCP counterparts for a number of reasons: (i) the implementation might not conform to existing standards; (ii) the algorithm parameters might be set differently; or (iii) because of implementation artifacts within the QUIC stack. As we discussed in §5.1.3, our new proposed metrics Δ-throughput (abbreviated as Δ-tput), and Δ-delay are often helpful in providing us with hints on the root cause of low conformance. In this section, we describe how we managed to improve the conformance of some of the low-conformance implementations identified earlier in §5.2. Most of our modifications required only a small number of lines of code (LoC). We summarize our findings in Table 5.4.

**Differences in implementation**. We had earlier proposed modifications to make `chromium` CUBIC and `mvfst` BBR more conformant [4]. We verified that these modifications are valid using our enhanced definition of the PE and Conformance. In addition, we also found modifications that could make `xquic` BBR and `quiche` CUBIC more conformant.

**Table 5.4:** Summary of successful modifications to low-conformant implementations (1 BDP buffer).

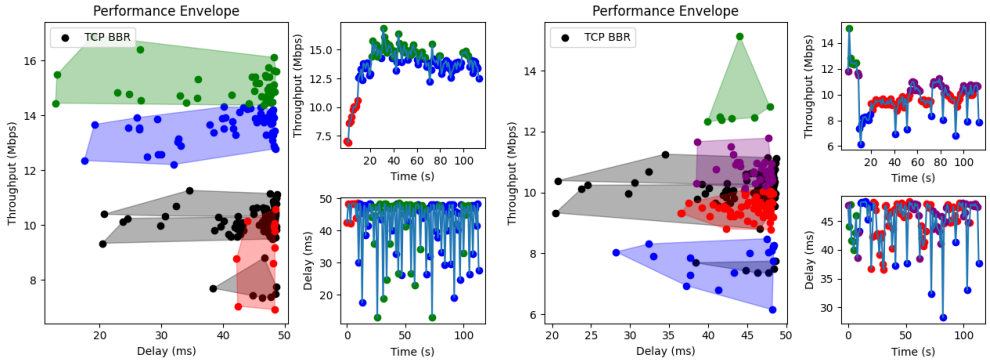
| | | | Original implementation | | | | Modified implementation | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Fixed? | Stack | Type | Conf | Conf-T | Δ-tput | Δ-d | Conf | Conf-T | Δ-tput | Δ-delay | LoC[#] | Remarks |
| ✓ | `chromium`[+] | CUBIC | 0.6 | 0.74 | +3 Mbps | 0 ms | 0.78 | 0.85 | 0 Mbps | 3 ms | 1 | Emulated flows reduced from 2 to 1 |
| ✓ | `mvfst`[+] | BBR | 0 | 0.7 | +9 Mbps | 0 ms | 0.8 | 0.8 | 0 Mbps | 0 ms | 2 | pacing gain reduced from 1.25 to 1 |
| ✓ | `xquic` | BBR | 0.15 | 0.42 | +4 Mbps | 0 ms | 0.38 | 0.47 | 0 Mbps | -2 ms | 2 | `cwnd` gain reduced from 2.5 to 2 |
| ✓ | `quiche` | CUBIC | 0.08 | 0.55 | +5.5 Mbps | 0 ms | 0.55 | 0.66 | +2 Mbps | 0 ms | 14 | Disabled RFC8312 [9] |
| ✗[*] | `xquic` | CUBIC | 0.55 | 0.64 | 0 Mbps | -5 ms | - | - | - | - | - | does not implement HyStart [112] |
| | | | 0.72 | 0.81 | -2 Mbps | 0 ms | - | - | - | - | - | Compared to CUBIC w/o HyStart |
| ✗ | `xquic` | Reno | 0.38 | 0.81 | -4 Mbps | -3 ms | - | - | - | - | - | Implementations verified to be |
| ✗ | `neqo` | CUBIC | 0 | 0.62 | -6 Mbps | -5 ms | - | - | - | - | - | compliant with existing standards. |

[#] Lines of code in required modification.
[+] Earlier identified and fixed [4].
[*] Implementation difference identified but not fixed. Implementation found to be conformant to TCP CUBIC with HyStart disabled.

The Conformance-T value for `xquic` BBR was almost triple its conformance, which suggested that `xquic` BBR could potentially be made significantly more conformant by parameter tuning. Upon investigation, we found its `cwndgain` was set to 2.5 instead of the RFC-recommended value of 2. By setting the `cwndgain` to 2, we were able to marginally improve conformance as shown in Figure 5.15.

After reviewing the implementation of `quiche` CUBIC, we discovered that RFC 8312 [127] was implemented. This draft proposes the rolling back of any back-off in the `cwnd` if a packet loss was deemed to be spurious. This mechanism has in fact not yet been implemented in the Linux kernel. When we disabled it in `quiche` CUBIC, we saw an immediate improvement in its conformance from 0.08 to 0.55 as shown in Figure 5.16. In general, we would expect QUIC CCA implementations to lag behind developments in the kernel. In this case, we have found a QUIC CCA implementation that leads kernel development.

**Missing Mechanism**. When we analyzed the implementation of `xquic` CUBIC, we found that TCP HyStart (RFC 9406) [112] was not implemented. TCP HyStart is a mechanism present in the Linux kernel that implements a modified slow start for CUBIC, where we will exit Slow Start when we see an increase in end-to-end delay. This makes the Hystart dramatically less aggressive than the traditional slow start. To verify



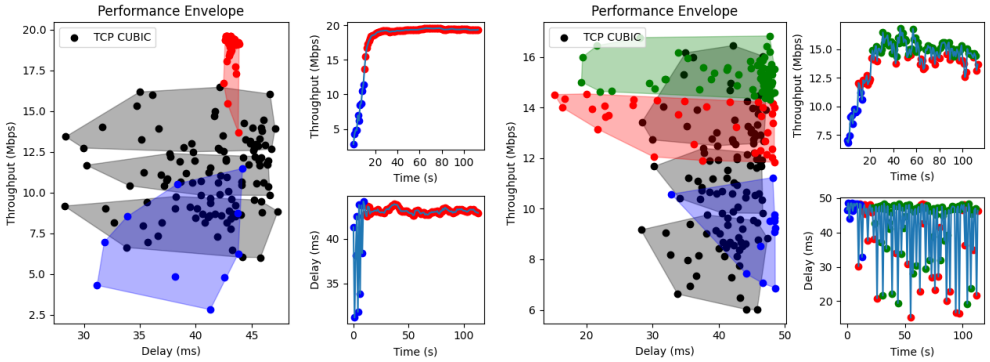

**(a)** `xquic` BBR original, Conf.= 0.15
Conf.-T= 0.42, Δ-tput= +4, Δ-delay= 0

**(b)** `xquic` BBR modified, Conf.= 0.38
Conf.-T= 0.47, Δ-tput= 0, Δ-delay= −2

**Figure 5.15:** `xquic` BBR's conformance before and after reducing `cwnd` gain from 2.5 to 2.

that this missing mechanism was the main cause of low conformance, we evaluated the conformance of `xquic` CUBIC with respect to TCP CUBIC with HyStart disabled. As shown in Table 5.4, the conformance was indeed much higher. We did not attempt to implement HyStart in `xquic` CUBIC to make it more conformant because HyStart is relatively complicated and we had already identified the root cause of low conformance.

**Indications of wider stack-level issues**. When we reviewed the implementations of `xquic` Reno and `neqo` CUBIC, we found them to be compliant with the standard algorithms. The parameter settings are also correct. This suggests that the low conformance is likely due to some artifact(s) in the QUIC stack rather than in the CCA implementation. This means that `xquic` developers need to pay attention not only to the implementations of the CCA but also to the implementation of the QUIC stack in order to achieve high conformance. The investigation into the low conformance of `xquic` Reno and `neqo` CUBIC is left as future work.




**(a)** `quiche` CUBIC original, Conf.= 0.08
Conf.-T= 0.55, Δ-tput= +5.5, Δ-delay= 0

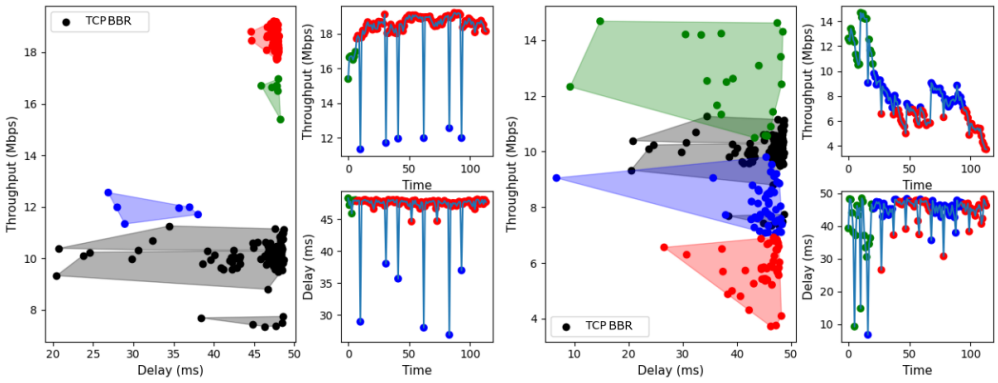
**(b)** `quiche` CUBIC modified, Conf.= 0.55
Conf.-T= 0.66, Δ-tput= +2, Δ-delay= 0

**Figure 5.16:** `quiche` CUBIC's conformance before and after disabling its detection of spurious packet losses (RFC8312 [9]).

## 5.4 Discussion

Given the low conformance that we have identified in modern QUIC stacks, we believe that QUIC congestion control research deserves more attention and further study as the results of classic CCA research may no longer apply. Fully understanding the interactions and impact of new QUIC stacks is likely to be a continuous process as these implementations morph with time. Also, we recognize that there are some limitations in our current measurement study.

**Refining bandwidth-share analysis.** The throughput ratios discussed in §5.2.3 provide an estimate of how well implementations can coexist with each other. However, in the future, we would like to refine our approach to measuring coexistence and general intra-CCA friendliness. In addition to running experiments over a larger range of network conditions, we would also like to experiment with different applications and measure their application-level metrics (such as QoE for video streaming). In our experiments, we launch both flows together. It is likely helpful to understand the impact of different start times and different flow durations on fairness.



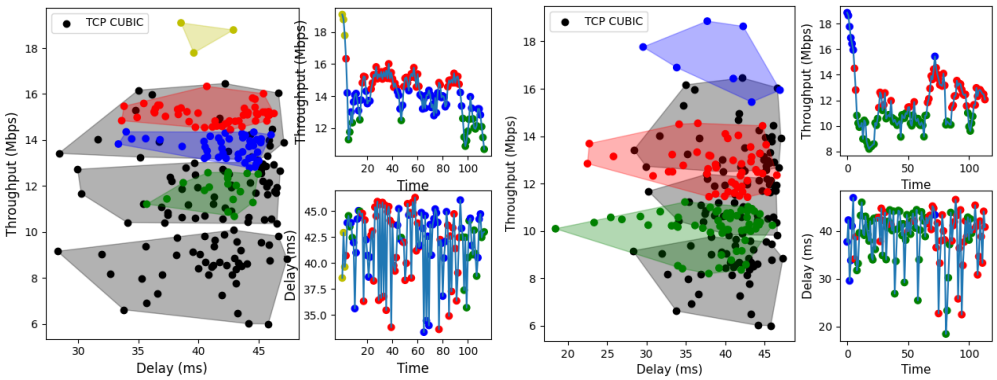

**(a)** `mvfst` BBR original, Conf.= 0.0
Conf.-T= 0.0, Δ-tput= +0.0, Δ-delay= 0

**(b)** `mvfst` BBR modified, Conf.= 0.0
Conf.-T= 0.0, Δ-tput= +0.0, Δ-delay= 0

**Figure 5.17:** `mvfst` BBR's conformance before and after reducing its pacing gain.

**Extending the Performance Envelope to other applications.** Besides benchmarking congestion control applications, the performance envelope also has the potential to serve as a tool for helping application choose their desired congestion control algorithms. Different applications usually value different network metrics. For example, live-streaming applications will generally value low latency, in contrast to applications that perform bulk downloads and value high throughput. Such applications can possibly leverage the performance envelope to identify the trade-off space they want to operate in and then select a congestion control algorithm whose performance envelope has the maximum overlap with their desired performance envelope.

**Systematic Root Cause Analysis.** While the methodology applied in this chapter has largely been successful in identifying low-conformance implementations of congestion control algorithms, we would like to do more to aid the debugging of these implementations. We feel that time series graphs (such as the ones in Figure 5.16) and Conformance-T are a good starting point in investigating which aspects an implementation may be differing in (such as `cwnd` or the sending rate). In the future, we would also like to automatically extract key parameters from implementations and try to correlate them with Δ-throughput and Δ-delay values of their Performance Envelopes. There is




**(a)** `chromium` CUBIC original, Conf.= 0.0 Conf.-T= 0.0, Δ-tput= +0.0, Δ-delay= 0

**(b)** `chromium` CUBIC modified, Conf.= 0.0 Conf.-T= 0.0, Δ-tput= +0.0, Δ-delay= 0

**Figure 5.18:** `chromium` CUBIC's conformance before and after changing the number of emulated flows from 2 to 1.

also scope for differentiating between implementation-level and stack-level differences for these QUIC implementations. For example, if we find that the same qualitative deviation in the PE across all the evaluated CCAs for a given QUIC stack (for example, say all the CCAs in a QUIC stack achieve lower throughput than their kernel counterparts), it may suggest that the root cause of this non-conformance lies in how the underlying QUIC stack is implemented, rather than in the implementation of the individual CCAs.

**Transitivity**. In our earlier study of 4 QUIC stacks [4], we found that the performance of the CCA implementations was transitive, i.e. if a CCA $X$ achieved higher throughput competing with CCA $Y$ and CCA $Y$ achieved higher throughput competing with CCA $Z$, then CCA $X$ would achieve higher throughput when competing with CCA $Z$. However, from our results of the 11 QUIC stacks evaluated in this chapter, we found that the relative performance of QUIC implementations is *not* transitive between different CCAs. For example, `lsquic` CUBIC beats `msquic` CUBIC and `msquic` CUBIC beats `chromium` BBR, but `lsquic` CUBIC does not beat `chromium` BBR when they compete in deep buffers. However, among the QUIC stacks that we evaluated, the intra-CCA performance seems to be transitive. That is, transitivity is likely to exist between QUIC

implementations for the same congestion control algorithm. A more detailed study of transitivity between CCAs remains as future work.

**Comparing Fairly Across Different CCAs.** Currently, while measuring the conformance of a QUIC implementation, we run it alongside its corresponding TCP implementation (or the reference flow). This is because our idea of conformance is built around *replaceability*–that is, we want to determine how easily a QUIC implementation can mimic a standard TCP implementation in terms of performance and behavior. However, this also means that our calculated PEs are only comparable to the PEs of other implementations implementing the same congestion control algorithm. In the future, we would like to define and experiment with running all QUIC implementations alongside the same standard background flow. This would allow us to have a fair basis to compare implementations of *different* congestion control algorithms.

**Keeping up with the kernel.** Even though the Linux kernel's TCP stack is a relatively stable reference for measuring the conformance of QUIC implementations, it is still a moving target. The kernel will also continue to evolve as new RFCs are proposed and implemented, as we have already seen with the implementation of Hystart in §5.3. In fact, RFC8312 [9], whose implementation in `quiche` reduced the conformance of its CUBIC implementation (Figure 5.16), is scheduled to be deployed only in the next stable version of the Linux kernel. These developments make a case for conducting regular conformance tests for QUIC implementations every time a new *milestone* kernel version with significant changes to the TCP stack is released.

## 5.5 Summary

In summary, we present the results of a measurement study of the congestion control algorithms in 11 popular open-source QUIC stacks. To the best of our knowledge, our measurement study is likely the most comprehensive evaluation of congestion control

algorithms in modern QUIC stacks to date. We address the limitations of previous approaches that evaluate QUIC congestion control and propose a new metric *Conformance-T* for identifying implementations where there is scope for improving conformance via parameter tuning. Our measurement study significantly advances the state of the art in our understanding of speciation [33, 69] and raises new research questions on the evolution of CCAs for QUIC.

## 5.6 Resources

Our measurement tool, along with the measurement traces for all the QUIC stacks discussed in this chapter are available on GitHub (`https://github.com/NUS-SNL/QUICbench`).

Chapter 6

# Keeping an Eye on Congestion Control in the Wild with *Nebby*

The composition of the Internet's congestion control landscape impacts how we size router buffers [30, 31], think about inter-flow fairness [3, 24, 25], and even decide on the deployability of new congestion control algorithms (CCAs) on the Internet [32]. In the past, relatively infrequent snapshots of the Internet's composition were sufficient for us to understand its congestion control landscape [19, 20]. However, recent developments (as we have discussed in §4 and §5) suggest that CCAs on the Internet are evolving faster than ever before.

The deployment of BBR and its variants is a perfect example of this rapid evolution. While BBR was first introduced back in 2016, the algorithm has continued to evolve over the years. At the time of writing, Google alone is known to have deployed three different versions of BBR [26, 27, 87]. Outside of Google, operators have also been found to deploy modified versions of BBR according to their own needs [4].

The adoption of QUIC [23] on the Internet is another catalyst that has influenced the evolution of the Internet's CCA landscape in recent years. While the QUIC standard itself does not introduce any new CCAs, QUIC congestion control is implemented in

the user space and thus makes it significantly easier to implement new CCAs and to deploy modified versions of existing CCAs. There is evidence that operators are already deploying their own variants of CCAs like CUBIC and BBR in their QUIC stacks [5]. These variants can behave very differently from their kernel counterparts.

Given that these developments have major consequences for the Internet's congestion control landscape, it is crucial to keep an eye on CCAs in the wild. Unfortunately, existing CCA identification tools [1, 19, 20, 21, 47, 48] do not work well with modern CCAs and encrypted protocols like QUIC.

In this chapter, we revisit the problem of CCA identification from first principles and articulate the key challenges in the context of today's rapidly evolving CCA landscape (§6.1.2). In particular, we argue that a CCA identification technique needs to be future-proof to handle new and yet unknown CCAs. We also need a new metric that can work well with modern rate-based CCAs. To address these challenges, we propose a principled approach to CCA identification that is extensible by design (§6.2).

Our approach is implemented as a tool called Gordon that uses bytes in flight (`BiF`) instead of the `cwnd` metric used by previous approaches [19, 20, 21, 47, 48**?** ]. Our key insight is that since rate-based CCAs use `cwnd` as a safeguard and not an operating point, measuring the `cwnd` is not sufficient for telling them apart. On the other hand, while `BiF` is equivalent to `cwnd` for loss-based CCAs, we show that it allows us to distinguish between different rate-based CCAs. To accurately estimate the `BiF` at the client, we introduce additional latency at a local bottleneck to gain visibility over a larger portion of the pipe between the target server and the client (§6.2.1). We also found a way to accurately estimate `BiF` for encrypted QUIC traffic (§6.2.2).

Our extensible classifier identifies CCAs by extracting segments based on the frequency and shape of their characteristic `BiF` periodic oscillations during steady state. Our approach works because existing CCAs converge to a *congestion avoidance* phase in the steady state. By identifying these characteristic segments for each CCA's `BiF`

trace, we show that a simple shape-based classifier is sufficient to identify all 12 CCAs available in the Linux kernel v5.18, and BBRv2 [26], with an average accuracy 96.7% (§6.3.1). Measurements are generally noisy. However, because there are many repeated segments in a trace, we have many opportunities to successfully detect the segments that correspond to different CCAs. To the best of our knowledge, Nebby is the first CCA identification tool that can identify CCAs for TCP and QUIC web servers and over a wide range of interactive applications.

We used Nebby to measure the Alexa Top 20,000 websites[1] during the period between Jun 2023 and Oct 2023 and made the following findings (§6.3):

1. While CUBIC remains the most popular CCA on the Internet, the deployment of CCAs can differ by region for TCP. We found that while TCP BBR's adoption is still substantial, its deployment in regions like Mumbai and Sao Paulo lags behind Ohio and Paris (§6.3.2).
2. Comparing our results with those of Mishra et al. [11], we found that BBR's share shrunk among the Alexa Top 20,000 websites since 2019. In fact, some websites have since switched from BBRv1 to CUBIC (§6.3.2).
3. BRRv2 was publicly released by Google in 2019. Most new adopters of BBR deployed BBRv2, instead of BBRv1. This is a positive sign, since BBRv2 is less aggressive towards loss-based flows. However, most of the early adopters of BBR (who are still running BBR) continue to run BBRv1, and have not yet migrated to BBRv2 (§6.3.2).
4. We detected the testing and deployment of BBRv3 in June 2023 before it was formally announced at the IETF in August 2023. This finding was confirmed by Google and demonstrates that Nebby is able to successfully detect the deployment of new and undocumented variants on the Internet by major players (§6.3.2).
5. We uncovered a group of websites deploying a class of unknown variants, which we call *AkamaiCC*. These CCAs are characterized by a blocky sending behavior and

[1]While Alexa has since been shut down, we used the last updated list [128] from February 2023 to do our measurements.

behave unlike any other known CCAs. Popular websites using AkamaiCC include `apple.com`, `hulu.com`, and `tiktok.com` (§6.3.3).

6. We show that QUIC adoption among the Alexa Top 20k websites is still relatively limited, with only about 8% of the websites measured responding to QUIC requests. CUBIC and BBR seem equally popular among the deployed QUIC services (§6.3.4).
7. We successfully identified the CCAs used by popular websites to serve video and audio traffic over interactive sessions over a Chrome browser. In addition, we found that it was common for different CCAs to be used for different content. BBR seems to be the CCA of choice for video streaming flows, while CUBIC is often used for static content (§6.3.5).

## 6.1 Background & Motivation

Since we essentially want to recapture a snapshot of the Internet's congestion control landscape, it would make sense to re-apply the methodologies discussed in §3 to at least measure the Alexa Top 20,000 websites over TCP. Therefore, when we decided to conduct this measurement study, our first instinct was to try to re-measure websites using Gordon.

### 6.1.1 Replicating Gordon

Gordon estimates a sender's congestion window by counting the number of unacknowledged packets in each RTT. To do so, Gordon makes a connection and drops packets till it sees a retransmission. Gordon repeats this process over hundreds of connections and then uses the resulting `cwnd` traces to identify the CCA. We expected the Internet to have evolved since Gordon's last measurement study in 2019, so we decided to use Gordon to classify the Alexa Top 10,000 websites. Unfortunately, we were sorely

**Table 6.1:** Distribution of TCP variants with *Gordon* [11].

| TCP variant | Websites | Proportion | Mishra et al. [1] (2019) |
|---|---|---|---|
| CUBIC [16] | 212 | 2.12% | 30.7% |
| BBRv1 [22] | 85 | 0.85% | 17.75% |
| CTCP [17]/Illinois[45] | 63 | 0.63% | 5.74% |
| Reno [14]/HSTCP [82] | 52 | 0.52% | 0.80% |
| Other CCAs | 0 | 0% | 18.87% |
| Unknown | 1,430 | 14.30% | 12.16% |
| Short flows | 6,282 | 62.82% | 7.47% |
| Unresponsive | 1,876 | 18.76% | 6.51% |
| Total | 10,000 | 100% | 100% |

disappointed to discover that Gordon was only able to successfully identify 4% of our measured websites and among these websites, only 4 categories of TCP variants were identified. We summarize the results in Table 6.1. We also reproduce the results from [1] (2019) in the last column of Table 6.1 for easy reference.

We see a sharp increase in unclassified and unknown websites. 62% of failures were due to the flows being too short, despite us crawling these websites for large web pages. This was because these websites often did not serve the requested page because they detected Gordon's measurements as a DoS attack. This is not surprising, since Gordon's methodology is extremely aggressive and can be inferred as malicious. In summary, it is not practical to use Gordon to classify CCAs run by websites today. Moreover, Gordon only focuses on TCP connections and can't classify flows serving web browsers and other real-world applications.

### 6.1.2 Why CCA Identification is Hard

At a high level, CCAs can be distinguished from each other based on how they react to different network conditions. Therefore, the basic approach of identifying CCAs on the Internet is relatively straightforward. We need to emulate different *network profiles* while connecting to a web server and measure how it reacts to them using some metric. Here,

we define the network profile as some combination of bandwidth and delay constraints enforced by the measurement tool on the connection.

However, the devil is in the details, and given the large number of CCAs that are available and deployed on the Internet today, identifying them is not as simple. Moreover, there are additional challenges associated with identifying CCAs on the Internet in a future-proof way. In this section, we articulate the key challenges for CCA identification and the requirements for further extensibility.

**Establishing Causality**. To accurately identify a CCA, we need to be confident that the response seen is indeed caused by our network profile and not any other natural variation in the network. This was less of a problem for early tools that used network profiles that generated responses that were unlikely to naturally occur on the Internet. For example, TBIT [19, 20] drops all the packets after initiating a connection and CAAI [21] delays ACKs by 1 second. Since both these behaviors rarely happen on the Internet, the responses they elicit can be safely assumed to be caused by the measurement tool and not the network. However, as discussed earlier, this is not effective when classifying more sophisticated CCAs. To address this, more recent tools like Inspector Gadget [47] and Gordon [1] create a localized bottleneck and apply more generic network profiles to it. Since most network variations happen at the bottleneck, this limits the possibility of natural variation on the Internet impacting the connection. Nebby also adopts this strategy to ensure causality is maintained between the network profile and the CCA's response.

**Handling Noisy Measurements**. Cross-traffic and network bottlenecks between the probing server and the target server will naturally introduce noise in the measurements. There is a need to eliminate noisy measurements. The general approach is to repeat measurements to eliminate outliers [1]. Gordon also attempted to do so by repeating measurements from different vantage points [1]. Gordon also repeats a measurement up to 5 times if it is not able to successfully classify it, albeit from the same

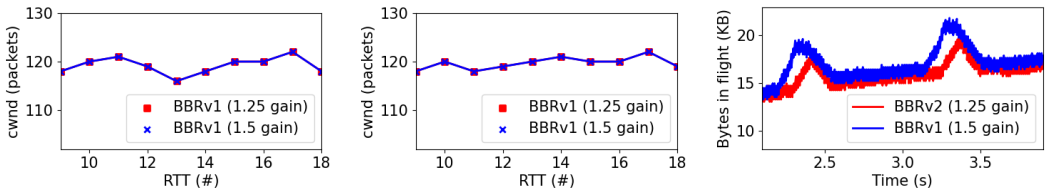
**Table 6.2:** Properties of CCA Identification tools.

| Tool | Primary Design Goals | | | | Extensibility Requirements | | |
|---|---|---|---|---|---|---|---|
| | Causality | Robustness to Noise | Identify Unknown CCAs | Cannot seem Hostile | Good Metric | Works with Encryption | Client Agnostic |
| TBIT [19, 20] | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| CAAI [21] | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| IG [47] | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ |
| Gordon [1] | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Nebby | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

vantage point. This will allow us to determine if websites deploy different CCAs in different regions. Within each trace, we have many repeated segments, so we have many opportunities to pick out the characteristic shape of the CCA even when there is noise.

**Handling New & Unknown CCAs**. Given that we expect more new and unknown CCAs to be deployed in the wild, a modern CCA identification tool must also be able to provide insight into the behavior of CCAs that it is not able to classify. Subsequently, it should be able to determine that they are substantively different from known CCAs. This is becoming increasingly important with the deployment of new and modified CCAs in QUIC [5]. The early tools were mainly classification tools [19, 21] that attempt to classify CCAs among a known set of CCAs and not able to detect new variants. Machine-learning-based like Inspector Gadget [47] and the work of Chen et al. [48] fare no better. Gordon was the first tool that conclusively detected an unknown and undocumented variant, by showing that Akamai deployed their own CCA variant [1], which Mishra et al. called AkamaiCC.

**Probe Traffic Cannot Seem Hostile.** Padhye and Floyd had clearly articulated that a CCA identification tool must not generate traffic that would be construed to be malicious to a web server [19]. In this light, it was surprising that Gordon [1] even worked at all in 2019, given that it opens connections hundreds of times and drops a large number of packets. Modern DDoS defenses have since kicked in and today they block Gordon for many websites. Nebby adopts a mostly light-weight approach when probing websites, and does not even introduce packet drops, unlikely previous tools[1, 19, 21].




**(a)** Gordon [1] **(b)** Inspector Gadget [47] **(c)** *Nebby*

**Figure 6.1:** Comparison of `cwnd` to `BiF` for BBRv1's `ProbeBw` phase.

**Need for a Good Metric**. Any response to a given network profile must be measured as a change in some metric, like the sending rate or the `cwnd`, which is defined as the maximum number of unacknowledged packets. The sending rate of a web server is hard to measure accurately since it will be morphed by every bottleneck it encounters on its path to the receiver. All previous tools therefore measure how the `cwnd` of a CCA changes during a connection to identify it. However, While `cwnd` was shown to be sufficient for identifying window-based CCAs and BBR [1], measuring the `cwnd` is not sufficient to differentiate between rate-based CCAs.

The reason why `cwnd` does not work well for rate-based CCAs is that `cwnd` is typically used by rate-based CCAs as a safeguard, and not an operating point. Given that using the `cwnd` can mask a sender's true behavior and estimating their sending rate is not practical, Nebby elects to measure bytes in flight (`BiF`), which is defined as the instantaneous number of unacknowledged packets, to determine a target server's response to a network profile (see details in §6.2.1).

In Figure 6.1, we plot the `cwnd` and `BiF` for two versions of BBR with the same `cwnd` but different pacing gains (1.25 and 1.5). It is clear that `cwnd` measurements (from Gordon and Inspector Gadget) do not differentiate between these two different versions of BBR, while `BiF` measurements (from Nebby) allow us to tell them apart. Since all previous CCA identification tools [1, 19, 20, 21, 47, 48] measure `cwnd` changes, they will not be extensible to the current crop and future iterations of rate-based CCAs.

**Handling encrypted packets**. The latest challenge posed to identifying CCAs on the Internet is transport protocols like QUIC. Since all previously discussed CCA identification tools were designed for TCP, they utilize the sequence and ACK numbers that are exposed unencrypted in the TCP header. However, QUIC packets are completely encrypted and only expose the source and destination IPs and the flow id. Therefore, previous approaches are not directly extensible to QUIC. To support non-TCP and QUIC flows, we need to be able to measure the response to a network profile without the need for sequence numbers and ACK numbers.

**Need to be Client Agnostic**. Earlier tools [1, 19, 20, 21, 47, 48] were able to classify traffic serving only a small set of TCP-based clients (like `wget` and `curl`). Since the choice of CCA can be expected to be influenced by the kind of application it supports, an ideal CCA identification tool needs to support a large variety of clients. in fact, for a modern CCA identification tool to be extensible and future-proof, there is a need for the tool to be client-agnostic.

**Summary**. In Table 6.2, we summarize how existing tools and Nebby address the various challenges and extensibility requirements. Our key contributions are twofold: we found a more accurate metric (`BiF`) that works well even for rate-based TCP variants and also added support for identifying CCA variants deployed on modern QUIC stacks and web browsers.

## 6.2 Methodology

Our general approach is relatively straightforward: we start a flow from a client to the target web server. The packets in the flow (both data and ACKs) are recorded at a local bottleneck (that we call the *capture point*) along the path between the client and server (that we will refer to as the *pipe*). In addition to recording the packets, the capture point is able to change the bandwidth available to the flow, introduce additional delay,

and also drop packets.

The captured trace is then processed with a classifier (§6.2.4) to identify the CCAs. By decoupling capturing of the trace and the classification process, we make it possible for the accuracy of the system to be improved incrementally without having to re-do our measurements. Our high-level approach is no different from previous approaches.

The key innovation in our work are in how we address the challenges laid out in §6.1.2 to ensure backward compatibility and future extensibility, as follows:

1. Instead of attempting to estimate `cwnd`, we estimate `BiF`[2] by introducing additional delay at the capture point (§6.2.1);
2. We developed techniques to handle QUIC packets (§6.2.2);
3. We use a minimal set of two network profiles that we show is sufficient to identify all 12 TCP variants in Linux kernel v5.18 and BBRv2 (§6.2.3);
4. We designed an extensible classifier that can identify all existing TCP variants in Linux kernel (§6.2.4) and show that it can be easily extended to identify new TCP variants (§6.3.3); and
5. Our approach can also be extended to support new applications and multiple flows (§6.2.5).

### 6.2.1 Estimating Bytes in Flight (`BiF`)

Consider a server and a client as shown in Figure 6.2. In a controlled lab setting, we can measure the `BiF` of a flow by setting up the capture point near the server because everything between the capture point and the client is visible. This is not possible for a remote server on the Internet, since the capture point can only be set up near the client. In such a situation, it is difficult to estimate the `BiF` accurately since the majority of the packets in the pipe are not visible.

Our key insight is that if we artificially introduce additional delay between the capture

[2]For loss-based (AIMD) TCP variants, `cwnd` and `BiF` are equivalent.

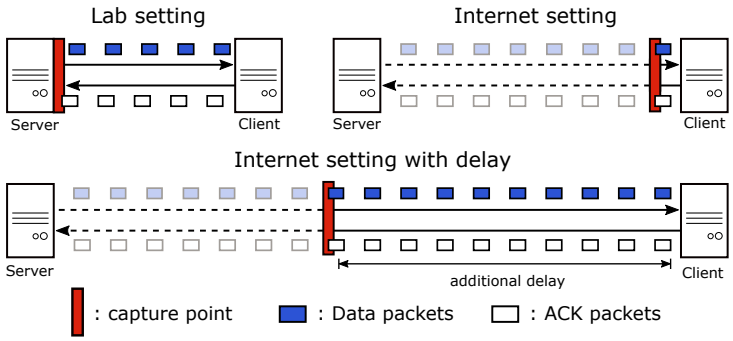



**Figure 6.2:** Using additional delay to increase the ratio of *visible* in-flight packets.

point and the client, we will then have visibility over a larger portion of the pipe and hence will be able to estimate `BiF` more accurately. In particular, because current Internet latencies are small (∼20 ms [129]), by introducing a latency of $x$ ms, we can effectively have visibility over $\frac{x}{x+20}$ of the pipe.

To determine the additional delay required, we set up control servers running different CCAs on AWS and measured their `BiF` with different amounts of delay introduced. We then compared these measurements with the ground truth `BiF` values exported directly from the sockets of these control servers. We plot the results of these experiments in Figure 6.3 for a server running CUBIC, Reno, and BBR. From these results, we see that the accuracy is close to 100% when the total additional delay is larger than 90 ms. Beyond this, accuracy might even drop. We found that this trend was consistent for all the CCAs available in the Linux kernel.

For TCP packets, we record the largest ACK and sequence numbers at the capture point and use the difference between them to estimate the `BiF`. We also track re-transmissions and lost packets to correct `BiF` estimates accordingly.

### 6.2.2 Handling QUIC packets

QUIC packets are encrypted and sequence numbers are not visible. In fact, there is no way to tell if a QUIC packet is a data packet or an ACK packet from the raw packet

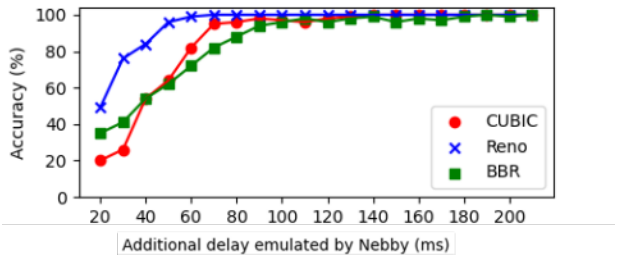



**Figure 6.3:** Impact of the additional delay on accuracy.

trace. Even if we were able to make this distinction, it is difficult to estimate how many bytes each ACK packet is acknowledging without access to the sequence numbers.

We address these uncertainties by making some realistic assumptions about a QUIC connection: (i) During a connection, we assume that all QUIC packets originating from the remote server are data packets and all the packets originating from the client are ACK packets. Since we care only about the `BiF` of the CCA running at the server and not the client, this assumption introduces no inaccuracies in our measurements. (ii) We also assume that each ACK packet acknowledges a fixed number of bytes. We estimate this value by dividing the total bytes transmitted by the server during the connection by the total number of ACK packets sent by the client. This is a fair assumption because most QUIC servers use a constant ACK frequency for the entire connection.

We exported the actual `BiF` logs from a `quiche` [104] sender running on an AWS instance and then compared it with what Nebby was measuring from a client machine. We ran this experiment for 20 trials in two different AWS regions. We found the accuracy of our `BiF` estimates for QUIC to be higher than 97%.

### 6.2.3 Minimal Set of Network Profiles

To recap, a *network profile* is a combination of some bandwidth, delay, and buffer constraints applied at the capture point, together with some actions like packet drops. Each network profile is an opportunity to differentiate between CCAs based on how they re-

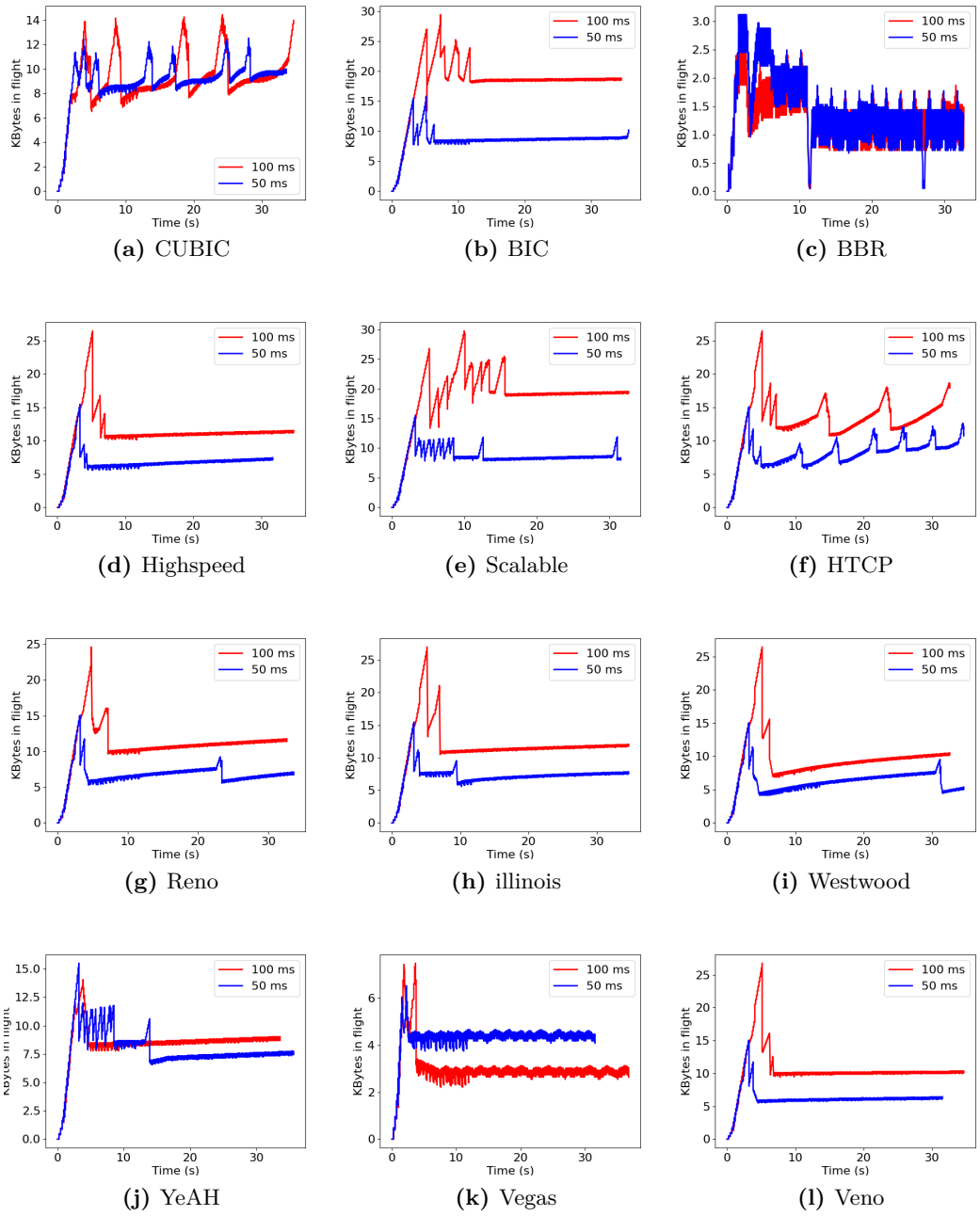
spond to that network profile. It is entirely possible that two different CCAs may have the same response to a network profile. In such cases, we need to perform measurements over additional network profiles that can help us differentiate between them. Therefore, the goal is to find the minimum number of network profiles required to differentiate between all the CCAs. Given that there are currently 12 available TCP variants in the Linux kernel v5.18 (and BBRv2 in Google's custom kernel), it was not surprising that we were not able to find one network profile that was able to allow us to distinguish between all of them.

However, we found that we were able to identify all 13 known CCAs with just 2 network profiles and without the need to introduce arbitrary packet drops, unlike previous approaches [11, 21]. By not introducing arbitrary packet drops, it not only simplifies the design of the network profile, but it also makes it less likely that our connection would be perceived as malicious by a remote sender. However, Nebby does allow *natural* packet drops to happen when there is a buffer overflow at its localized bottleneck buffer.

**Impact of bottleneck bandwidth**. With a smaller bottleneck bandwidth, it would take longer to download a given webpage. This means that we would see a longer measurement trace, and provide us with more information from a single measurement. However, we found the `BiF` traces to be extremely noisy at bandwidths lower than 100 Kbps. On the other hand, if the emulated bottleneck bandwidth was larger than 200 Kbps, the noise was significantly reduced.

We crawl all the target websites to find the largest available webpage. For all the target websites, we were able to find a page that was at least 400 KB in size. This meant that all our measurements would be at least 16 s long. In practice, all our measurements were longer than 18 s because of slow start.

**Set of 2 Network Profiles**. Nebby makes all measurements with a bottleneck bandwidth of 200 Kbps and the bottleneck buffer set at 2 BDP. Surprisingly we found that by introducing a 50 ms one-way delay using Mahimahi [125], the network profile




**(a)** CUBIC **(b)** BIC **(c)** BBR

**(d)** Highspeed **(e)** Scalable **(f)** HTCP

**(g)** Reno **(h)** illinois **(i)** Westwood

**(j)** YeAH **(k)** Vegas **(l)** Veno

**Figure 6.4:** Traces of TCP congestion control algorithms in the current Linux kernel.

would result in visually distinct `BiF` graphs for all 13 known CCAs. However, since some CCAs can have similar shapes under this network profile (like New Reno, Illinois, and HSTCP (see Figures 6.4g, 6.4h, and 6.4d)), we also added an additional network profile with a larger delay (100 ms one-way delay). We plot the `BiF` vs time graph for all the CCAs in the Linux kernel under both these network profiles in Figure 6.4.

### 6.2.4 Designing an Extensible Classifier

Our classification methodology is based on the observation that all CCAs, regardless of their underlying philosophy, will eventually converge to a stable operating point in the steady state. This is often called the *congestion avoidance* phase. This is typically done in one of two ways: (i) they either oscillate about their target operating point using congestion signals as periodic negative feedback (loss-based AIMD/MIMD CCAs) or (ii) they try to predict the correct operating point by explicitly modeling the network path (BBR and its many versions). Even if a CCA takes the later approach, the sender will have to probe the network periodically to obtain accurate measurements to update its network model. For BBR, these probing behaviors take form of the `ProbeBW` and `ProbeRTT` phases.

To use this periodicity in a CCA's behavior to classify existing TCP variants currently available in the Linux kernel, including BBRv2[26], our classifier extracts these periodic behaviors from `BiF` measurement traces and classifies them as different CCAs based on their periodicity and *shape* (see Figure 6.5). For the set of known CCAs in the kernel, we developed 2 classifiers: one for loss-based CCAs and another for BBRv1/v2. We can easily extend Nebby to identify more CCA variants by adding new classifiers that can be run concurrently with our 2 current classifiers. We describe how this works in §6.3.3.

Before a trace can be used for classification, we first remove the noise with a low-pass filter ("smoothening") and segment the smoothened trace into several chunks. These segments are then sent to the different (currently 2) classifiers. This process is summarized

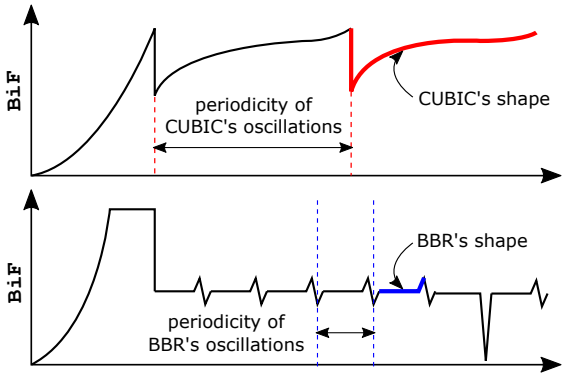



**Figure 6.5:** Characteristic features of periodic oscillations for CUBIC and BBR.

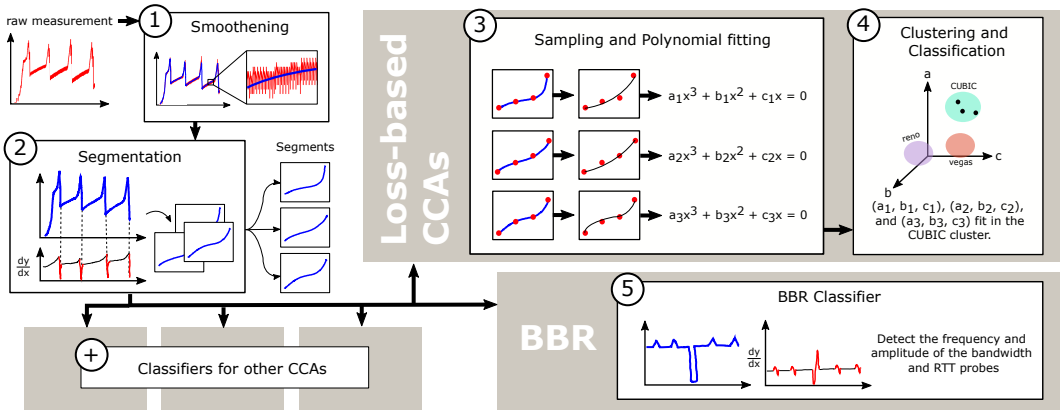



**Figure 6.6:** How Nebby's Classifier works.

in Figure 6.6. We describe the various components in detail below.

(1) **Smoothening**. The raw `BiF` traces captured can be noisy because of ACK compressions and cross traffic on the Internet. To remove this noise, we remove all the variations that happen at timescales that are shorter than an RTT (since they are likely to be introduced by the network and not the CCA) by doing an FFT on the `BiF` values and removing all component frequencies that are larger than $\frac{1}{RTT}$.

(2) **Segmentation**. Since most CCAs have similar slow start phases, we ignore slow start and attempt to identify CCAs from their behavior during congestion avoidance. As discussed earlier, CCAs typically exhibit periodicity during congestion avoidance, where they periodically probe for more bandwidth, and then eventually back off when they

encounter congestion, or if the buffer overflows. We extract these periodic waveforms as individual segments that are punctuated by periods of 'back-offs,' by computing the first derivative over the `BiF` trace and identifying the back-offs by their characteristic high negative gradients. We extract the regions between these back-offs as individual segments for a given trace. A typical trace generally gets divided into multiple segments.

**Loss-based (AIMD) Classifier**. Loss-based CCAs exhibit periodicity because they back off periodically after a buffer overflow. The key approach to classification is to fit all the input segments as polynomials and then classify based on the coefficients of these polynomials by comparing to the coefficients of the reference implementations in the Linux kernel.

**(3) Sampling and Polynomial Fitting**. The input segments will generally be of varying length. Hence, we first normalize all the `BiF` values for a segment between 0 and 1 and sample 200 points uniformly on the segment. We then try to fit first, second, and third-degree polynomials to these 200 points using `numpy`'s `polyfit` function. We do not go beyond third-degree polynomials because we found that a cubic polynomial is sufficient to capturing the shape of the most complicated waveforms that are generated by CCAs like CUBIC. Each of these polynomials is ranked based on the following error score:

$$Error = \text{MSE} + \lambda * \text{Degree} * \text{Sum(coefficients)}$$

Here, the MSE is the mean square error and $\lambda$ is a tunable parameter that can be set to anything between 0 and 1. We note that our error function bears a similarity to common Lasso regression functions and penalizes polynomials with higher degrees. This is because, in general, it is easy to over-fit higher-degree polynomials. We therefore need to add a penalty term for these higher degree polynomials. We empirically set $\lambda$ to 0.7 since it results in the clearest distinctions between different polynomials. The output of this procedure is up to 3 coefficients $a$, $b$, and $c$ for each segment.

④ **Clustering and Classification**. Once we have representative polynomials for all the segments for a measurement trace, we compare the shapes of these segments with those of known loss-based CCAs by comparing their coefficients ($a$, $b$, and $c$) to that for the fitted polynomials. As noted earlier, each trace will often yield multiple segments. In such cases, we classify a trace as a CCA, if (i) all or some of its component segments match a known CCA; and (ii) none of the segments match another known CCA. In other words, a trace will be classified as a known CCA if only some of its segments match a known CCA while the other segments are classified as unknown. There were no instances where the segments from a trace were matched to two different CCAs in any of our measurements.

To generate the control data required to derive the required coefficients for the known CCAs, we set up control servers in AWS instances in Singapore, Mumbai, Ohio, Paris, and Sao Paulo and measured them from a host in our laboratory using Nebby. The measurements were made using the two network profiles described in §6.2.3. Each CCA was run 50 times from each vantage point, giving us a total of 250 measurements for each CCA. For each CCA, we determined the representative polynomial using `polyfit` as the previously described, giving us a large number of polynomials for each CCA. Since each measurement can consist of multiple segments, we were able to derive hundreds of polynomials for each CCA from our 250 measurements. The polynomials assigned to the segments for the known CCAs in the Linux kernel are listed in Table 6.3. The clusters formed by all the coefficients of these polynomials are illustrated in Figure 6.7. We can see that with our chosen network profiles, the loss-based CCAs formed distinct clusters that allowed us to tell the different CCAs apart.

We tested all the coefficients generated by our control tests using the D'Agostino K2 test and the Shapiro-Wilk test to verify if these coefficients were normally distributed for all of the target CCAs. We used a soft-fail hypothesis, where if a CCA's coefficients passed either of the tests, they were considered to be normally distributed. All our

target CCAs passed at least one of the two tests. This allowed us to model each CCA's coefficients as a normal distribution, and treat each CCA's polynomial as a multivariate random variable. Since we can model each feature's polynomial as a multivariate random variable, it opens up the possibility of using a large variety of supervised learning algorithms. We chose to use Gaussian Naive Bayes (GNB) because it works with continuous data and does not require mapping the data to a higher dimensional feature space (as is the case for Support Vector Machines). We were reluctant to use such methods as it would mean that some new feature other than the polynomial coefficients which reflect the shape of the CCAs would be used to make the classification. As stated earlier, we only want to use the shape of the CCAs to classify measurements to avoid the risk of overfitting. GNB gives us this freedom. The prior for the GNB was set to be a discrete distribution which allows each CCA to be chosen with equal probability. The GNB classifier gives us a probability that a given feature can belong to a certain CCA. In practice, we have seen probabilities for most of our features to be skewed to only one CCA. In cases where multiple CCAs have equally high probabilities, we classify the feature as Unknown. As discussed earlier, a measurement can have multiple features. In such cases, a measurement is classified as a CCA only if all features in that measurement belong to that CCA. A measurement can also be classified as a known CCA if only some of the features belong to that known CCA while the other features are classified as unknown. We found no instances where features from the same measurements matched two different CCAs.

(5) **BBR Classifier**. Our BBR classifier classifies a trace as either as BBRv1, BBRv2, or Unknown. We identify the variant by scanning for BBR's characteristic periodic probing behavior, as follows:

- BBRv1 has a characteristic bandwidth probing behavior where it increases the sending rate by 25% every 8 RTTs (i.e. `ProbeBW`). This behavior is clearly visible to Nebby (as seen in Figure 6.1c) and easily detectable by looking for periodic spikes in the first

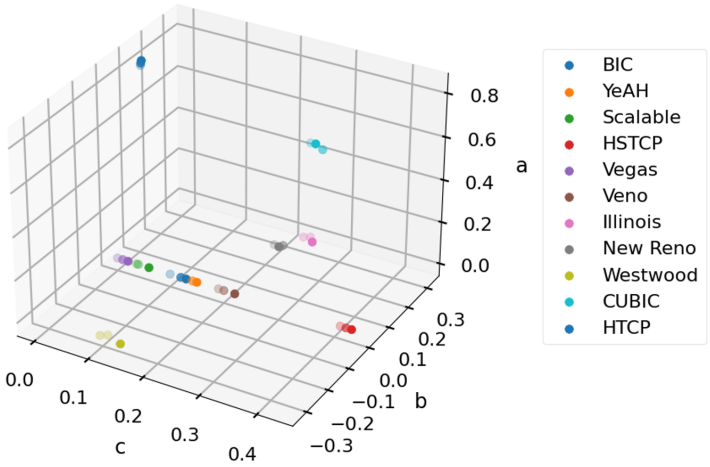



**Figure 6.7:** Coefficients for the polynomials ($ax^3+bx^2+cx+d=0$) of all the loss-based CCAs form distinct clusters.

derivative w.r.t. to the time of the extracted segments. BBRv1 also backs off every 10 seconds in order to estimate the minimum RTT of the path (i.e. `ProbeRTT`). Therefore, if we see a rate-based sender probe for bandwidth every 8 RTTs and backing off every 10 seconds, we conclude that the sender is BBRv1.

- The probing periods for BBRv2 are less well-defined. After slow start, BBRv2 typically enters its bandwidth cruise phase where it sends at the bottleneck bandwidth without any probing for a period that depends on the BDP. For our network settings, this probe period is about 2 seconds. Like BBRv1, BBRv2 also backs off periodically to measure the minimum RTT, albeit every 5 seconds. Therefore, if we see a rate-based send that is stable during congestion avoidance for at least 2 seconds and backs off every 5 seconds, we conclude that the CCA must be BBRv2.

**Handling New & Undocumented CCAs**. Given the design of our classification framework, it can extended naturally in 3 ways: (i) existing classifiers can be modified to use the segment corresponding to Slow Start; (ii) an additional classifier can be constructed from observing the properties of a new CCA (an example is described in §6.3.3); or (iii) additional network profiles can be added.

**Table 6.3:** Different degree clusters with their CCAs.

| Linear | Quadratic | Cubic |
|---|---|---|
| BIC, YeAH, Scalable, HSTCP Vegas, Veno | Illinois New Reno Westwood | CUBIC, HTCP |

### 6.2.5 Supporting Web Browsers & Multiple Flows using Selenium

One of Nebby's advantages over previous tools is that it can work with a larger range of applications. In addition to TCP measurements using `wget` and QUIC measurements using a `quiche` client, Nebby can also classify a flow generated by a live web browser. We used a simple Selenium[130] wrapper written in about 25 line of Python code to launch connections and stream dynamic web content like video via a Google Chrome browser.

This allows us to capture the same sequence of flows an application usually generates while accessing the Internet. Since a browser can launch multiple concurrent connections, we ran a modified version of Nebby with our Selenium client that creates a separate bottleneck queue to isolate each connection so that each flow can be classified separately. Our clients also generate a HAR (HTTP Archive) file after every connection to allow us to correlate individual flows to individual asset requests.

## 6.3 Evaluation

In this section, we evaluate Nebby's accuracy and present our results for measurements over `wget` (TCP), `quiche` (QUIC), and a selenium web browser (TCP). All measurements were done from five viewpoints (AWS datacenters) around the world, namely Ohio, Paris, Mumbai, Singapore, and Sao Paulo. All traces, unless specified, were collected between June 2023 and October 2023. Nebby was implemented in 100 lines of Bash and 900 lines of Python. Nebby is open-source and available on Github [131].

**Table 6.4:** Classification accuracy.

| | Classified as | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | BBRv1 | BBRv2 | BIC | CUBIC | HSTCP | HTCP | Illinois | New Reno | Scalable | Vegas | Veno | Westwood | YeAH | Unknown |
| BBRv1 | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| BBRv2 | 6% | **94%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| BIC | 0% | 0% | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| CUBIC | 0% | 0% | 5% | **95%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| HSTCP | 0% | 0% | 0% | 0% | **98%** | 2% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| HTCP | 0% | 0% | 0% | 0% | 0% | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| Illinois | 0% | 0% | 0% | 0% | 0% | 0% | **88%** | 8% | 0% | 4% | 0% | 0% | 0% | 0% |
| New Reno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **100%** | 0% | 0% | 0% | 0% | 0% | 0% |
| Scalable | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 8% | **92%** | 0% | 0% | 0% | 0% | 0% |
| Vegas | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 2% | 0% | **98%** | 0% | 0% | 0% | 0% |
| Veno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **100%** | 0% | 0% | 0% |
| Westwood | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 8% | 0% | 0% | 0% | **92%** | 0% | 0% |
| YeAH | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **100%** | 0% |

### 6.3.1 Measurement Accuracy and Usability

To determine the accuracy of our Nebby with its 2 current classifiers, we set up controlled web servers on the AWS cloud in 5 regions around the world: Ohio, Paris, Mumbai, Singapore, and Sao Paulo. Each of these control servers was configured to run all the CCAs variants available in the Linux kernel v5.18. We achieved an average accuracy of 96.7% (see details in Table 6.4). The level of accuracy is comparable to the state-of-the-art tool Gordon [1], except that we achieve slightly better accuracy for CUBIC and BBR. While both Gordon and Nebby use shape-based classifiers, Gordon's measurements are much more coarse-grained (one data point per RTT) compared to Nebby (one data point per packet). As a result, Nebby can distinguish between all the variants in the Linux kernel, unlike Gordon, which was not able to distinguish between some pairs of TCP variants like Veno and Vegas. As discussed in §6.1.2, this is because `BiF` is a better metric than the `cwnd` metric used by Gordon. In terms of usability, in 2023, Gordon was able to identify CCAs for only 4% of Alexa Top 20k websites when measured from the Singapore region. This is because it creates traffic patterns deemed hostile by websites (discussed in §6.1.2). Nebby on the other hand identified ∼78% of Alexa Top 20k websites when measured from the Singapore region (Table 6.5).

**Table 6.5:** Distribution of CCA variants among Alexa Top 20k websites measured from different viewpoints by Nebby.

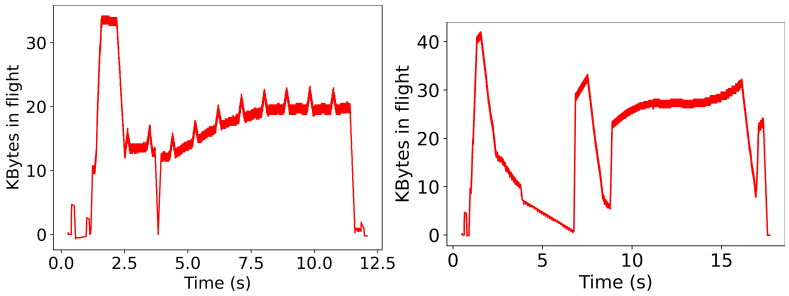
| Variant | Ohio | | Paris | | Mumbai | | Singapore | | Sao Paulo | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Websites | Share | Websites | Share | Websites | Share | Websites | Share | Websites | Share |
| BBRv1 | 2,594 | 13% | 1,900 | 9.5% | 1,670 | 8% | 2,541 | 12.7% | 1,280 | 6.4% |
| BBRv2 | 515 | 2.6% | 373 | 1.9% | 12 | 0.1% | 251 | 1.3% | 0 | 0% |
| BIC | 712 | 3.5% | 807 | 4% | 837 | 4.2% | 402 | 2% | 227 | 1.1% |
| CUBIC | 8,202 | 41% | 8,406 | 42% | 8,822 | 44.1% | 8,673 | 43.4% | 6,982 | 34.9% |
| HSTCP | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% |
| HTCP | 583 | 2.9% | 370 | 1.9% | 522 | 2.6% | 421 | 2.1% | 363 | 1.8% |
| Illinois | 721 | 3.6% | 684 | 3.4% | 1,121 | 5.6% | 625 | 3.1% | 229 | 1.1% |
| New Reno | 1,840 | 9.2% | 1,509 | 7.5% | 3,032 | 15.2% | 2,093 | 10.5% | 2,683 | 13.5% |
| Vegas | 878 | 4.4% | 421 | 2.1% | 301 | 1.5% | 526 | 2.6% | 511 | 2.5% |
| Veno | 112 | 0.6% | 382 | 1.9% | 52 | 0.3% | 21 | 0.1% | 11 | 0.1% |
| Westwood | 201 | 1% | 170 | 0.9% | 0 | 0% | 0 | 0% | 0 | 0% |
| Scalable | 18 | 0.1% | 0 | 0% | 0 | 0% | 6 | 0% | 0 | 0% |
| YeAh | 123 | 0.6% | 89 | 0.4% | 64 | 0.3% | 0 | 0% | 112 | 0.6% |
| Unknown | 3,501 | 17.5% | 4,889 | 24.4% | 3,621 | 18.1% | 4,441 | 22.2% | 7,602 | 38% |
| Unresponsive | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% |
| Total | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% |

### 6.3.2 Results for Alexa Top 20k Websites

To understand how the Internet's congestion control landscape has evolved since the last measurement study done by Mishra et al. [11] in 2019, we used Nebby to identify the CCA variants for the Alexa Top 20,000 websites from the 5 aforementioned viewpoints (Ohio, Paris, Mumbai, Singapore, and Sao Paulo). These measurements were made using a `wget` client over TCP. We crawled all the target websites for the largest web pages we could find to record the longest possible measurement traces. We present our results in Table 6.5.

By comparing our results to those from the 2019 study [11], we make the following observations:

1. **Different deployments across regions.** in [11], it was reported that websites deployed the same CCAs in all 5 regions. Our measurements using Nebby suggest that this is no longer the case. As is clear from Table 6.5, websites deploy variants like CUBIC, BBR, and Reno differently in different regions. Overall, Nebby detected that 13.6% of the websites were deploying different variants in different regions. About half of these websites (7% of the total) were electing to use CUBIC in Mumbai and/or




**(a)** Ohio (BBRv1) **(b)** Mumbai (CUBIC)

**Figure 6.8:** Traces for `amazon.com` in different regions.

**Table 6.6:** Websites (11%) found to deploy New Reno.

| Previously classified by Gordon [11] as | |
|---|---|
| New Reno | 1% |
| Vegas | 1% |
| Unknown | 5% |
| Not measured | 4% |
| Total | 11% |

Sao Paulo while running BBR in all other regions. An example of one such website is `amazon.com` which was served in all regions using BBRv1 except Mumbai, where we found it being served using CUBIC (see Figure 6.8).

2. **Websites deploying New Reno**. Mishra et al. had earlier in 2019 reported that New Reno was deployed on only 0.8% of websites [1]. In contrast, Nebby found New Reno to be deployed on some 11.1% of the 20,000 surveyed websites. On further investigation, we found that among these 11% of websites, 5% were previously unclassified ('Unknown') and 4% were not present in the earlier list of Alexa top 20,000 sites. A small number (1%) were classified by Gordon as Vegas. This is summarized in Table 6.6.
3. **Adoption of BBRv1.** We found a slight dip in the absolute number of websites that choose to deploy BBR in the Alexa Top 20,000 websites. Even in Ohio, which is the region with the largest deployment of BBR, the number of websites using BBR

has dropped from 18% in 2019 to 15.5% in 2023. That said, it should be noted that the Alexa list [132] has evolved significantly since the last measurement study. Overall, we only share 52% of the same measured websites with [11]. Among these common websites, 12% of them (6% of the total) have migrated from using BBR in 2019 to CUBIC in 2023. A large number of these websites (9%, about 4.5% of the total) are hosted by Cloudflare. Some of these notable websites are `bbc.com` and `wikihow.com`. It remains true that most websites choosing to deploy BBR tend to serve video workloads, adult content, or large files (for example, `mega.nz`). We summarize the CCAs deployed by some of these 'heavy-hitter' websites in Table 6.7.

4. **Slow adoption of BBRv2.** Even though Google itself reportedly switched from BBRv1 to BBRv2 back in 2020, the adoption of BBRv2 seems to be slow. Only about 5% of the websites we identified as deploying BBRv2 in Ohio had upgraded from BBRv1 back in 2019. This suggests that most websites that deploy BBRv2 today are *new* adopters of BBR. Some examples of these new adopters are `rakuten.com` and `primevideo.com`, both of which deployed CUBIC in 2019. More than 98% of the websites that were deploying BBRv1 in 2019 and were also measured by Nebby are either still deploying BBRv1 (86%), or have switched to CUBIC (12%).

**Catching the deployment of BBRv3.** During our measurements in June 2023 we noticed that all of the `google` domains and `youtube.com` were deploying a version of BBR that was neither BBRv1 or BBRv2 (see Figure 6.9). We hypothesized that what Nebby had actually measured in June 2023 was an early deployment of BBRv3, which was released to the community only in August 2023. We confirmed this finding with Google [133].

### 6.3.3 Extending Nebby to identify new CCAs

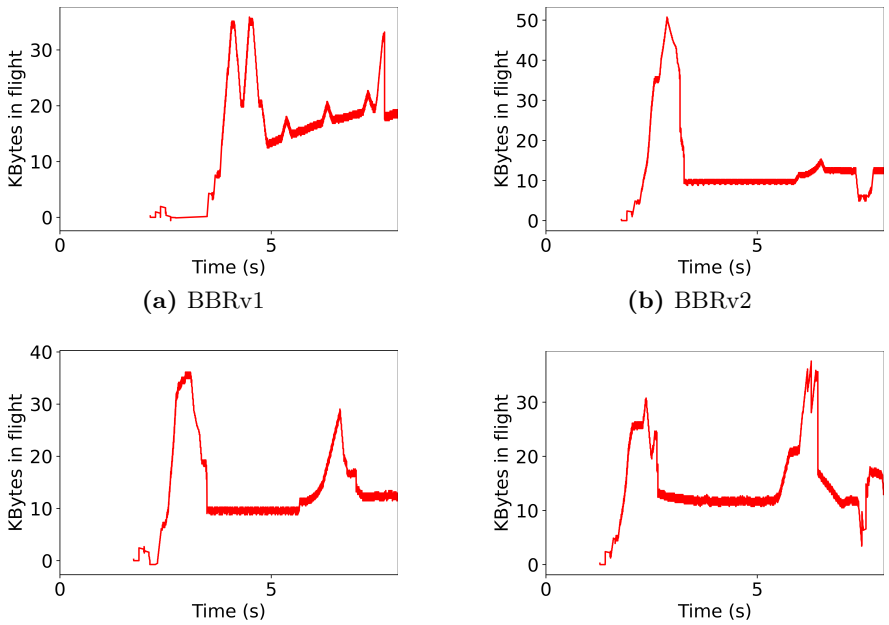
In 2019, Mishra et al. reported that Akamai deployed an undocumented variant that they referred to as AkamaiCC [1]. In 2023, we confirmed that websites hosted by Akamai

**Table 6.7:** CCAs deployed by most popular websites on the Internet by traffic-share.

| Websites | Traffic share [98] | CCA |
|---|---|---|
| `google` domains | 13.85% | BBRv3 |
| `netflix.com` | 13.74% | Reno |
| `facebook.com` | 6.45% | CUBIC |
| `apple.com` | 4.59% | AkamaiCC |
| `disneyplus.com` | 4.49% | CUBIC |
| `amazon.com` | 4.24% | BBRv1 |
| `tiktok.com` | 3.93% | AkamaiCC |
| `primevideo.com` | 2.67% | BBRv2 |
| `hulu.com` | 2.44% | AkamaiCC |

continue to deploy an undocumented variant. Two of these sites were Apple and Hulu and the corresponding traces are shown in Figures 6.10a and 6.10b. As shown in these figures, the defining characteristic of this undocumented CCA is that it would typically send data at some fixed rate for several seconds before backing off. This backoff was not triggered by dropped packets or any bandwidth limits. The fixed send rate did not seem to be determined by either the BDP or the RTT. This behavior is different from what was observed by Mishra et al. where they found the `cwnd` to be proportional to the BDP. It is therefore likely that AkamaiCC has evolved since the last measurement study in 2019, or Akamai might have deployed a CCA different from the one that was deployed in 2019.

Given these observations, we wrote a pluggable classifier for Nebby that detected if a flow backed off in intervals between 10 to 20 s, and maintained `BiF` at a consistent level between these back-offs. Since we do not have the ground truth for AkamaiCC, the classification parameters were determined from traces obtained from 10 known Akamai-hosted websites that Nebby originally classified as 'Unknown'. We add this new AkamaiCC classifier to our original set of 2 classifiers (loss-based and BBR). By running this new classifier over our full data set, the CCAs for all the known Akamai-hosted websites (approximately 6%) were identified as AkamaiCC. This demonstrates that Nebby is easily extensible to new CCAs beyond the known and documented CCAs. In addi-




**(a)** BBRv1 **(b)** BBRv2

**(c)** google.com (Jun 2023) **(d)** BBRv3 (Aug 2023)

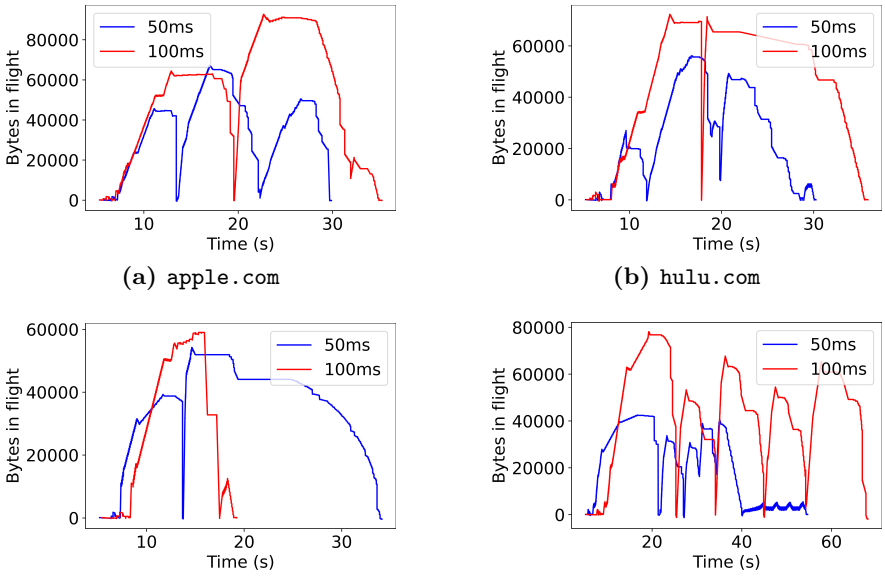
**Figure 6.9:** Catching the deployment of BBRv3 in the wild in Aug 2023.

tion to these Akamai-hosted websites, we also found another 1% of websites (that were not hosted by Akamai) that deployed an AkamaiCC-like variant. Two such examples (TikTok and Pornhub[3]) are shown in Figures 6.10c and 6.10d.

### 6.3.4 CCA Implementations in QUIC Stacks

QUIC [23] is quickly gaining popularity on the Internet and is set to be the standard transport with HTTP3. Meta already supports 75% of its traffic using its `mvfst` QUIC stack [134]. Mishra et al. had earlier shown that the implementations of standard TCP variants in existing QUIC stacks can be quite different from that of the kernel [33]. In Table 6.9, we list the open-sourced stacks that we investigated and the available CCAs for them. Since QUIC implementations can behave significantly differently from their

[3]No pornographic material was watched in the course of this research. All data access was done with a headless browser.




(a) `apple.com` (b) `hulu.com`

(c) `tiktok.com` (d) `pornhub.com`

**Figure 6.10:** Traces for websites deploying AkamaiCC.

kernel counterparts, we re-evaluated the accuracy of our classifier for all the QUIC stacks listed in Table 6.9.

In Fig. 6.11, we show some representative QUIC network profiles that we have found to be quite different from standard kernel implementations. Comparing to the network profiles for kernel implementation shown in Fig. 6.4, we can see that the profiles are mostly similar, except for `chromium` CUBIC, `quiche` CUBIC and `mvfst` BBR. We note that `chromium` CUBIC and `mvfst` BBR were identified by Mishra et al. to be non-conformant QUIC CCA implementations [33].

In Table 6.10, we produce the confusion matrix for the classification of some CCA implementations in these QUIC stacks. In general, our classifier works very well for most of the CCA implementations in the investigated QUIC stacks. Our classifier has an average accuracy of 92.8%. Our accuracy is not as high for `quiche` CUBIC (78%), `xquic` Reno (80%), and `mvfst` BBR (86%), but this is hardly surprising since these variants were earlier identified by Mishra et al. to be non-conformant QUIC CCA imple-

**Table 6.8:** Distribution of QUIC CCA variants as measured from different viewpoints on the Internet.

| Variant | Ohio | | Paris | | Mumbai | | Singapore | | Sao Paulo | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Websites | Share | Websites | Share | Websites | Share | Websites | Share | Websites | Share |
| CUBIC | 622 | 3.1% | 829 | 4.1% | 927 | 4.6% | 796 | 4% | 403 | 2% |
| BBR | 1,036 | 5.2% | 703 | 3.5% | 197 | 1% | 268 | 1.3% | 479 | 2.4% |
| BBRv2 | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% | 0 | 0% |
| New Reno | 25 | 0.1% | 10 | 0% | 11 | 0% | 18 | 0.1% | 10 | 0% |
| Unknown | 101 | 0.5% | 242 | 1.2% | 847 | 4.2% | 702 | 3.5% | 892 | 4.5% |
| Unresponsive | 18,216 | 91.1% | 18,216 | 91.1% | 18,216 | 91.1% | 18,216 | 91.1% | 18,216 | 91.1% |
| Total | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% | 20,000 | 100% |

**Table 6.9:** List of open-source QUIC/TCP stacks studied.

| Organization | Stack | CUBIC | BBR | Reno |
|---|---|---|---|---|
| Alibaba | `xquic` [110] | ✓ | ✓ | ✓ |
| Amazon Web Services | `s2n-quic` [109] | ✓ | ✗ | ✗ |
| Cloudflare | `quiche` [104] | ✓ | ✗ | ✓ |
| Go | `quicgo` [106] | ✓ | ✗ | ✓ |
| Google | `chromium` [102] | ✓ | ✓ | ✗ |
| H2O | `quicly` [107] | ✓ | ✗ | ✓ |
| LiteSpeed | `lsquic` [105] | ✓ | ✓ | ✗ |
| Meta | `mvfst` [101] | ✓ | ✓ | ✓ |
| Microsoft | `msquic` [103] | ✓ | ✗ | ✗ |
| Mozilla | `neqo` [111] | ✓ | ✗ | ✓ |
| Rust | `quinn` [108] | ✓ | ✗ | ✓ |

mentations [33]. By *non-conformant*, we mean that their behavior deviates significantly from their respective kernel CCA implementations. We added the Conformance [33] for all the benchmarked QUIC CCA implementations as an extra column in Table 6.10.

To investigate the CCAs deployed by websites that support QUIC, we repeated measurements of the Alexa Top 20,000 website by sending requests using `quiche`'s QUIC client using Nebby. We found that only 8.9% of the 20,000 sites responded to QUIC requests. Most of these websites supporting QUIC were hosted by Cloudflare or were Facebook domains. All these websites also deployed the same congestion control algorithms they deployed over TCP. Our results are summarized in Table 6.8. We found no evidence of undocumented variants being deployed on QUIC stacks. It seems likely that most of the websites that were classified as 'Unknown,' could not be classified because

**Table 6.10:** Confusion Matrix for QUIC CCA variants.

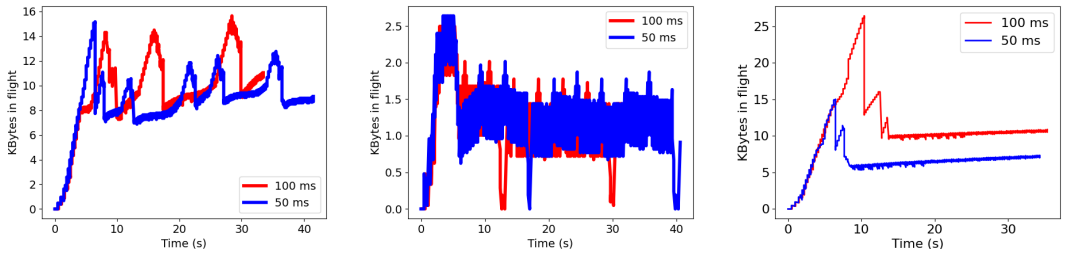
| | | Classified as | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Organization | Variant | BBRv1 | BBRv2 | BIC | CUBIC | HSTCP | HTCP | Illinois | New Reno | Scalable | Vegas | Veno | Westwood | YeAH | Unknown | Conformance[33] |
| Alibaba | `xquic` CUBIC | 0% | 0% | 0% | **88%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 12% | 0.55 |
| AWS | `s2n-quic` CUBIC | 0% | 0% | 3% | **92%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 5% | 0.76 |
| Cloudflare | `quiche` CUBIC | 0% | 0% | 12% | **78%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 10% | 0.08 |
| Go | `quicgo` CUBIC | 0% | 0% | 12% | **84%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 4% | 0.87 |
| Google | `chromium` CUBIC | 0% | 0% | 6% | **94%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0.6 |
| H2O | `quicly` CUBIC | 0% | 0% | 0% | **82%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 18% | 0.68 |
| LiteSpeed | `lsquic` CUBIC | 0% | 0% | 4% | **92%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 4% | 0.95 |
| Meta | `mvfst` CUBIC | 0% | 0% | 0% | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0.9 |
| Microsoft | `msquic` CUBIC | 0% | 0% | 0% | **98%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 2% | 0.7 |
| Mozilla | `neqo` CUBIC | 0% | 0% | 7% | **88%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 5% | 0 |
| Rust | `quinn` CUBIC | 0% | 0% | 0% | **90%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 10% | 0.7 |
| Alibaba | `xquic` BBR | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0.15 |
| Google | `chromium` BBR | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0.7 |
| LiteSpeed | `lsquic` BBR | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0.59 |
| Meta | `mvfst` BBR | **86%** | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 14% | 0 |
| Alibaba | `xquic` Reno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **80%** | 0% | 0% | 0% | 0% | 0% | 20% | 0.38 |
| Cloudflare | `quiche` Reno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0.8 |
| Go | `quicgo` Reno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **98%** | 0% | 0% | 0% | 0% | 0% | 2% | 0.92 |
| H2O | `quicly` Reno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0.8 |
| Meta | `mvfst` Reno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0.94 |
| Mozilla | `neqo` Reno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **92%** | 0% | 0% | 0% | 0% | 0% | 8% | 0.62 |
| Rust | `quinn` Reno | 0% | 0% | 0% | 0% | 0% | 0% | 0% | **100%** | 0% | 0% | 0% | 0% | 0% | 0% | 0.96 |

of noisy measurements.

### 6.3.5 Video Measurements with *Selenium*

In addition to measuring websites over `wget` and QUIC, we also measured websites using a standard web browser as described in §6.2.5. One of Nebby's strengths over its predecessors is that it allows us to study streaming and interactive applications. We measured popular websites while streaming video on demand (VOD), live video, audio, and while being on a video call. Since such applications tend to open multiple concurrent connections, we use a modified version of Nebby that assigns separate bottleneck queues to each flow. Our results are summarized in in Table 6.11. Overall, we noticed the following key trends.

**BBR is the preferred CCA for video traffic**. To investigate if certain CCAs were preferred for certain asset types, we identified which flows were serving which elements in the webpage and correlated their CCAs with them. We found that most audio and video streaming websites like Primevideo, AppleTV, Spotify, Apple Music, YouTube, Douyin,




(a) `quiche` CUBIC (b) `mvfst` BBR (c) `xquic` New Reno

**Figure 6.11:** Traces of non-conformant CCAs implemented in popular QUIC stacks.

**Table 6.11:** CCAs serving popular web services running on a Selenium client.

| Website | Region of popularity | Activity | Connections | Max Concurrent Connections | CCAs for Audio/Video Traffic | CCAs for Static Assets |
|---|---|---|---|---|---|---|
| Netflix | Global | VOD | 28 | 5 | **New Reno*** [135] | New Reno, CUBIC |
| Primevideo | Global | VOD | 12 | 6 | BBR | BBR |
| AppleTV | Global | VOD | 16 | 6 | BBR | BBR, CUBIC |
| Disney+ | Global | VOD | 20 | 6 | CUBIC | CUBIC |
| HBO | Global | VOD | 10 | 4 | BBR | CUBIC |
| Tiktok | Global | VOD | 21 | 4 | AkamaiCC | AkamaiCC, CUBIC |
| YouTube | Global | VOD, live video | 81 | 6 | **BBRv3*** [133] | **BBR*** [133] |
| Twitch | Global | VOD, live video | 118 | 6 | BBR | CUBIC |
| Spotify | Global | VOD, streaming audio | 8 | 5 | **BBR*** [136] | **BBR*** [136] |
| Apple Music | Global | streaming audio | 16 | 6 | BBR | BBR, AkamaiCC |
| Zoom | Global | video call | 39 | 6 | BBR | CUBIC |
| Meet | Global | video call | 60 | 5 | **BBRv3*** [133] | **BBR*** [133] |
| Hulu | US | VOD | 41 | 6 | AkamaiCC | AkamaiCC |
| Douyin | China | VOD | 5 | 6 | BBR | BBR |
| Bilibili | China | VOD | 10 | 3 | BBR | BBR |
| Hotstar | India | VOD | 12 | 5 | BBR | BBR |
| Jiocinema | India | VOD | 12 | 6 | CUBIC | CUBIC |

* Verified through personal correspondence or public tech blog posts.

Bilibili, Twitch, HBO, and Hotstar used some version of BBR to stream audio and video. However, there were a few exceptions to this rule, namely Netflix (New Reno), TikTok and Hulu (AkamaiCC), and Disney+ and Jiocinema (CUBIC).

**A website can use different CCAs for different flows.** We also found instances where websites were using different CCAs to deliver different assets. For example, we observed AppleTV, Twitch, and HBO use BBR to stream video and CUBIC to load static assets like banner ads. It is likely this variation exists because different CDNs cache these assets. Simple websites with only static web content always had all their data delivered by only one CCA.

**Inter-flow interaction for the same websites.** It is well known that CUBIC and BBR flows do not mix well [3, 25]. Therefore, when we saw some websites using a combination of CUBIC and BBR flows to deliver their web pages, we were curious to see how these flows would interact with each other. Therefore, for these webpages alone, we ran Nebby in its default single bottleneck setting to see how these flows would interact. Interestingly, for AppleTV, we found CUBIC and BBR flows interacting and negatively impacting each other's performance. We often observed that a CUBIC flow delivering a banner ad on `appletv.com` could cause degradation to the long-running BBR flow that was delivering video chunks for the video player. While this exact interaction might be an artifact of Nebby's constraint bandwidth setting, this is enough evidence that developers need to be careful about how they deploy CCAs to avoid causing performance issues inadvertently.

## 6.4 Discussion

Our measurement results have raised many questions on the future of the Internet's congestion control landscape. Our methodology also has scope for improvement on many fronts. In this section, we discuss these questions and future work.

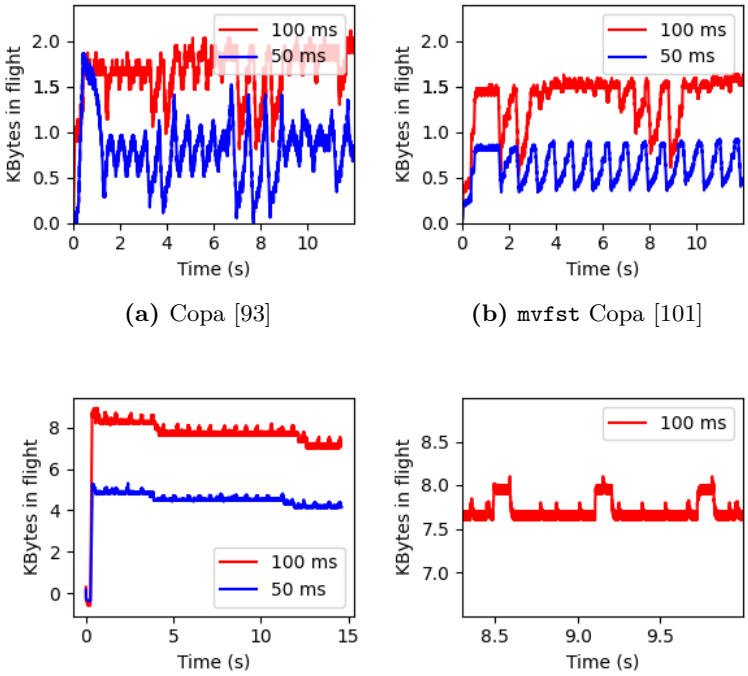
**Internet Evolution**. The original motivation for this study was the question: how has the congestion control environment of the Internet changed? Given the rapid adoption of BBR in 3 years since it was introduced in 2016, the burning question in 2019 was whether BBR would eventually replace CUBIC as the dominant congestion control on the Internet. In 2022, Mishra et al. modeled the interactions between CUBIC and BBR and hypothesized that the adoption of BBR would likely slow down because as the proportion of flows switch over to BBR, the advantage of doing so will start to drop [3]. Beyond a critical mass, CUBIC will end up outperforming BBR when it is in the minority. Our latest study suggests that Mishra et al. might be on to something since the

proportion of BBR sites has hardly changed since 2019, despite rapid initial adoption.

**Different Strokes for Different Applications, Different Localities**. One of the surprising findings of our latest measurement study is that the preferences for CCAs is not uniform, i.e. providers do not seem to have a preference of one CCA over the rest. In fact, we have found instances where a provider can deploy different CCAs under different contexts. For example, Apple deploys BBR for videos and CUBIC for ads in the same(!) session (see §6.3.5).

**Classifying CCAs beyond those in the Linux kernel**. As discussed, Nebby can be extended to more CCAs as they are deployed on the Internet. For example, Meta implements Copa [93] in their QUIC stack. As an extension to *Nebby*'s classifier, we wrote a simple Copa classifier. This classifier identified Copa based on its periodic oscillations around the bottleneck bandwidth that occur every five RTTs (see Figure 6.12). Our classifier was able to successfully classify both the original UDP implementation of Copa [93] as well as `mvfst`'s implementation [101] with an accuracy of 88%. Interestingly, `mvfst` Copa's oscillations were less visible at higher RTTs, even though in theory Gordon should be able to view a larger portion of Copa's `BiF`. When we ran this classifier for our Alexa Top 20k traces, none of the existing websites were identified to be running Copa, including Facebook domains. This is not surprising, since Copa is reported to be deployed by Meta only at the uplink [137].

We also wrote a classifier for PCC Vivace [37], which proved to be more challenging. Vivace is an online optimization algorithm that periodically probes above and below the receive rate to see if it can improve its utility. These probes are relatively small, but we can see in the 100 ms delay profile shown in Figure 6.12d that Gordon is able to observe the resulting oscillations. However, our classifier could only identify these *steps* in the `BiF` only about half the time, perhaps because the variations in the `BiF` are relatively small. As a result, our PCC Vivace classifier's accuracy suffered and was about 58%. We believe that with further study, it is certainly possible to develop a more accurate




**(a)** Copa [93] **(b)** `mvfst` Copa [101]

**(c)** PCC Vivace [37] **(d)** Zoomed in view for a PCC trace.

**Figure 6.12:** `BiF` traces for Copa and PCC Vivace.

classifier for PCC Vivace and we leave this as future work.

**QUIC CCA variants**. It has been shown that while QUIC stacks implement standard CCA variants, many of these variants behave somewhat differently from standard kernel implementations [33]. The results of our current study seem to suggest that these non-conformant variants can be classified successfully even though they do not behave exactly like standard kernel implementations. This trend might not continue to hold in the future and the characterization of the current congestion control landscape will be increasingly complex.

In this measurement study, we have focused solely on identifying the congestion control variant used and we used our classifier that was trained on the kernel implementations. It is plausible for us to improve the accuracy of our classifier by using the traces

from the QUIC implementations as well and for us to do a more detailed measurement study to identify not only the CCA but the actual QUIC stack for existing QUIC servers on the Internet. This remains as future work.

## 6.5 Summary

In the early 2000s, measurement studies on the distribution of congestion control algorithms were generally conducted roughly every 10 years. This was fine because the evolution of congestion control algorithms was relatively slow. Today, we are in an era of rapid change where new CCAs are developed every year. With Gordon, researchers finally have a reliable and extensible way to keep abreast of the evolution of the Internet congestion control landscape.

## 6.6 Resources

Our measurement tool Nebby, along with the `BiF` traces for the Alexa Top 20,000 websites will be made available on GitHub (`https://github.com/NUS-SNL/Nebby`).

Chapter 7

# Conclusion

In this thesis, we present a 5-year view of the Internet's congestion control landscape through measurement and mathematical modeling. We took two snapshots of the mix of congestion control algorithms on the Internet through Internet-scale measurement studies conducted in 2019 (see §3) and late 2023 (see §6).

Inspired by BBR's rapid adoption on the Internet, we also proposed a mathematical model and presented a game-theoretic analysis of competing CUBIC and BBR flows (see §4). Based on these results, we made a bold prediction that BBR's performance gains over CUBIC, and therefore its rate of adoption on the Internet, is likely to slow down as the proportion of BBR flows increases on the Internet.

We also showed that speciation in QUIC congestion control risks further increasing the heterogeneity in Internet congestion control if left unchecked (see §5). Our tool QUICbench is able to detect QUIC CCA implementations that are not conformant to their reference kernel implementations and even suggest fixes.

**Table 7.1:** Evolution of the Internet's Congestion Control Landscape (2001–present).

| Class | CCA | 2001 [19] | 2004 [138] | 2011 [21] | 2019, Gordon [1] | 2023, Nebby [6] |
|---|---|---|---|---|---|---|
| Loss-based AIMD | New Reno [14] | 35% (1,571) | 25% (21,266) | | 0.8% (160) | 11.1% (11,157) |
| | Reno [7] | 21% (945) | 5% (4,115) | 12.5% (623) | - | - |
| | Tahoe | 26% (1,211) | 3% (2,164) | | - | - |
| Loss-based MIMD | CUBIC [16] | - | - | 22.3% (1,115) | 30.7% (6,139) | 41% (41,085) |
| | BIC [15] | | | 10.6% (531) | 0.9% (181) | 3% (2,985) |
| | HSTCP [82] | | | 7.4% (369) | [R] | 0% (0) |
| | Scalable [76] | | | 1.4% (69) | 0.2% (39) | 0% (24) |
| Delay-based AIMD | Vegas [43] | - | - | 1.2% (58) | 2.8% (564) | 2.6% (2,637) |
| | Westwood [77] | | | 2% (104) | 0% (0) | 0.4% (371) |
| Delay-based MIMD | CTCP [17] | - | - | 6.7% (334) | 5.7% (1,148) | [C] |
| | Illinois [139] | | | 0.6% (28) | | 3.4% (3,380) |
| | Veno [44] | | | 0.9% (45) | [V] | 0.6% (578) |
| | YeAH [42] | | | 1.4% (72) | 5.8% (1,162) | 0.4% (388) |
| | HTCP [75] | | | 0.4% (18) | 2.8% (560) | 2.3% (2,259) |
| Rate-based | BBRv1 [78] | - | - | - | 17.8% (3,550) | 10% (9,985) |
| | BBR G1.1 [1] | | | | 0.8% (167) | - |
| | BBRv2 [26] | | | | - | 1.1% (1,151) |
| | BBRv3 | | | | - | 0.2% (204) |
| Unclassified | | 17.3% (792) | 53% (44,950) | 4% (198) | 12.2% (2,432) | 16.7% (16,733) |
| AkamaiCC | | - | - | - | 5.5% (1,103) | 7.2% (7,117) |
| Short Flows | | - | - | 26% (1,300) | 7.5% (1,493) | - |
| Unresponsive | | 0.7% (30) | 14% (11,529) | - | 6.5% (1,302) | - |
| Abnormal SS* | | - | - | 2.9% (144) | - | - |
| Total hosts | | 100% (4,550) | 100% (84,394) | 100% (5,000) | 100% (10,000) | 100% (100,000) |

[R] Classified together with New Reno
[V] Classified together with Vegas
[C] CTCP has been deprecated in Windows
* Websites identified by CAAI as having Abnormal Slow Starts

## 7.1 Summary of Internet CCA Evolution (2001–present)

One of the key contributions of this thesis is revealing the massive increase in CCA heterogeneity today compared to the pre-BBR Internet. We summarize the results from Chapters 3 and 6 and tabulate them along with the results from previous similar measurement studies in Table 7.1. We note that since Nebby uncovered different CCA deployment rates in different regions, we have added the results from all the regions to make the proportions more comparable to previous studies which are not region-specific.

When viewed in context, a number of trends become clear:

1. First, BBR, along with its variants, is currently a significant player on the Internet. While BBR's adoption has seen a dip in terms of pure website count, most major players who were early adopters of BBRv1 have decided to stick with deploying BBR and its variants. Between our 2019 and 2023 measurement studies, we have

also seen the set of rate-based CCAs expand.

2. On the other hand, recent trends indicate diversity amongst the set of AIMD/MIMD CCAs on the Internet is reducing, with most websites still using an AIMD/MIMD CCA choosing to deploy either CUBIC or New Reno.

3. Finally, undocumented variants like *AkamaiCC* are also slowly gaining traction. While AkamaiCC was limited to Akamai-hosted websites in 2019, this is no longer true today. This indicates that there is a need to keep an eye on these undocumented variants on the Internet and track their adoption.

Overall, while infrequent snapshots of the Internet's congestion control landscape were sufficient in the past, we argue that there is a need to review the mix of CCAs deployed by popular websites on the Internet more frequently today. We expect future-proof and extensible CCA classification tools like Nebby to lead the way on this front.

## 7.2 The cause and effect of CCA heterogeneity

While the rapid increase in heterogeneity in the Internet's congestion control landscape that we have observed might seem alarming at first, it is important to remember that this heterogeneity has not been created in a vacuum. Over the years, CCA design has been inspired by the evolving demands of nascent web applications. For example, historically when the Internet was not very well provisioned, throughput was the main driver for performance. Classic AIMD/MIMD algorithms like CUBIC and New Reno fit this paradigm well, especially in the presence of deep enough buffers. As bandwidths increased, webpage responsiveness started being driven by latency, inspiring efforts for reducing end-to-end latency using early delay-based CCAs like Vegas. There was a need for low latency and consistently high throughput as video traffic share increased on the Internet. BBR was first introduced to meet these demands. It is therefore no surprise

that when Google first deployed BBR in 2016, one of the first services they tested it on was YouTube. As discussed in §6, BBR remains a popular choice for websites delivering video content. On the other hand, CUBIC remains popular for delivering static assets like banner ads on most websites.

The lesson here is simple. Since the makeup of the Internet's congestion control landscape is application-driven, so should our ideas of stability, fairness, deployability, and provisioning.

**An Evolving definition of *Fairness* on the Internet.** Traditionally, fairness on the Internet has been thought of in terms of bandwidth fairness. In the early days of the Internet, this was true, since throughput was the main driver for performance for all flows. However, as the Internet, the applications that run on it, and the CCAs that support these applications all mature, our idea of fairness needs to evolve as well. Since the network supports different flows that have different performance requirements, fairness between them should not be judged based on just throughput. In the past, there have been proposals for utility models [140] that allow flows that have different utility functions to be compared to one another. BBRv1 was often critiqued for being 'irresponsible' for not being bandwidth fair to traditional CCAs like CUBIC [25]. These ideas of deployability must evolve as well [32].

**Taming the Zoo.** Given that the majority of Nash Equilibrium distributions that we have found are *mixed* distributions of CUBIC and BBR, it is likely that these two algorithms will have to co-exist on the Internet for the foreseeable future. We therefore need to work on networking solutions that work well with not just one class of congestion control algorithms, but a diverse mix of both of them and probably other variants. In this aspect, solving how well traffic from different classes of applications and the CCAs they chose to deploy can coexist is the next big question in Internet congestion control.

The main challenge in allowing these congestion control algorithms to coexist lies in finding a way to allow them to achieve their different performance goals while sharing

the same bottleneck. For example, when a delay-sensitive and a throughput-sensitive flow both share a bottleneck with a FIFO queue, it is impossible for the delay-sensitive flow to reduce queuing delay in the presence of the competing throughput-sensitive flow that will keep the bottleneck buffer filled. Queuing disciplines for modern Internet traffic need to be cognizant of this challenge and provide ways to isolate different classes of flows that value different network metrics.

**Incentives to switch to better congestion control**. BBR's deployment trajectory from first being introduced in 2016 to now in 2023 also raises an important question on the incentives to switch to objectively *better* CCAs on the Internet. For example, it is well-known that when a bottleneck has only BBR flows, they are able to keep queue occupancy low while ensuring utilization. One can therefore argue that BBR is a better congestion control algorithm for the Internet than CUBIC, which tends to fill bottleneck buffers and keeps the queuing delay high.

In the past, transitions to such modern and *better* congestion control algorithms have been possible on the basis of performance incentives alone. For example, CUBIC could consistently outperform New Reno in terms of throughput, and therefore quickly became a dominant force on the Internet's congestion control landscape [21]. Not only can BBR not enjoy the same guarantee of superior performance (see §4), but it is also subject to a lot more stringent deployability checks. One can imagine that if CUBIC was proposed today in an all-Reno Internet, it would also be considered *unfair* and too aggressive.

In this sense, we must balance the need for a fair Internet against the need for a better and more performant Internet. Our threshold for deployability must be reviewed more carefully so that it allows innovation and improvement in the CCA design space while also ensuring that we do not inflict too much harm on flows that choose to stick to legacy congestion control algorithms.

## 7.3 Future Work

We expect the work in this thesis to inspire future research in Internet congestion control. We briefly discuss some possible research questions below.

**Re-answering questions on buffer sizing** We need to review how we size buffers on the Internet in two respects. The first is understanding how current buffer sizing rules of thumbs [40] are fair in the face of a realistic heterogeneous mix of CCAs and not just loss-based CCAs. The second is viewing sizing buffers as a way to bounding unfairness between traditional loss-based CCAs and modern rate-based CCAs like BBR. Traditionally, the function of the buffer has only been to ensure utilization. While modeling buffers for the modern Internet, we not only have to consider link utilization but also what impact those buffer sizes have on the fairness between different competing CCAs.

**In-network support for congestion control algorithms.** The zoo that the Internet has become today is also an indication that we are perhaps are at the end of the road of what we can achieve with end-point-only congestion control algorithms. In the future, in order to keep the Internet running smoothly and simultaneously support the demands of different classes of applications, we will need in-network support in Internet congestion control.

**Continuous and longitudinal measurement studies.** All signs point to heterogeneity on the Internet continuing to increase in the future, especially with the continued revisions to BBR and its family of variants, as well as the adoption of QUIC. In this aspect, we strongly feel the need to keep an eye on the Internet's congestion control landscape and conduct more frequent measurement studies.

**Dealing with bad actors.** Our experience with QUIC also leaves us with no doubt that it will encourage people to continue to test and deploy new and modified CCAs on the Internet. Aside from the results presented in this thesis, many developers working

on proxy tools are using QUIC as a way to implement extremely aggressive congestion control algorithms as a way to bypass circumvention measures. Brutal [141], for example, is a CCA implemented in Hysteria's QUIC stack that *speeds up* its sending rate when it sees packet loss! mKCP [142], implemented along with the v2ray reverse-proxy also gives the sender an option to turn off congestion control and send data at a fixed high rate. As the population of such bad actors increases on the Internet, researchers will have to come up with creative ways to limit the damage such aggressive CCAs can do to other flows on the Internet.

# Bibliography


[1] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. The Great Internet TCP Congestion Control Census. In *Proceedings of SIGMETRICS*, 2019.

[2] Ayush Mishra, Jingzhi Zhang, Melodies Sims, Sean Ng, Raj Joshi, and Ben Leong. Conjecture: Existence of nash equilibria in modern internet congestion control. In *Proceedings of APNet*, 2021.

[3] Ayush Mishra, Wee Han Tiu, and Ben Leong. Are we heading towards a BBR-dominant internet? In *Proceedings of IMC*, 2022.

[4] Ayush Mishra, Sherman Lim, and Ben Leong. Understanding speciation in QUIC congestion control. In *Proceedings of IMC*, 2022.

[5] Ayush Mishra and Ben Leong. Containing the cambrian explosion in QUIC congestion control. In *Proceedings of IMC*, 2023.

[6] Ayush Mishra, Lakshay Rastogi, Raj Joshi, and Ben Leong. Keeping an eye on congestion control in the wild with nebby. In *Proceedings of SIGCOMM*, 2024.

[7] Van Jacobson. Congestion avoidance and control. *SIGCOMM CCR*, 18(4):314–329, 1988.

[8] Leonard Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *ICC 1979; International Conference on Communications, Volume 3*, 1979.

[9] L. Xu, s. Ha, Vidhi Goel, and Lars Eggert. Spurious Congestion Events, 2023. https://tools.ietf.org/id/draft-ietf-tcpm-rfc8312bis-00.html.

[10] Canada Sandvine Inc. Waterloo, ON. The 2018 global internet phenomena report, 2018. https://www.sandvine.com/phenomena.

[11] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi,

and Ben Leong. Gordon: Congestion control identification tool, 2019. https://github.com/NUS-SNL/Gordon.

[12] Internet World Stats. Internet usage statistics, 2023. https://www.internetworldstats.com/stats.htm.

[13] I Stoica. A comparative analysis of TCP tahoe, reno, new-reno, sack and vegas. *Communication Networks, Student Project*, 2005.

[14] Vern Paxson and Mark Allman. TCP Congestion Control. RFC 5681, 2009.

[15] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *Proceedings of INFOCOM*, 2004.

[16] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[17] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. A compound TCP approach for high-speed and long distance networks. In *Proceedings of INFOCOM*, 2006.

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

[19] Jitendra Padhye and Sally Floyd. On inferring TCP behavior. In *Proceedings of SIGCOMM*, 2001.

[20] Alberto Medina, Mark Allman, and Sally Floyd. Measuring the evolution of transport protocols in the internet. *SIGCOMM CCR*, 35(2):37–52, 2005.

[21] Peng Yang, Juan Shao, Wen Luo, Lisong Xu, Jitendra Deogun, and Ying Lu. Tcp congestion avoidance algorithm identification. *IEEE/ACM Transactions on Networking*, 22(4):1311–1324, 2011.

[22] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based Congestion Control. *CACM*, 60(2):58–66, 2017.

[23] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport, RFC 9000, 2021. https://datatracker.ietf.org/doc/html/rfc9000.

[24] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of BBR congestion control. In *Proceedings of ICNP*, 2017.

[25] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. Modeling BBR's interactions with loss-based congestion control. In *Proceedings of IMC*, 2019.

[26] 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. ICCRG at IETF 104, 2019. https://bit.ly/2HgGOuQ.

[27] BBRv3: Algorithm Bug Fixes and Public Internet Deployment, 2023. http://tinyurl.com/bbrv3ietf.

[28] Active QUIC implementations, 2021. https://github.com/quicwg/base-drafts/wiki/Implementations.

[29] Ed. M. Bishop. RFC 9114 HTTP/3, 2022. https://www.rfc-editor.org/rfc/rfc9114.html.

[30] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. *ACM SIGCOMM CCR*, 34(4), 2004.

[31] Bruce Spang, Serhat Arslan, and Nick McKeown. Updating the theory of buffer sizing. In *Proceedings of SIGMETRICS*, 2022.

[32] 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 Hotnets*, pages 17–24, 2019.

[33] Ayush Mishra and Sherman Lim. QUICBench, 2022. https://github.com/NUS-SNL/QUICbench.

[34] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. BBR: Congestion-based Congestion Control. *CACM*, 60(2):58–66, 2017.

[35] Alexey Ivanov. Evaluating bbrv2 on the dropbox edge network, 2019. https://tinyurl.com/yyrs68pp.

[36] Erik Carlsson and Eirini Kakogianni. Smoother streaming with bbr, 2018. https://tinyurl.com/yyt5tbhd.

[37] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC Vivace: Online-Learning Congestion Control. In *Proceedings of NSDI*, 2018.

[38] Ed. M. Bishop. Hypertext Transfer Protocol Version 3 (HTTP/3), 2021. https://tools.ietf.org/html/draft-ietf-quic-http-34.

[39] Jitendra Pahdye and Sally Floyd. On inferring TCP behavior. In *Proceedings of SIGCOMM*, 2001.

[40] Nick McKeown, Guido Appenzeller, and Isaac Keslassy. Sizing router buffers (redux). *SIGCOMM CCR*, 49(5):69–74, 2019.

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

[42] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. Yeah-tcp: yet another highspeed tcp. In *Proceedings of PFLDnet*, 2007.

[43] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of SIGCOMM*, 1994.

[44] Cheng Peng Fu and S. C. Liew. TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks. *IEEE JSAC*, 21(2):216–228, 2006.

[45] Shao Liu, Tamer Başar, and R. Srikant. Tcp-illinois: A loss and delay-based congestion control algorithm for high-speed networks. In *Proceedings of VALUETOOLS*, 2006.

[46] Peng Yang and Lisong Xu. A survey of deployment information of delay-based TCP congestion avoidance algorithm for transmitting multimedia data. In *Proceedings of GLOBECOM Workshops*, 2011.

[47] Sishuai Gong, Usama Naseer, and Theophilus A Benson. Inspector gadget: A framework for inferring TCP congestion control algorithms and protocol configurations. In *Network Traffic Measurement and Analysis Conference*, 2020.

[48] Xiaoyu Chen, Shugong Xu, Xudong Chen, Shan Cao, Shunqing Zhang, and Yanzan Sun. Passive TCP Identification for Wired and Wireless Networks: A Long-Short Term Memory Approach. *arXiv preprint:1904.04430*, 2019.

[49] Desta H. Hagos, Paal E. Engelstad, Anis Yazidi, and Øivind Kure. General TCP state inference model from passive measurements using machine learning techniques. *IEEE Access*, 6:28372–28387, 2018.

[50] Douglas E. Comer and John C. Lin. Probing TCP implementations. In *Proceedings of USTC*, 1994.

[51] W. Sun, L. Xu, and S. Elbaum. Scalably testing congestion control algorithms of real-world TCP implementations. In *Proceedings of ICC*, 2018.

[52] Ralf Lübben and Markus Fidler. On characteristic features of the application level delay distribution of TCP congestion avoidance. In *Proceedings of ICC*, 2016.

[53] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle. Towards a deeper understanding of TCP BBR congestion control. In *Proceedings of IFIP Networking*, 2018.

[54] Belma Turkovic, Fernando A. Kuipers, and Steve Uhlig. Fifty shades of con-

gestion control: A performance and interactions evaluation. *arXiv preprint arXiv:1903.03852*, 2019.

[55] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. PCC vivace: Online-learning congestion control. In *Proceedings of NSDI*, 2018.

[56] Simon Scherrer, Markus Legner, Adrian Perrig, and Stefan Schmid. Model-based insights on the performance, fairness, and stability of bbr. In *Proceedings of IMC*, 2022.

[57] Aditya Akella, Srinivasan Seshan, Richard Karp, Scott Shenker, and Christos Papadimitriou. Selfish behavior and stability of the internet: A game-theoretic analysis of tcp. *ACM SIGCOMM CCR*, 32(4), 2002.

[58] Steve Chien and Alistair Sinclair. Convergence to approximate nash equilibria in congestion games. In *Proceedings of SODA*, 2007.

[59] Tuan Anh Trinh and Sándor Molnár. A game-theoretic analysis of TCP vegas. In *Quality of Service in the Emerging Networking Panorama*, 2004.

[60] 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 SIGCOMM*, 2017.

[61] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. HTTP over UDP: an Experimental Investigation of QUIC. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC)*, pages 609–614, 2015.

[62] Prasenjeet Biswal and Omprakash Gnawali. Does QUIC Make the Web Faster? In *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2016.

[63] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. How quick is QUIC? In *Proceedings of International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[64] 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 Internet Measurement Conference (IMC)*, pages 290–303, 2017.

[65] Mirko Palmer, Thorben Krüger, Balakrishnan Chandrasekaran, and Anja Feldmann. The QUIC Fix for Optimal Video Streaming. In *Proceedings of the Work-*

*shop on the Evolution, Performance, and Interoperability of QUIC*, pages 43–49, 2018.

[66] James Pavur, Martin Strohmeier, Vincent Lenders, and Ivan Martinovic. QPEP: A QUIC-Based Approach to Encrypted Performance Enhancing Proxies for High-Latency Satellite Broadband. In *Proceedings of NDSS*, 2021.

[67] Darius Saif, Chung-Horng Lung, and Ashraf Matrawy. An Early Benchmark of Quality of Experience Between HTTP/2 and HTTP/3 using Lighthouse. In *IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2021.

[68] Google Chrome, Lighthouse, 2022. https://github.com/GoogleChrome/lighthouse.

[69] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, pages 14–20, 2020.

[70] Neal Cardwell. tcp_BBR: add BBR congestion control, 2017. https://bit.ly/2VAJcDD.

[71] The top 500 websites on the internet, 2018. https://www.alexa.com/topsites.

[72] DARPA. Internet protocol. RFC 791, 1981.

[73] Yuchung Cheng Jerry Chu, Nandita Dukkipati and Matt Mathis. Increasing TCP's Initial Window. RFC 6928, 2013.

[74] Jan Rüth, Christian Bormann, and Oliver Hohlfeld. Large-scale scanning of TCP's initial window. In *Proceedings of IMC*, 2017.

[75] Douglas Leith, R Shorten, and Y Lee. H-tcp: A framework for congestion control in high-speed and long-distance networks. In *Proceedings of PFLDnet*, 2005.

[76] Tom Kelly. Scalable tcp: Improving performance in highspeed wide area networks. *SIGCOMM CCR*, 33(2):83–91, 2003.

[77] Claudio Casetti, Mario Gerla, Saverio Mascolo, Medy Y Sanadidi, and Ren Wang. Tcp westwood: end-to-end congestion control for wired/wireless networks. *Wireless Networks*, 8(5):467–479, 2002.

[78] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, and Van Jacobson. BBR Congestion Control. IETF Draft, 2017. https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-00.

[79] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *Proceedings of ATC*, 2015.

[80] GNU Project. wget, 2019. https://www.gnu.org/software/wget.

[81] Netfilter Organization. libnetfilter_queue, 2019. https://bit.ly/2HimY17.

[82] Sally Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649, 2003.

[83] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. RACK: a time-based fast loss detection algorithm for TCP. IETF Draft, 2019. https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-rack-05.

[84] TCP Optimizations - Akamai Developer, 2019. https://bit.ly/35LYJUx.

[85] David X Wei, Cheng Jin, Steven H Low, and Sanjay Hegde. Fast tcp. In *In Proceedings of IEEE/ACM Transactions on Networking*, 2007.

[86] Brien Posey. Explore the cubic congestion control provider for windows, 2019. https://bit.ly/2VfhxoA.

[87] TCP BBR congestion control comes to GCP: your Internet just got faster, 2017. https://bit.ly/2Hk4WLH.

[88] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. Modeling bbr's interactions with loss-based congestion control. In *Proceedings of IMC*, 2019.

[89] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proceedings of SIGCOMM*, 2004.

[90] Mihaela Enachescu, Yashar Ganjali, Ashish Goel, Nick McKeown, and Tim Roughgarden. Routers with very small buffers. In *Proceedings of INFOCOM*, 2006.

[91] Bruce Spang, Serhat Arslan, and Nick McKeown. Updating the theory of buffer sizing. *Performance Evaluation*, 151, 2021.

[92] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson, and Amin Vahdat. Tcp BBR congestion control comes to gcp – your internet just got faster, 2017. https://tinyurl.com/yc7bd9jk.

[93] Venkat Arun and Hari Balakrishnan. Copa: Practical Delay-Based Congestion Control for the Internet. In *Proceedings of NSDI*, 2018.

[94] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. Continuous in-network round-trip time monitoring. In *Proceedings of SIGCOMM*, 2022.

[95] Tomoki Kozu, Yuria Akiyama, and Saneyasu Yamaguchi. Improving rtt fairness on cubic tcp. In *2013 First International Symposium on Computing and Networking*, pages 162–167. IEEE, 2013.

[96] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of BBR congestion control. In *Proceedings of ICNP*, 2017.

[97] Jim Gettys. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 15(3):96–96, 2011.

[98] Canada Sandvine Inc. Waterloo, ON. The 2022 global internet phenomena report, 2022. https://www.sandvine.com/phenomena.

[99] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *Operating Systems Review*, 42:64–74, 07 2008.

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

[101] Facebook's QUIC implementation, mvfst, 2022. https://github.com/facebookincubator/mvfst.

[102] Google's QUIC implementation, chromium, 2022. https://www.chromium.org/quic/playing-with-quic.

[103] Microsoft's QUIC implementation, msquic, 2022. https://github.com/microsoft/msquic.

[104] Cloudflare's QUIC implementation, quiche, 2022. https://github.com/cloudflare/quiche.

[105] LiteSpeed's QUIC implementation, lsquic, 2022. https://github.com/litespeedtech/lsquic.

[106] Go's QUIC implementation, quic-go, 2022. https://github.com/lucas-clemente/quic-go.

[107] H2O's QUIC implementation, quicly, 2022. https://github.com/h2o/quicly.

[108] Rust's QUIC implementation, quinn, 2022. https://github.com/quinn-rs/quinn.

[109] Amazon Web Services's QUIC implementation, s2n-quic, 2022. https://github.com/aws/s2n-quic.

[110] Alibaba's QUIC implementation, xquic, 2022. https://github.com/alibaba/xquic.

[111] Mozilla's QUIC implementation, neqo, 2022. https://github.com/mozilla/neqo.

[112] P. Balasubramaniam, Y. Huang, and M. Olson. HyStart++: Modified Slow Start for TCP, 2023. https://www.rfc-editor.org/rfc/rfc9406.

[113] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1), 1979.

[114] iPerf, the Speed Test Tool for TCP, 2021. https://iperf.fr/iperf-doc.php.

[115] Akamai quic, 2023. https://akaquic.com/.

[116] Tommy Pauly. Quic usage at apple, 2021. https://tinyurl.com/applequic.

[117] Ietf QUIC implementation in haskell, 2023. https://github.com/kazu-yamamoto/quic.

[118] askF5. Overview of the big-ip http/3 and QUIC profiles, 2022. https://support.f5.com/csp/article/K60235402.

[119] Ietf QUIC implementation in java, 2023. https://bitbucket.org/pjtr/kwik/src/master/.

[120] ngtcp2, 2023. https://github.com/ngtcp2/ngtcp2.

[121] nginx-quic, 2023. https://hg.nginx.org/nginx-quic/.

[122] picoquic, 2023. https://github.com/private-octopus/picoquic.

[123] aioquic: QUIC network protocol in python, 2023. https://github.com/aiortc/aioquic.

[124] quant, 2023. https://github.com/NTAP/quant.

[125] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for HTTP. In *Proceedings of USENIX ATC*, 2015.

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

[127] Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. CUBIC for Fast Long-Distance Networks. RFC 8312, 2018. https://tools.ietf.org/html/rfc8312.

[128] ExpiredDomains.net. Alexa Top Websites - Last save, 2023. https://www.expireddomains.net/alexa-top-websites/.

[129] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. Continuous in-network round-trip time monitoring. In *Proceedings of the SIGCOMM*, 2022.

[130] Selenium, 2024. https://www.selenium.dev.

[131] Nebby, 2024. www.github.com/NUS-SNL/Nebby.

[132] The top 500 websites on the internet, 2023. https://www.alexa.com/topsites.

[133] Google. Personal correspondance, 2023.

[134] Matt Joras and Yang Chi. How Facebook is bringing QUIC to billions, 2020. https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/.

[135] NetFlix. Personal correspondance, 2023.

[136] Eirini Kakogianni Erik Carlsson. Smoother streaming with bbr, 2018. https://engineering.atspotify.com/2018/08/smoother-streaming-with-bbr/.

[137] Nitin Garg. Engineering at Meta: Evaluating COPA congestion control for improved video performance, 2019. https://engineering.fb.com/2019/11/17/video-engineering/copa/.

[138] Sharad Jaiswal, Gianluca Iannaccone, Christophe Diot, Jim Kurose, and Don Towsley. Inferring TCP connection characteristics through passive measurements. In *Proceedings of INFOCOM*, 2004.

[139] Shao Liu, Tamer Başar, and R. Srikant. TCP-Illinois: A Loss and Delay-based Congestion Control Algorithm for High-speed Networks. In *Proceedings of VALUETOOLS*, 2006.

[140] Daniel Pérez Palomar and Mung Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8), 2006.

[141] Tcp-brutal: Congestion control algorithm that increases speed on packet loss, 2023. https://news.ycombinator.com/item?id=38164574.

[142] mKCP Transport, 2023. https://www.v2ray.com/en/configuration/transport/mkcp.html.