UNIVERSITÄT DES SAARLANDES

Towards a Robust and Reproducible Evaluation Framework for Congestion Control Algorithms

Dissertation zur Erlangerung des Grades des
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
der Fakultät für Mathematik und Informatik der
Universität des Saarlandes

vorgelegt von

Emilia Ndilokelwa Weyulu

Saarbrücken, 2025

Day of Colloquium	04 December 2025
Dean of the Faculty	Univ.-Prof. Dr. Roland Speicher
Chair of the Committee	Prof. Dr.-Ing. Thorsten Herfet
Reporters	Prof. Dr. Anja Feldmann
	Dr. Balakrishnan Chandrasekaran
	Dr. Antoine Kaufmann
Academic Assistant	Dr. Johannes Zirngibl

Abstract

Network congestion, the state where systems such as switches or routers receive more data than they can handle, leads to packet losses, increased network delay, and reduced throughput for all data passing through such a congested system. Congestion control remains a key research problem in networking, with both industry and academia proposing solutions to improve network performance. Research on congestion-control algorithms (CCAs) spans about four decades, yet there is still no "one-size fits-all" solution. Due to evolving hardware (programmable switches and NICs), changing traffic patterns, stringent application requirements, etc., new CCAs are being proposed every so often.

CCAs for the public Internet or WANs are usually designed by researchers in academia or industry, or by operators of large-scale networked systems. These algorithms must then run in networks where they share network resources with other (existing) CCAs. So, before deployment, designers test if a proposed CCA performs well in its intended environment, without negatively impacting cross-traffic. These evaluations are typically conducted using simulations, modeling, or emulations on lab-based testbeds, or a small or limited set of real-world network paths. Performance insights from such evaluation approaches, however, cannot guarantee that a CCA performs well when it is eventually deployed on complex, real-world network paths. In addition, the evaluation of CCAs is a time-consuming process, compounded by the difficulty of selecting representative network parameters. Furthermore, it is impractical for designers to exhaustively test all possible scenarios in which a CCA might eventually be used.

Despite the innovation and research effort that has been invested into designing new CCAs, there is still no consensus within the networking community on how to benchmark their performance. Determining how any CCA falls short compared to the rest, and, most importantly, along what dimensions, remains difficult to answer. Even performing all the pairwise comparisons between the algorithms is hard, because each algorithm behaves differently depending on the underlying network environment.

In this thesis, we advocate for a fundamental rethinking of how we approach CCA evaluations. Rather than prescribing a standardized set of tests to be universally applied—an approach that has historically failed to achieve consensus, we emphasize the importance of aligning evaluations with their underlying objectives. By shifting the focus in this way, the burden on designers to subject their CCAs to an exhaustive list of experiments can be avoided, while simultaneously addressing the reproducibility challenges that currently plague this field.

To this end, we developed a rigorous and reproducible "recipe" for evaluating CCAs. With this recipe, we were able to uncover fundamental issues in the design of Google's new CCA, BBRv3–work which was recognized with a "Best Paper" award at PAM'24. Furthermore, this research work has helped to highlight the critical network signals one needs to leverage in the design of network-assisted CCAs.

Zusammenfassung

Übersetzt mit DeepL.com (kostenlose Version)

Netzüberlastung, d. h. der Zustand, in dem Systeme wie Switches oder Router mehr Daten empfangen, als sie verarbeiten können, führt zu Paketverlusten, erhöhter Netzverzögerung und verringertem Durchsatz für alle Daten, die ein solches überlastetes System durchlaufen. Die Staukontrolle ist nach wie vor ein zentrales Forschungsproblem im Netzbereich, wobei sowohl die Industrie als auch die Wissenschaft Lösungen zur Verbesserung der Netzleistung vorschlagen. Die Forschung zu Algorithmen zur Staukontrolle (CCAs) erstreckt sich über etwa vier Jahrzehnte, doch gibt es noch immer keine "Einheitslösung für alle". Aufgrund der sich weiterentwickelnden Hardware (programmierbare Switches und NICs), sich ändernder Verkehrsmuster, strenger Anwendungsanforderungen usw. werden immer wieder neue CCAs vorgeschlagen.

CCAs für das öffentliche Internet oder WANs werden in der Regel von Forschern in Hochschulen oder der Industrie oder von Betreibern großer vernetzter Systeme entwickelt. Diese Algorithmen müssen dann in Netzen laufen, in denen sie sich die Netzressourcen mit anderen (bestehenden) CCAs teilen. Daher testen die Entwickler vor dem Einsatz, ob eine vorgeschlagene CCA in der vorgesehenen Umgebung gut funktioniert, ohne den Querverkehr negativ zu beeinflussen. Diese Bewertungen werden in der Regel mit Hilfe von Simulationen, Modellen oder Emulationen auf laborgestützten Testumgebungen oder einer kleinen oder begrenzten Anzahl von realen Netzwerkpfaden durchgeführt. Die aus solchen Evaluierungsansätzen gewonnenen Leistungserkenntnisse sind jedoch keine Garantie dafür, dass eine CCA auch dann gut funktioniert, wenn sie schließlich auf komplexen, realen Netzpfaden eingesetzt wird. Darüber hinaus ist die Bewertung von CCAs ein zeitaufwändiger Prozess, der durch die Schwierigkeit, repräsentative Netzparameter auszuwählen, noch erschwert wird. Darüber hinaus ist es für die Entwickler unpraktisch, alle möglichen Szenarien, in denen eine CCA letztendlich eingesetzt werden könnte, umfassend zu testen.

Trotz des Innovations- und Forschungsaufwands, der in die Entwicklung neuer CCAs investiert wurde, gibt es in der Netzgemeinde noch immer keinen Konsens darüber, wie ihre Leistung zu bewerten ist. Es ist nach wie vor schwierig, festzustellen, inwieweit eine CCA im Vergleich zu den anderen abfällt, und vor allem, in welchen Bereichen. Selbst die Durchführung aller paarweisen Vergleiche zwischen den Algorithmen ist schwierig, da sich jeder Algorithmus je nach der zugrunde liegenden Netzwerkumgebung anders verhält.

In dieser Arbeit plädieren wir für ein grundsätzliches Überdenken der Art und Weise, wie wir an CCA-Bewertungen herangehen. Anstatt eine standardisierte Reihe von Tests vorzuschreiben, die universell angewendet werden sollen - ein Ansatz, der in der Vergangenheit keinen Konsens erzielt hat -, betonen wir, wie wichtig es ist, die Bewertungen an den zugrunde liegenden Zielen auszurichten. Durch diese Verlagerung des Schwerpunkts kann vermieden werden, dass die Konstrukteure ihre CCAs einer erschöpfenden Liste von Experimenten unterziehen müssen, während gleichzeitig die Probleme der Reproduzierbarkeit angegangen werden, die diesen Bereich derzeit plagen.

Zu diesem Zweck haben wir ein rigoroses und reproduzierbares “Rezept” für die Bewertung von CCAs entwickelt. Mit diesem Rezept waren wir in der Lage, grundlegende Probleme im Design von Googles neuer CCA, BBRv3, aufzudecken - eine Arbeit, die auf der PAM’24 mit einem “Best PaperAward ausgezeichnet wurde. Darüber hinaus hat diese Forschungsarbeit dazu beigetragen, die kritischen Netzwerksignale hervorzuheben, die bei der Entwicklung von netzwerkgestützten CCAs genutzt werden müssen.

Acknowledgements

Completing a PhD has certainly been the *hardest* thing I have ever done. I am beyond grateful to the people who helped me get through it, this journey would not have been possible without you. First and foremost, I owe my deepest gratitude to my two advisors, Anja Feldmann and Balakrishnan Chandrasekaran. Your unwavering support, mentorship and guidance kept me moving forward, even when I was not sure I could. Thank you for giving me the freedom to grow and learn, and for never letting me give into the doubts. Though I still have a lot to learn, you have truly taught me to deeply understand and appreciate the research process.

To my two closest collaborators, Seifedine Fathalli and Danesh Zeynali, you two were my PhD battle buddies. I will never forget the long Skype calls, endless debugging sessions, and numerous brainstorming meetings. We were truly in the trenches together and I am very lucky to have gotten the chance to work with, and learn from you two. Many thanks to my “Bestie”, Fariba Osali, for being a dependable friend around the office and for the many laughs. Thank you too to Tiago Heinrich for the never-ending encouragement during my thesis writing process and for never allowing me to slack off.

I also want to extend a warm appreciation to the INET admin crew—Rainer, Iris and Joerg–thank you for always handling our numerous, often quirky requests with such patience. And to the whole INET team, it was truly an honor to share this journey with you all.

Last but not least, my greatest gratitude goes to my family and friends. To my mother, whose love, strength, and belief in me taught me that I could forge my own path, unconstrained by my gender or origin. To my father, the most selfless man I have ever known—I miss you every day. To my siblings, extended family, and friends, both in Germany and Namibia, thank you for standing by me through it all. I could not have done this without your love and encouragement.

Publications

Pre-published Papers

Parts of this thesis are based on the following peer-reviewed papers that were published in international conferences. All my collaborators are listed among the co-authors.

International Conferences

Danesh Zeynali, Emilia N. Weyulu, Seifeddine Fathalli, Balakrishnan Chandrasekaran, and Anja Feldmann. "Promises and Potential of BBRv3". In: *Passive and Active Measurement (PAM).* 2024

Workshops

Danesh Zeynali, Emilia N. Weyulu, Seifeddine Fathalli, Balakrishnan Chandrasekaran, and Anja Feldmann. "BBRv3 in the public Internet: a boon or a bane?" In: *Applied Networking Research Workshop (ANRW).* 2024

Anja Feldmann, Balakrishnan Chandrasekaran, Seifeddine Fathalli, and Emilia N. Weyulu. "P4-enabled Network-assisted Congestion Feedback: A Case for NACKs". In: *Stanford Workshop on Buffer Sizing (BS)* (2019)

Under Submission

Parts of this thesis are based on the following papers that are currently under submission. All my collaborators are listed among the co-authors.

International Conferences

Emilia N. Weyulu, Danesh Zeynali, Seifeddine Fathalli, Adrian Zapletal, Balakrishnan Chandrasekaran, Anja Feldmann, and Fernando Kuipers. *A Recipe for Benchmarking Congestion Control Algorithms: Towards unifying and standardizing CCA evaluations.* Under Submission. 2025

Journal

Emilia N. Weyulu, Seifeddine Fathalli, Danesh Zeynali, Adrian Zapletal, Balakrishnan Chandrasekaran, Anja Feldmann, and Fernando Kuipers. *A call for an open, reproducible, standardized approach for evaluating congestion control algorithms.* Under Submission. 2025

Seifeddine Fathalli, Emilia N. Weyulu, Danesh Zeynali, Balakrishnan Chandrasekaran, and Anja Feldmann. *Network-assisted Congestion Feedback.* Under Submission. 2025

Pre-published Papers: Not part of this thesis

I have authored the following peer-reviewed paper that has already been published but is not part of this thesis. All my collaborators are listed among the co-authors.

International Conferences

Emilia Ndilokelwa Weyulu and Dirk Trossen. "Exploring The Benefits of In-Band Service Routing". In: *IFIP Networking.* 2024

Contents

1 Introduction

The Internet is an essential part of our lives, used by an estimated 5.3 billion people who depend on it for everyday services such as banking, remote working, socialising, education, research, entertainment and many others [8]. It was initially developed as an experimental packet-switched network for the Advanced Research Projects Agency Network (ARPANET) in the late 1960s, and it has since evolved into the vast and widespread global network that sustains modern communication and information exchange. The Internet's importance was especially highlighted during the outbreak of the COVID-19 pandemic in 2020 which caused global lockdowns, forcing almost everyone in the world to work, study and have their social interactions online [9]. Many of those services have stayed online since then, making the Internet a backbone of everyday life.

Transport protocols are key for facilitating the various online services and interactions on the Internet; they enable the transmission of application messages over communication links from one computer to another. On top of these transport protocols, congestion control algorithms (CCAs) are designed to ensure fair resource allocation among users (or applications). As described by Peterson et al., resource allocation refers to "the process by which network elements try to meet the competing demands that applications have for network resources–primarily link bandwidth and buffer space in routers or switches" [10]. This allocation process involves routers and switches communicating network resource availability to end-hosts using 'signals' (e.g., delay and/or loss).

Network resources are inherently scarce primarily due to hardware limits, and the vast volume of data that is transmitted over today's networks. This challenge is further exacerbated by the continuously growing data volumes driven by the proliferation of Internet-connected devices, such as PCs, mobile devices, household devices or wearables, and industrial machines that collectively generate an ever-increasing amounts of data [11]. As a result of this network resource scarcity, data transmissions often exhibits unreliable performance, and end-hosts must adjust their sending behavior in response to congestion signals to avoid overwhelming the network [10, 12].

The end-hosts' response to congestion is highly dependent on the type and granularity of the signals received. Advancements in network hardware (e.g., programming switches and NICs), network infrastructure (e.g., LEO satellite networks) bring about new congestion signals, thereby necessitating revisiting, rethinking, and reevaluating CCA designs to address emerging demands effectively. Similarly, novel application domains (e.g., virtual and extended reality applications) with more stringent requirements and new traffic patterns that put even more pressure on limited network resources, serve as a perennial source of novel opportunities and challenges for designing CCAs. These developments provide a fertile ground for innovative designs, and as a result, the networking community, in both academia and industry, have continuously updated and developed new CCAs, leading to rich body of work, spanning decades [3, 13–18].

The resulting CCA schemes, unsurprisingly, cater to a wide range of network conditions, and differ substantially in their assumptions about operating environments, leading to substantial heterogeneity of CCAs used in the Internet [19–21]. While traditional CCAs were designed on top of the Transmission Control Protocol (TCP), emerging protocols such as Quick UDP Internet Connections (QUIC) are making it even easier to design, test and deploy CCAs. The quick development cycle is because QUIC operates in user-space, thereby enabling fast deployment unlike TCP, and enables content providers to design novel CCAs or customize existing ones to cater to the performance requirements of their complex web applications [22]. This will potentially lead to even more increased Internet CCA heterogeneity. This healthy diversity, however, has a key shortcoming: It makes comparisons between any CCA designs difficult.

The Internet being a shared network means that it operates using a set of agreed-upon standards, and CCAs are no exception. Therefore, before a CCA is publicly deployed on the Internet for wider use, the Internet Engineering Task Force (IETF), a standards organization for the Internet, *strongly advocates* that CCAs should be rigorously evaluated to prevent "harm" to existing network traffic and ensure continued Internet stability [23]. A large number of CCAs are, however, developed and tested within closed environments, which makes it difficult to generalize the performance of a particular CCA observed during development to other network scenarios. The lack of generalizability is further amplified for CCAs intended to operate over the public Internet, where network paths often exhibit a diverse range of complex behaviours [24].

Another dimension that presents significant challenges in reproducing evaluations is the private infrastructure used to both develop and test these CCAs, whose specifications are typically not *fully* disclosed. Consequently, the majority of reproduction trials yield outcomes that deviate from those reported in the original studies, and lead to CCAs exhibiting substantial performance differences when tested in different environments, often resulting in unfair interactions with existing CCAs [1, 25, 26]. Compounding the issue is the manner in which the various CCA designs are evaluated: The evaluation approaches significantly differ from one another. The measures (e.g., utilization, fairness, and convergence), metrics (e.g., Jain's fairness index, harm [27], slowdown [28], and flow-completion time), network conditions (i.e., flow startup times and round-trip times (RTTs), etc.), and deployment scenarios (i.e., flow sizes, traffic workloads, bottleneck topology, etc.) used vary vastly across studies.

The networking community has recognized the importance of comprehensively testing CCAs that are meant for Internet-wide deployment [23, 29, 30]. Early efforts to establish guidelines for rigorous CCA evaluations were described by Sally Floyd [29, 31], highlighting the lack of consensus within the community regarding such evaluations. In the decade and a half since that work, however, little has changed. This dissertation is an attempt to change this status quo, and we bootstrap our work through two fundamental observations and motivating factors: (i) There is a critical need to highlight the current lack of consensus regarding CCA evaluations and the resulting 'reproducibility crisis,' (ii) There exists no standardized evaluation framework for CCAs.

1.1 Dissertation Goal

The overarching goal of this dissertation is to highlight the lack of consensus in CCA evaluations, and raise the level of discourse on CCA benchmarking to focus on the *purpose* of any such evaluation, i.e., the coverage of purposes of the tests we intend to run on a CCA. Subsequently, we introduce an evaluation framework that *unifies* and *adapts* prior approaches for supporting comprehensive and open evaluations of Internet CCAs by identifying the fundamental questions that a CCA designer or reviewer expects any such evaluation to answer. To reach this goal, we have the following research objectives and their accompanying research questions:

Research objective 1: Examine the current state of congestion control evaluations, with a focus on identifying and addressing the challenges associated with reproducibility.

1. How do designers currently evaluate CCAs?
2. Are these evaluations reproducible?

To satisfy the first research objective and answer the research questions above, we assess the current state of Internet congestion control evaluations by conducting a broad literature review and reproducing evaluations for a selected set of CCAs.

Research objective 2: Analyze how the lack of consensus in congestion control evaluations creates significant challenges for the broader networking community.
The lack of a consensus on what constitutes a practical and rigorous evaluation of a CCA poses several issues to the networking community at large. It makes it hard for reviewers to compare a CCA against others, which is crucial for both gaining confidence in a specific design or refining it during development. Through our literature review and corner case analysis of select CCAs, we highlight the difficulty in determining where one CCA scheme fares better than some others, and whether this is because of the choice of evaluation metrics, or assumptions made about the operating environment, or a combination of both factors.

Research objective 3: Design a framework to facilitate systematic, rigorous and comprehensive testing of CCA proposals.

1. How does a CCA cope with the performance unpredictability of the end-to-end path?
2. How does a CCA adapt to cross-traffic?
3. How does a CCA handle workload dynamics?

To satisfy the third objective and answer the research questions above, we present a CCA benchmarking recipe by focusing on the objectives (or intents) of the evaluations, rather than the tests that comprise such an evaluation. The recipe distills, adapts, and expands decades of prior work on CCAs evaluations into three succinct fundamental questions concerning a CCA's behavior. Our benchmarking recipe offers designers quick insights into the shortcomings of CCAs, and allows operators to compare and contrast CCAs and, crucially, understand the implications of introducing a new CCA in the public Internet. Additionally, the comprehensive guideline pushes the networking community towards standardizing an evaluation framework that facilitates an open, practical, and rigorous evaluation of CCAs.

Research objective 4: Examine how the benchmarking recipe fares in examining the performance of modern Internet CCAs.
First, we use the CCA benchmarking recipe to examine performance of the BBR variants, a modern CCA which has been deployed by various Content delivery networks (CDNs), and reveal several new insights. Secondly, we evaluate a network-assited CCA that uses telemetry data provided by programmable switches.

1.2 Contributions

We support the dissertation goals through the main contributions described as follows:

Openness and reproducibility in congestion control evaluations
Approach: To highlight the "lack of consensus" in CCA evaluations as asserted in our thesis goal, we conduct a comprehensive analysis of prior work and shed light on the various approaches employed by the networking community. In the absence of a "standardized" evaluation guide for CCAs, how to evaluate a CCA is left up to the implementer. As a result, the literature exhibits a wide range of evaluation approaches, making it challenging to directly compare the performance of different CCAs. This not only makes it difficult to compare a CCA's performance but also to guarantee that an algorithm will perform well when it is eventually deployed on complex, real-world network paths. Given the vast number of possible network conditions and parameters, it is impractical for individual evaluators to fully evaluate an algorithm's performance across all potential deployment scenarios.

Key Insights: In our detailed analysis of prior work, we highlight a "lack of consensus" in congestion control evaluations. Through an extensive literature review, we find that the evaluation scenarios, conditions and even metrics used in prior work differ vastly, often with little or no justifications on these choices. Additionally, we shed light on the "reproducibility crisis" in congestion control research. Most prior work do not release their artifacts, and even when they do, those released artifacts cannot be easily re-run or re-evaluated, which adds to the challenge of evaluating CCAs. Of the 168 congestion control studies published between 2014 and 2024 at well-known networking venues, we find that *only* 36% of them had published artifacts. Among the work with publicly available artifcats, a few (2%) were deemed functional or reproducible as indicated by the respective venue's "Artifacts Functional" and "Artifacts Reproducible" badges. This widespread lack of openness hinders future work on CCAs from performing a rigorous benchmarking of the designs. Furthermore, using corner case analysis on select CCAs deployed on the public Internet, we highlight the difficulty in establishing a *performance profile* for determining where one CCA scheme fares better than others due to the various evaluation approaches employed in literature.

A recipe for benchmarking congestion control algorithms
Approach: Building on our literature review and corner case analysis, we derive a systematic evaluation approach that helps to identify CCA performance edge cases that may be found in the general Internet, and which should be considered by all evaluations concerning new CCAs to be deployed om the Internet. Our CCA benchmarking recipe adapts and expands decades of prior work on CCAs evaluations into three succinct fundamental questions that focus on the objectives of an evaluation, rather than the tests that comprise such an evaluation. The recipe offers designers quick insights into the shortcomings of a CCA, and allows operators to compare and contrast CCAs and, crucially, and understand the implications of introducing a new CCA in the public Internet.

Key Insights: To this end, we evaluated a number of well-known, modern CCAs in a range of real-world network conditions and uncovered several new insights. For instance, we find that Copa, a delay-based algorithm designed at MIT and used in Facebook's network for Android devices live video uploads [15, 32], has a tendency to underutilize available network bandwidth, especially in the face of bandwidth fluctuations. Our benchmarking recipe also reveals that when contending flows arrive at different times at a bottleneck link—a common scenario in the public Internet—PCC and BBRv1 converge to their fair shares quickly regardless of bottleneck buffer sizes, performing much better than the latest deployed BBR version, BBRv3. Had this recipe been available to the designers and operators, it might have informed the deployment plan of BBRv3. Furthermore, our recipe also shows that Sage, a learning-based CCA, which by design learns the optimal behavior from decades of prior work, struggles to converge in the typical bottleneck buffer setting in the public Internet.

Examining the performance of modern congestion control approaches
Approach: In our first case study, we use our CCA benchmarking recipe to conduct an in-depth performance analysis of modern, widely deployed Internet CCAs, Google's BBR variants. The first BBR version, BBRv1, was already reported to carry as much as 40% of all Internet traffic in 2019, just 3 years after it was introduced [19]. Recent measurement studies indicate that BBR's traffic share has increased, with traffic also transmitted using the updated variants, BBRv2 and BBRv3 [20, 21].

In our second case study, we evaluate a new, network-assisted congestion control framework for the Internet. We exploit readily available congestion telemetry data on programmable switches and implement a mechanism for separating mice from elephants at the switch by using a dedicated queue for each flow type. Thereafter, we generate a Network Congestion Feedback (NCF) signal at the switch experiencing congestion and directly send it to the flow sender. We use the NCF signal as a feedback mechanism that contains exact information about the current state of congestion, allowing the flow senders to react accordingly.

Key Insights: From the first case study, we show that throughput unfairness towards loss-based CCAs, i.e., Cubic remains an issue even in the newest version of BBR, BBRv3, and that this unfairness is nearly identical to BBRv1's behavior, especially in shallow buffer scenarios. Additionally, we find that BBR's bias towards long-RTT flows persists in BBRv3, with the unfairness magnified when the difference in RTTs of competing flows is significant. Cubic, in comparison, offers better throughput for flows with short RTTs than long RTTs. From the second case study, we showcase the benefits of isolating flow types and of early congestion control feedback. We show that NCF is the only algorithm able to prevent a flood of mice flows from substantially degrading the throughput of elephant flows, while still ensuring that mice flows experience minimal RTTs. Other findings are that NCF ensures high fairness while minimizing queue usage across a range of delay configurations, and offers these benefits even when competing flows have dissimilar RTTs under both shallow and deep buffer configurations. Compared to three widely used CCAs, NCF offers the lowest FCTs (Flow Completion Times) to short flows even in challenging multiple bottleneck scenarios.

1.3 Publications and Collaborations

Parts of this thesis are based on previously published works in collaboration with other researchers. We outline the main contributions of the thesis author to the research incorporated in this dissertation:

Chapter 3: Openness and Reproducibility in Congestion Control Evaluations
This chapter examines challenges with current CCA evaluation approaches and how the lack of consensus in evaluations creates significant challenges for the broader networking community, including the resulting reproducibility crisis. We identify performance corner cases for several modern CCAs that may have been overlooked due to the absence of standardized testing methodologies. This work is currently under submission. The author's main contribution are as follows: (i) Joint formulation of the research question, (ii) A survey of congestion control evaluation approaches, (iii) Experimental evaluation and analysis of CCAs using CCA benchmarking recipe (iv) Visualization, narrative conception and writing.

Chapter 4: A Recipe for Benchmarking Congestion Control Algorithms
This chapter introduces the CCA benchmarking recipe and the insights that laid the

groundwork for it. We emphasize the need for a standardized evaluation platform that is able to facilitate a practical and rigorous evaluation of CCAs slated for deployment on the Internet. The work described in this chapter is currently under submission in collaboration with the authors listed in [4]. The author's main contribution are as follows: (i) Joint formulation of the research question, (ii) Investigating Internet traffic profiles by analysing anonymized Internet traces, (iii) Joint design, development and implementation of the CCA benchmarking platform, (iv) Joint experimental evaluation and analysis of CCAs using the CCA benchmarking recipe, (v) Joint visualization, joint narrative conception and writing.

Chapter 5: Case Study 1: BBR Congestion Control
This chapter uses the CCA benchmarking recipe designed in Chapter 4 to take a deeper look into the performance and behaviour of a widely deployed and used CCA, BBR. It comprises of papers published at PAM 2024 and ANRW 2024 in collaboration with authors listed in [1, 2]. The author's main contributions are: (i) Addition of BBRv2 and BBRv3 kernels to evaluation platform (experiment infrastructure) (ii) Joint analysis of experiment data (iii) Joint visualization, narrative conception and writing.

Chapter 6: Case Study 2: Network-assisted Congestion Control
This chapter presents the evaluation of a network-assisted congestion feedback mechanism for Internet scenarios. NCF takes advantage of readily available congestion telemetry data on programmable switches to send immediate, sub-RTT feedback to the sender, and implement a mechanism for separating mice from elephants at the switch by using a dedicated queue for each flow type. This chapter is based on a paper published at the Workshop on Buffer Sizing (BS 2019) and another currently under submission. The papers are the results of collaborations with the authors listed in [3, 6]. The authors' main contributions are: (i) Investigation of user-space TCP stacks to enable development of end-host algorithm (i.e., NaCC) (ii) Joint development and implementation of the NaCC algorithm (iii) Joint evaluation of the system and analysis of experiment data (iv) Joint visualization, narrative conception and writing.

1.4 Thesis Structure

I structure this dissertation as follows: **Chapter 2** lays the foundation with the relevant concepts needed to better understand network congestion and traditional congestion control approaches used in TCP. We then present an overview of commonly used CCAs, and highlight research introducing modern approaches to congestion control that move away from kernel-based algorithms to user-space ones, i.e., on top of QUIC. Additionally, we take a look at approaches that take advantage of network telemetry data to strengthen congestion detection and responses, as well as the use of machine learning in developing new CCAs.

In **Chapter 3**, we highlight the challenges with current CCA evaluation approaches and how the lack of consensus in congestion control evaluations creates significant challenges for the broader networking community, including the resulting reproducibility crisis. We examine the difficulties of comprehensively evaluating CCAs faced by implementers, taking into account the various simulation, emulation and testbed setup options available to the community. We also give an overview of how benchmarking has helped advance the field in other areas, and posit what role such a benchmark can play in the evaluation and development of CCAs.

In **Chapter 4**, we present the CCA benchmarking recipe, an easily reproducible evaluation approach that provides a comprehensive overview of a CCA's performance in various network scenarios. We discuss the insights behind the recipe's design and use it to highlight several new performance insights that were missed in the original introductions of several CCAs.

Chapter 5 presents a case study carried out using our CCA benchmarking recipe to examine the performance of modern, widely-deployed Internet CCAs.

In **Chapter 6**, we present another case study to evaluate NCF using our CCA benchmarking recipe. NCF is a new, network-assisted congestion control framework for the Internet that allows for the separation of traffic based on flow types, makes use of readily available telemetry data in programmable switches, and implements a quick, sub-RTT congestion response. We evaluate NCF using the recipe and compare it's performance with widely deployed and used CCAs. Finally, **Chapter 7** summarizes the main findings of this dissertation, and outlines a discussion for future work.

2 Background

This chapter provides the foundational context for this dissertation by defining the fundamental concept of network congestion. It presents an overview of the evolution of congestion control, tracing its progression from the classic TCP approaches to more advanced algorithms. Additionally, we examine recent approaches to congestion control that deviate from traditional methods, including user-space algorithms, machine learning-based techniques, and algorithms that incorporate network signals to enhance congestion detection and mitigation. Finally, we highlight methodologies for evaluating CCAs and describe how such evaluations account for diverse network conditions, workloads, and traffic patterns.

2.1 Network congestion

At its inception, the Internet served a small number of users running minimal applications, with data transmission rates limited to a few kilobits per second (Kbps) for applications such as email, newsgroups, remote login, and file transfers [33]. End-to-end reliability was inherently assumed within the network architecture, and there were no in-built mechanisms to prevent a single user or application from monopolizing network resources [33]. However, as the Internet evolved into a large-scale, decentralized network composed of heterogeneous systems and a growing number of users running diverse applications, the resources required for data transmissions became increasingly constrained.

A significant factor driving this growth was the introduction of the World Wide Web in the 1980s, resulting in the interconnection of networks, and currently hosting millions of applications and serving billions of users [8, 11]. The Internet functions by multiplexing system resources, where network links and switch/router queues are shared among multiple users and/or applications [10]. The interconnection of multiple devices to limited network resources led to the emergence of network congestion, as the network infrastructure struggled to accommodate the growing demand [33].

Network congestion occurs when the demand for network resources exceeds the available capacity, leading to performance degradation such as increased latency, packet loss, jitter, and reduced throughput. This typically happens in high-traffic scenarios where multiple senders or applications compete for limited bandwidth [33]. Over time, incoming packets will begin to accumulate at switch buffers along the path as they await transmission over the link, which results in queuing delays. If the senders do not reduce their transmission rates, this eventually leads to packet loss when the switch buffer gets filled.

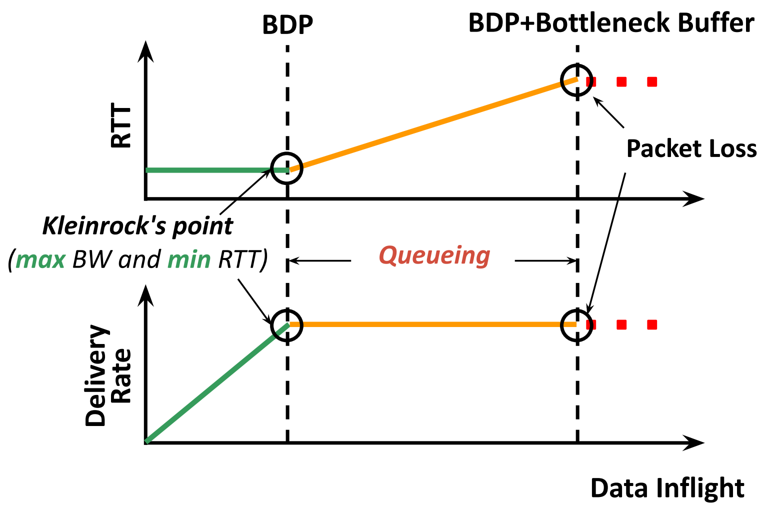


Figure 2.1: *Illustrating "Kleinrock's optimal operating point" [34] between increasing in-flight data and RTT, as well as delivery rate. Based on [35].*

Figure 2.1 illustrates this relationship between data that a sender transmits (e.g., in-flight data), RTT and delivery rate. Kleinrock's optimal operating point of a network is where maximum delivery rate is achieved with minimal RTT, indicating the most efficient use of network resources [34]. Initially, when the sender's in-flight data is insufficient to fully utilize the link, the RTT remains constant, and the sender can transmit at the rate permitted by the link capacity. Once the in-flight data reaches the link capacity, i.e., the pipe's bandwidth-delay product (BDP), the link becomes saturated and a queue starts building up at the bottleneck. This queuing results in a linear increase in RTT [35].

When the in-flight data exceeds the combined capacity of the BDP and the bottleneck buffer capacity, packet loss occurs, leading to degraded performance. In Figure 2.1, sustained operation to the right of the BDP line signifies congestion, and congestion control mechanisms aim to regulate how far rightward, on average, a flow can operate [35]. When senders do not "slow down" or reduce their transmission rates during a congestion event (e.g., sustained operation to the right of the BDP+Bottleneck line), network performance can completely collapse as the resources will be spent trying to resend lost data, a state known as congestion collapse [12].

2.2 TCP Congestion control

TCP is the most widely used transport protocol on the Internet today because it provides end-to-end reliability for data sent over unreliable links and guarantees error-free, in-order data delivery [33]. Other in-built characteristics include session management (establishing connections before data transfer and tearing down the connection afterwards), flow control (avoiding to send too much data into the network so as not to overwhelm the receiver) and congestion control (adjusting the sending rate based on observed network conditions to avoid network congestion and ensure fairness) [12]. Congestion detection and mitigation have been integral to TCP since its early days, leading to the development of various algorithms catering to diverse network conditions and application requirements. These algorithms help TCP ensure reliability by dynamically responding to and recovering from network congestion.

TCP's ability to provide reliability over unrealiable network links resulted in its adoption as the primary transport protocol for most Internet applications [36], hence, the overall performance and efficiency of the Internet are closely tied to the effectiveness of TCP's CCAs. The basic approach for managing congestion in TCP involves having senders reduce their transmission rate when congestion is detected, typically indicated by either packet loss or delayed acknowledgements (`ACK`s) [12]. However, TCP end-hosts lack visibility into the internal state of the network, and it is difficult to determine precisely when and by how much to reduce the transmission rate, and when it is appropriate to increase it again. The end-hosts also cannot know for certain whether the loss or delay was as a result of congestion, i.e., in wireless or mobile scenarios, and have to rely on implicit signals. This limited insight complicates the process of adjusting transmission speeds in response to network conditions.

The inherent limitations of end-host visibility into the network's congestion state is one of the fundamental challenges in designing effective congestion control mechanisms. Congestion signals are indirect and delayed, senders must actively probe the network to estimate available bandwidth, and bandwidth availability itself is subject to continuous fluctuations as application flows arrive and depart. Consequently, senders must dynamically adapt their sending rate based on the inferred conditions of the network.

The additive-increase/multiplicative-decrease (AIMD) algorithm offers a principled solution to this distributed control problem by balancing aggressiveness and caution: it increases the sending rate linearly in the absence of congestion signals and reduces it multiplicatively in response to detected congestion. AIMD historically emerged as the most effective known method for achieving convergence and fairness across competing flows [37]. Congestion control, as implemented through AIMD, operates in tandem with flow control to ensure both efficient bandwidth utilization and stable network behavior.

When congestion occurs in a network where senders employ AIMD, the senders get a signal to decrease their sending rate, i.e., in the form of an `ACK` indicating a packet loss or high delay. This ensures that the transmission rate is modulated based on the flow of ACKs, maintaining a controlled balance between sending data and preventing congestion. The actual amount of data sent by the sender also takes into account how much data the receiver can accept [37] in order to avoid overwhelming the receiver (e.g., flow control) as expressed in equation 2.1.

$$W = min(cwnd, rwnd) \tag{2.1}$$

where congestion window (`cwnd`) represents the amount of bytes that the sender is allowed to transmit and the receiver's advertised receive window (`rwnd`) the amount of bytes the receiver is able to receive, at a particular point in time. The sender's actual sending rate is determined by the minimum of `cwnd` and `rwnd`.

2.2.1 Slow start

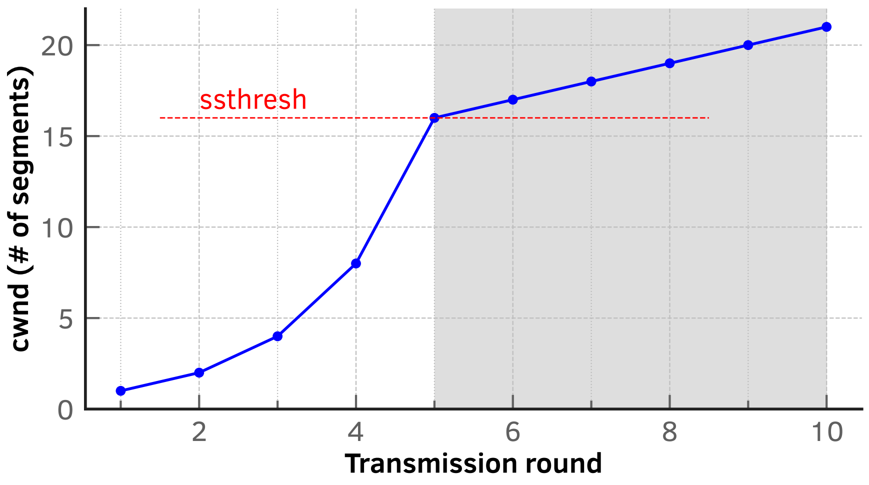


Figure 2.2: *Evolution of `cwnd` in slow start and congestion avoidance. Based on [38].*

At the beginning of a connection, network conditions are unknown. New TCP connections typically probe the network to determine the available bandwidth capacity. This approach is called *slow start* and is designed to prevent the connection from transmitting large bursts of data that could potentially overwhelm the network and cause congestion [39]. To start this phase, the sender sets its `cwnd` to a predetermined number of segments, known as the initial congestion window (`initcwnd`). In the early implementations of TCP, the default values for the `initcwnd` were highly conservative, typically ranging from 1 to 4 segments [40]. However, to accommodate modern networks that have significantly higher capacities, this default value has been increased to approximately 10 segments [41]. In recent years, many major CDNs have further moved away from this suggested standard and customize their `initcwnd` in an effort to reduce latency, with instances of `initcwnd` = 100 segments observed in the Internet [42].

The transmission of a window of data packets and the reception of their corresponding ACKs typically spans at least one round-trip time (RTT). During the TCP slow start phase, the `cwnd` increases exponentially (transmission round 1-5 in Fig. 2.2), doubling with each RTT, provided that acknowledgments are received successfully and no congestion signals are observed. This exponential growth of the `cwnd` enables the sender to quickly discover the bottleneck capacity along its path. Consequently, this exponential growth increases `cwnd` fairly rapidly, eventually surpassing the bottleneck link capacity and causing a congestion event, i.e., packet loss or a delayed `ACK`, which in turn forces the TCP connection to slow down. The point where TCP makes the switch from exponentially increasing its `cwnd` is called the *slow start threshold (ssthresh)* [39] (see Fig. 2.2).

2.2.2 Congestion avoidance

After reaching *ssthresh*, a TCP connection switches from exponentially increasing its `cwnd` to only increasing it by one segment for every successfully delivered packet window (i.e. 1 packet per RTT). This helps to keep probing for any additional capacity that may become available but without doing so aggressively, in a phase called *congestion avoidance* [39], shown by the shaded region in Fig. 2.2.

In contrast to the exponential increase of `cwnd` employed during the slow start phase, the congestion avoidance phase follows a more linear progression, resulting in a gradual increase in the sender's transmission rate. This allows a sender's `cwnd` to be as close as possible to the optimum bottleneck link capacity but without large increases that would cause further congestion on the link. This approach enables the sender to cautiously probe the available network capacity by incrementing the `cwnd` one segment at a time. If the transmission of an additional packet causes the buffer at the bottleneck link to overflow, the number of packets in flight would have been minimal, leading to an easier recovery process [39]. The `cwnd` of a TCP connection is typically characterized in relation to the BDP of a network path, which is defined as the product of the network link capacity and the RTT. Maintaining a `cwnd` near the BDP allows a sender to keep the link busy without increasing queue occupancy at the bottleneck buffer.

2.3 Evolution of congestion control algorithms

The field of congestion control has remained a significant area of research since the early development of the Internet. Researchers have proposed a wide range of approaches, from designs that enhance the performance of basic TCP algorithms to those that aim to cater to increasingly high-speed, high-bandwidth networks. Additionally, algorithms have been developed to handle evolving network dynamics (i.e. wireless, cellular, satellite links with variable delays and bandwidths [43–45]), or deal with applications that have stricter Quality of Experience (QoE) guarantees, such as video streaming or Real-time communication (RTC).

CCAs largely determine the manner and extent to which applications utilize network resources (i.e., bandwidth), and empower applications to adapt their delivery rate

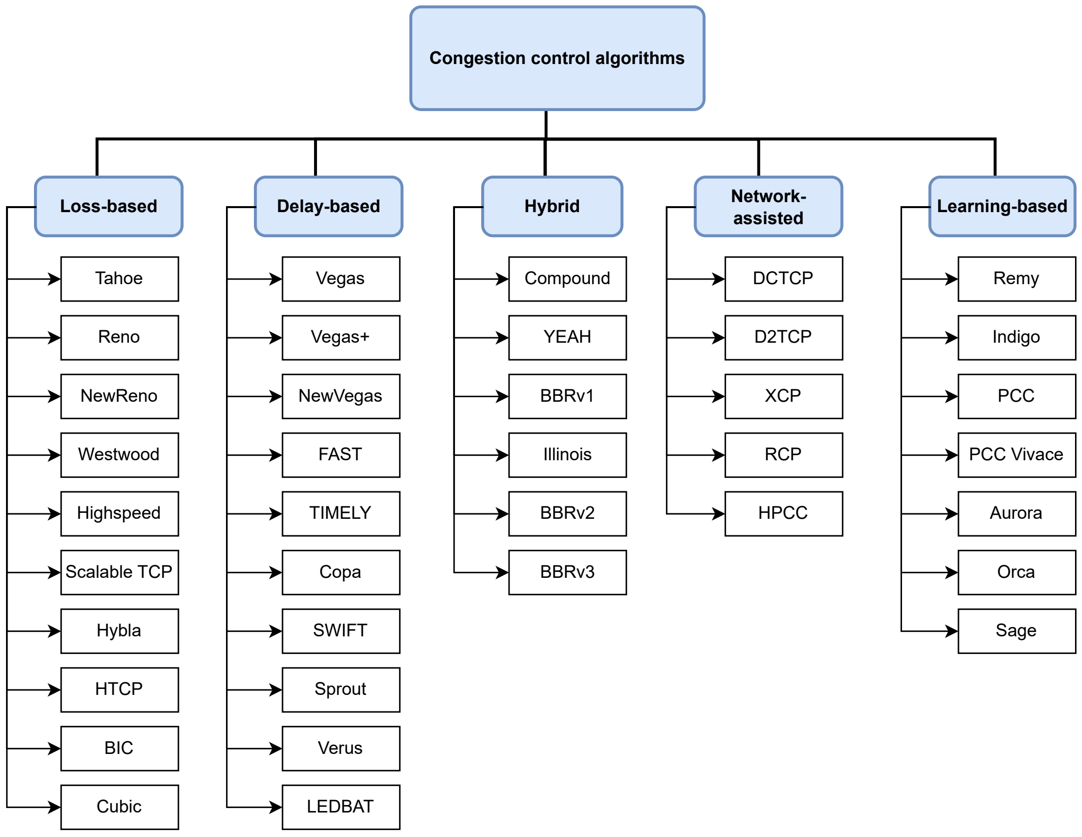


Figure 2.3: *CCA families.*

to available bottleneck bandwidth. Additionally, they allow contending applications to share the bottleneck bandwidth equitably among each other, and, thereby, reduce the likelihood of or prevent congestion in the network. This section examines key classes of CCAs that have been introduced to address congestion in various network scenarios as illustrated in Fig. 2.3.

2.3.1 Loss-based algorithms

In response to the network congestion collapse of the late 1980s [12], Van Jacobson introduced TCP Tahoe, which employed the slow start mechanism and reacted to packet loss by resetting the `cwnd` to one segment, regardless of whether the loss was detected via a timeout or three duplicate `ACK`s. This approach led to significant underutilization of network capacity, as the `cwnd` was always reset to one following packet loss, greatly reducing transmission efficiency [12]. To mitigate this issue, TCP Reno [40] was introduced in 1990, incorporating a "fast recovery" algorithm which, instead of resetting `cwnd` to 1 after detecting loss via three duplicate `ACK`s, enabled a TCP Reno sender to retransmit the lost packet and thereafter set `cwnd` to be equal to the previous *ssthresh* value. This modification allowed TCP Reno to maintain the `cwnd` closer to the optimal value of the BDP, resulting in better link utilization and more efficient recovery from packet loss [33].

The TCP Reno fast recovery algorithm had limitations when multiple packet losses occurred within a single transmission window. Specifically, a TCP Reno sender would prematurely exit the fast recovery process upon receiving a cumulative ACK for packets that were not yet retransmitted, leaving some lost packets unhandled [12]. To address this issue, TCP NewReno [46] was developed with the ability to respond to partial `ACK`s (i.e., that acknowledge only some of the transmitted packets) by retransmitting the packets that were sent after the one acknowledged by the partial `ACK`. This mechanism ensures that TCP NewReno can handle multiple packet losses within the same transmission window without waiting for a retransmission timeout. TCP NewReno remains in the fast recovery phase until all outstanding data from the start of fast recovery has been acknowledged, allowing for more efficient recovery from multiple losses [46].

Although TCP NewReno was considered stable and was therefore widely deployed, as link BDPs increased, its performance became insufficient. To address the challenges posed by high-BDP links, several new CCAs were proposed, among others High-Speed TCP (HSTCP), TCP BIC and TCP Cubic [47]. TCP Cubic optimizes performance on high-speed, long-distance networks using a cubic function to control its `cwnd` growth, resulting in smoother and faster adjustments to the bottleneck bandwidth. After a packet loss, TCP Cubic takes note of the size of `cwnd` at which the loss occurred and denotes it as W_{max}, such that its cubic function is centered around this W_{max} value for the next iterations. When `cwnd` is near W_{max}, the function grows slowly to avoid overshooting the bottleneck link capacity–thereby helping to avoid congestion, and when `cwnd` is far below or above W_{max}, the function accelerates growth, to enable a sender to quickly utilize available bandwidth. It has become as widely deployed as TCP NewReno and has been the default CCA in Linux since kernel version 2.6.18 [47].

2.3.2 Delay-based algorithms

The loss-based algorithms—such as Tahoe, Reno/NewReno, Cubic—all seek to determine the available bottleneck bandwidth by sending as much data as possible and filling up the bottleneck buffer. This results in persistent queues, which may under some scenarios result in bufferbloat, where large buffers are frequently full resulting in substantial packet latencies [48]. Thus, an alternative branch of CCAs which estimate congestion by measuring end-to-end delays along the network path, began with the introduction of TCP Vegas in 1994 [49].

TCP Vegas estimates the expected link throughput using `cwnd` and the *minimum observed* RTT. This expected throughput is then compared to the actual throughput, calculated from the currently observed RTT. The difference between these two values is evaluated against two thresholds, α and β, which dictate whether the sender should increase, decrease, or maintain its current sending rate. For instance, if the actual throughput is significantly lower than the expected throughput, the network is likely congested, and the sender should reduce its sending rate. This mechanism allows TCP Vegas to be proactive in detecting congestion early, without filling up the bottleneck buffer, thereby avoiding increased latency, frequent retransmissions, and oscillatory behaviours observed in ‘loss-based’ TCP versions [49].

A downside to TCP Vegas is that, since it aims to keep RTT low and therefore the buffers as empty as possible, it underperforms when competing with more aggressive, buffer-filling TCP versions, e.g., Reno or Cubic [12]. Additionally, TCP Vegas’ RTT estimation can be disrupted in cases where the network route changes leading to a decrease in throughput [50]. Nevertheless, using delay as an indicator of congestion has proven beneficial, and has inspired the development of numerous widely adopted CCAs including Compound TCP (CTCP) [51], used by default on all Windows servers, Low Extra Delay Background Transport (LEDBAT) [52], used for background data transfers, and the bottleneck bandwidth and round-trip propagation time (BBR) variants, developed by Google, and currently used by several CDNs [19, 21].

The prevalence of high-capacity, high-speed Internet connections led to the emergence of applications with increasingly stringent performance requirements, necessitating the evolution of Internet infrastructure to accommodate a diverse range of services. Furthermore, new connectivity options—e.g., 5G cellular networks, WiFi-7, and terrestrial Internet via low Earth orbit (LEO) satellites like Starlink—have been deployed, offering users improved speeds. This enhanced Internet infrastructure supports an increasing number of applications with varying traffic flow requirements, and which must coexist despite their distinct needs (i.e., high throughput versus low latency). Additionally, user expectations for application performance has become increasingly high [53, 54].

As a result, substantial investments have been made by industry and academic researchers alike to enhance network and application performance, including the development of new CCAs to handle this evolving mix of Internet traffic, and primarily focusing on controlling latency and jitter. Of the most widely used of these new CCAs is the BBR algorithm developed by Google in 2016 [35]. Unlike prior CCAs, BBR characterizes a network path by periodically estimating the bottleneck bandwidth and RTT, and appropriately paces the sender to avoid queues building up along the path. BBR’s goal is to quickly converge to Kleinrock’s optimal operating point [34]. Essentially, in stable state, BBR aims to maximize throughput while minimizing delay and loss.

As a result of BBR’s good performance in avoiding queue build-up while maintaining a high transmission rate, Google has used BBR for a significant fraction of its network traffic both within its datacenters and on its WAN since 2017 [55]. However, many researchers have scrutinized BBR’s interaction dynamics with loss-based CCAs–still the most widely used CCAs in the Internet–and found BBR to be highly unfair [1, 2, 26, 56].

In 2019, Google released BBRv2 to address issues seen in BBRv1 [57]. Unlike BBRv1, BBRv2 reacted to loss and/or ECN signals to facilitate an equitable sharing of bandwidth with loss-based CCAs e.g., Cubic and NewReno, and optimized the `cwnd` update logic [57]. Despite the inter-CCA and RTT fairness improvements that Google reported [57, 58], independent evaluations showed it suffering from low link utilization [1, 59–61].

BBRv3, the most recent release, was introduced in July 2023 at the 117th IETFs meeting, with claims of addressing several concerns with the previous BBR versions [62]. Described as a minor revision of BBRv2, its key improvements include quick bandwidth convergence to fair shares both with and without loss or ECN signals, as well as optimizations to minimize queuing delays and packet losses both during and shortly after slow start. BBRv3 is currently being considered for adoption in the IETFs congestion control working group (CCWG) [63].

Another of the delay-focused CCAs is Copa, designed at MIT [15] and deployed at Facebook for their Android video streaming platform [32]. Copa estimates queueing delay from its RTT observations and aims for a target sending rate that is inversely proportional to the observed queueing delay in order to keep queues at the minimum [15]. Other CCAs such as Sprout [44] and Verus [45] are specifically designed for cellular wireless networks and leverage delay information to adapt to the dynamic nature of cellular channels and optimize their rate control mechanisms. These specialized congestion control methods often demonstrate significantly better performance compared to traditional, general-purpose schemes, especially in wireless scenarios [15].

2.3.3 User-space congestion control

Due to its reliability features, TCP has long been the dominant transport protocol, and significant efforts and resources have been invested in optimizing its performance over the years [33, 36]. In certain use cases such as video streaming or HTTP-based web transfers, these optimizations seem to have reached a performance ceiling, showing limited improvements in efficiency and user Quality of Experience (QoE) [64, 65]. In the case of HTTP web transfers, the limitations come from the head-of-line (HOL) blocking at the TCP bytestream level as well as TCP's need for a handshake before establishing a session between a client and a server [65]. When Transport Layer Security (TLS) [66] is employed to secure these sessions, an additional handshake is introduced, further increasing the latency associated with downloading web pages. This challenge is particularly pronounced for typical web transfers, which often involve relatively small amounts of data [67], and could ideally be completed within a short time frame if not for the cumulative overhead of TCP+TLS+HTTP handshakes.

To reduce web latencies and address issues with TCP's ossification, Google introduced QUIC [64]. QUIC is a UDP-based protocol that runs at the application layer and has since been standardized by the IETF [68]. It is expected to replace TCP in

HTTP/3 [69], and has been reported to already carry a significant portion of Internet traffic [70].

QUIC substantially reduces web latency by eliminating the need for multiple handshakes during connection establishment. Moreover, it moves logic for flow control and congestion control mechanisms to the application layer, offering developers a flexible framework to rapidly design, deploy, and test novel or modified CCAs, without the hassle of otherwise necessary OS upgrades. Although most CCAs on QUIC are based on existing TCP CCAs, many CDNs have leveraged QUIC to tailor existing CCAs to their specific requirements [22], introducing even more heterogeneity in CCAs being used for data transfers in the Internet. Consequently, previous studies have shown that most CCAs implemented within these different QUIC stacks deviate from their TCP kernel counterparts [22, 71].

2.3.4 Network-assisted congestion control

A key limitation of classic TCP CCA proposals is their reliance on imprecise congestion signals, typically reflected by the receiving host. This dependency results in a delay of at least one RTT before corrective measures can be applied. To supplement these approaches, several mechanisms were proposed to elicit explicit network support for congestion control, including DECBit (setting a bit in the packet's header traversing through a congested router) [72], RED (dropping packets when queue occupancy exceeds a defined threshold) [73], and ECN (marking the packet's IP header with a 'congestion experienced' bit) [74]. Other approaches recognized the importance of reducing the congestion feedback loop, and use in-network signaling to send congestion signals directly to senders. An early initiative in this direction was QCN [75], which provided direct feedback to senders, although these congestion notifications are restricted to the L2 domain.

The early in-network congestion control solutions inspired plenty of follow-ups, particularly within the realm of datacenters (e.g., XCP [76], RCP [77], DCTCP [78] and TIMELY [79]). XCP uses a congestion header to communicate flow state (e.g., `cwnd` and RTT) between routers and receivers, while RCP requires routers to compute per-flow rates which is then reflected back to the sender by the receiver. On the other hand, DCTCP adjusts its `cwnd` based on the scale of congestion at the router (e.g., based on the number of ECN-marked packets received), while TIMELY relies on accurately estimating RTT in a datacenter environment where end hosts support hardware time stamping.

The advent of programmable switches made it feasible to have explicit in-network congestion signaling by facilitating the use of in-depth information from In-band Network Telemetry (INT) metadata. This enabled mechanisms such as FastLane [80], HPCC [81] and Bolt [82] to leverage this information to refine their congestion mitigation strategies. FastLane uses programmable switches to send high-priority notifications directly to senders upon detecting a packet drop, enabling the rapid retransmission of lost packets. In contrast, HPCC adds data about the current load on the switch egress port to traversing packet headers, and relies on receivers to reflect this information back to senders [81]. Similarly, Bolt [14] uses precise, direct congestion information from the switch to proactively control a sender's rate.

Like Bolt, proposals such as RoCC [83] and BFC [84] also use sub-RTT congestion signaling at intermediate nodes. RoCC tracks elephant flows at the switch and calculates each flow's fair rate using the queue size, although it is designed specifically for RDMA traffic. BFC is a per-hop, per-flow control mechanism that uses dedicated queues for optimum performance, and requires upstream switches to pause queues when a bottleneck sends congestion signals, an approach impractical in WAN environments [83]. Zhuge [16] is a recent proposal in the wireless networks realm that shortens the control loop by modifying the access point (AP) to inform the senders of the delay experienced by its packets at the AP.

On the queueing front, recent advancements include the Low Latency, Low Loss, Scalable throughput (L4S) framework that improves congestion quantification and reduces packet delays by signaling congestion earlier than the traditional ECN approach [85]. L4S combines a scalable CCA (e.g., TCP Prague [86]), Accurate ECN (AccECN) [87], and a dual-queue AQM [88] that isolates L4S and non-L4S traffic at the bottleneck. Efficient operation of L4S requires both endpoints to support its specific signaling mechanisms.

2.3.5 Learning-based algorithms

A majority of the CCAs in use today primarily rely on pre-defined rule-based mechanisms to manage congestion. However, the complexity and heterogeneity of modern networks pose significant challenges to such rigid approaches, and rules that perform well in one environment may exhibit suboptimal performance in others, or in scenarios where network conditions rapidly change. To address these limitations, researchers have explored how to leverage machine learning algorithms to dynamically adapt to changing network conditions. Unlike rule-based approaches, the learning-based CCAs use historical data to classify or predict outcomes for network conditions.

One of the first examples of a learning-driven CCA is Remy [89], which is designed to optimize transmission behavior based on a pre-modeled objective function. The Remy parameters can be tuned to adjust the aggressiveness of the protocol [89]. Similarly, Indigo [90] adopts an offline data-driven approach, leveraging insights from training on captured networks traces to fine-tune congestion control parameters at the sender. The Performance-oriented Congestion Control (PCC) framework introduced several CCA variants that aim to optimize a specified objective function dynamically at runtime, unlike Remy and Indigo [91]. For instance, PCC Vivace [92] uses online optimization to move towards a sending rate utility value derived from observed performance metrics. This utility value helps it to ensure high bandwidth utilization and fast, stable convergence to available capacity.

Other CCA schemes leverage deep reinforcement learning (DRL) to improve rate adaptation. Aurora [93] uses DRL to adjust the sender's rate based on a linear reward function, which prioritizes throughput while penalizing loss and latency. Similarly, Orca [94] combines DRL with traditional congestion control mechanisms to calculate a reward function based on observed packet delivery rate, delay and loss. Recently, Sage [18] used a data-driven approach to learn from existing TCP CCAs, and automatically develops an optimized congestion control policy that improves performance in varying network conditions.

2.4 Evaluating congestion control algorithms

The extensive body of research on CCAs outlined in §2.3 contributes to the complexity of determining how any one scheme falls short in comparison to the rest, and, most importantly, along what dimensions. Performing pairwise comparisons to help explain performance discrepancies between schemes is hard because most algorithms are designed with distinct objectives, leading to varied behavior that is further influenced by the underlying network environment. Nonetheless, comprehensive testing of CCAs is crucial to prevent harm to existing network traffic, especially on shared network environments like the public Internet.

A key challenge in performing rigorous CCA evaluations lies in the high-dimensional nature of the parameter space. Many factors influence the behavior of a CCA, including end-host system settings (e.g., kernel configuration and NIC optimizations), network configurations (e.g., buffer size, delays, and queuing discipline) to traffic conditions (e.g., mixing of different flow distributions and types of cross-traffic). Additionally, performance trade-offs may only become evident through the examination of a broad set of performance metrics across diverse network conditions. Since there is no universally accepted methodology for evaluating CCAs, nearly every work that proposes a CCA cherry-picks the metric or evaluation scenarios (e.g., topology and workload) to demonstrate that it meets the objectives identified by its designers. This approach limits generalizability, raises concerns about bias in performance assessments, and risks overlooking critical failure points when CCAs are deployed in real-world environments.

Efforts have been made to establish common evaluation metrics [23, 29], simulation frameworks [95], emulation tools [96–98], and testbeds for research [90, 99]. Mininet [96] enables the emulation of network topologies using Linux network namespaces [100], facilitating reproducible experiments independent of hardware constraints. TEACUP [97] and FLENT [98] leverage physical testbeds to support reproducible CCA evaluations by providing test automation frameworks and mechanisms for data collection and analysis (although FLENT can be used with both physical hardware and network namespaces).

Another proposal is the Pantheon [90] which sought to address the issue of reproducibility by providing a shared platform for evaluating CCAs. It included 17 CCAs and publicly archived evaluation results. Transperf [99] has been used by the Google BBR team during the development of the BBR algorithms and can be used with physical hardware, however, it has limited evaluation scenarios and is not widely used.

2.4.1 Evaluation topologies

The dumbbell topology is the most commonly used topology in CCA evaluations, where two sets of hosts—typically, senders or traffic "sources" and receivers or traffic "sinks"—are connected using a three-hop network path with one bottleneck link between two (on-path) routers [30]. This topology is easy to set up, even on a low budget, and is quite simple: By simply altering the bottleneck link configuration (e.g., bandwidth and delay), a CCA's behavior can be evaluated in a range of network conditions. Moreover, one can use it to effectively illustrate a CCA's behavior in the face of congestion and packet loss because the topology emulates the common scenario where a flow traverses only one congested link. However, while the dumbbell topology provides useful insights, it is not representative of the complexities of real-world Internet paths.

In reality, paths between two arbitrary hosts in the Internet may traverse through datacenters, Internet Service Providers (ISPs), and Internet Exchange Points (IXPs), with varying router configurations and traffic contention levels. Besides, a flow traversing such a path may contend with different sets of flows (i.e., cross-traffic) at each hop—a scenario that the simple dumbbell topology cannot model adequately. For instance, datacenter networks often commonly employ Clos Networks/Fat-Tree topologies [101], which may shape outgoing traffic in ways that influence CCA behavior. However, large-scale datacenter infrastructures are not easily accessible for research purposes, making controlled experiments on such networks challenging.

To counter the shortcomings of the dumbbell topology in modeling the complexity of the Internet, researchers often supplement the dumbbell topology with the "parking lot" topology [30], which can model multiple-bottleneck scenarios. The parking lot topology has been used in various empirical evaluations of CCA behavior [102–107], as well as in theoretical analysis to study max-min fair resource allocation [108, 109]. While direct evidence of parking lot topologies in real networks is limited, previous work [110] and our findings in Chapters 4 and 6 demonstrate that multiple bottlenecks along a flow path can significantly impact CCA performance, and that results from single-bottleneck scenarios do not necessarily generalize to multiple-bottleneck network scenarios.

To analyze real-world CCA behavior, researchers conduct experiments over the "wild Internet". These type of experiments are inherently difficult to reproduce, as the traversed network topologies, conditions, and competing traffic characteristics are unknown and vary dynamically. Incorporating the parking lot topology into evaluations enhances robustness by providing a controlled setup where flows traverse several contention points with cross-traffic, enabling a systematic assessment of a CCA's response to diverse Internet conditions.

2.4.2 Evaluation metrics

Applications often have different performance requirements, and no single metric can capture how well a CCA caters to the application's needs. Web-browsing and gaming applications are typically more sensitive to latency, and flow completion times (FCTs) or "slowdowns" [28] are hence well-suited to evaluate the choice of CCA for such applications. In contrast, video-streaming applications are both latency and bandwidth sensitive, thus a performance profile for these applications is better observed by looking at throughput, loss, and convergence time (i.e., time for a flow to reach its "fair" share). Some applications may also be tolerant to loss, while others not, and many applications' performance requirements may even change over time. Vielhaus et. al [111] provide a comprehensive survey of important metrics to consider for different CCA evaluation approaches.

Since designers may not know a priori how the deployment environment may evolve over time, choosing a battery of different metrics can help in preventing poor deployment experiences. Extending the choice of metrics beyond what is best suited for the application in hand also helps future researchers to realize what performance tradeoffs the designers had made. For instance, metrics that provide insights into the interactions between cross-traffic with differing RTTs, as well as inter-CCA behavior include Jain's fairness index, *fness* [112], and *Harm* [27]. A more detailed discussion on such metrics is presented in Sally Floyd's memo [29]. The selection of appropriate metrics allow us to quantify a CCA's compatibility with competing network flows and assess its broader implications for application performance.

2.4.3 Workload considerations

Designers typically assess the performance of CCAs using long-running flows (or *elephants*), which enable comprehensive observation of the CCA's behavior throughout the life cycle of the flow. The MAWI dataset [67] provides daily network traffic traces from trans-Pacific link collected over 16 years, and has been widely used for network traffic analysis. This dataset shows that long-running flows contribute most towards traffic volume (or size) in the Internet. However, it also reveals that flow sizes vary a lot and exhibit a heavy-tailed distribution. A significant fraction of the flows are short-running (or *mice*) and have fewer than 10-12 packets. These flows terminate well before they progress to the (typical) congestion avoidance phase, making it impractical to congestion control them. As a result, such short flows can saturate a network link at any time [113].

Sampling realistic workloads helps enable a robust evaluation of a CCA's performance across different flow sizes and assess its ability to cope with dynamic traffic changes (i.e., due to flow arrivals and departures). Especially in the Internet, where we do not have much control over flow arrivals or types or sizes, evaluations based on "'real network traffic' are crucial for avoiding the introduction of a new CCA wreaking havoc on existing traffic in the Internet [114]. An appropriate workload tests the CCA's ability to cater to different flow sizes at different load levels in a network, and can help determine whether a CCA is best suited for a concerned application or environment.

2.5 Summary

This chapter provides a brief overview of congestion control, covering both the traditional CCA approaches and more recent proposals. Despite extensive research and development, and the long-standing history of congestion control, a robust algorithm that performs well across a diverse range of network conditions remains an ongoing challenge.

The various approaches discussed in this chapter underscore the heterogeneity of the current CCA landscape and highlight the challenge in selecting an appropriate CCA for specific network environments. Moreover, we emphasize the importance of rigorous and robust performance evaluations for comparing different algorithms as well as for understanding their interactions, particularly within shared networks such as the public Internet, where multiple CCAs coexist.

The diverse range of tools, methodologies and metrics used to evaluate CCAs introduce significant reproducibility challenges, particularly when assessing prior designs under new conditions and scenarios. These challenges arise due to several factors, including CCA designers *not* making their implementations (in their entirety) publicly available, not explicitly declaring all design choices and assumptions, or failing to sufficiently describe their testbeds or evaluation setups. Such limitations hinder explainability about CCA behavior, making it difficult to draw consistent and generalizable conclusions. The wide range of congestion control strategies discussed in this chapter necessitates the development of comprehensive and rigorous evaluation methodologies to assess CCA interactions and overall efficiency using various topologies, metrics and workloads.

3 Openness and Reproducibility in Congestion Control Evaluations

Research on CCAs spans more than four decades with an extensive body of literature and its pace shows no signs of “slowing down.” Most networking venues have consistently featured one or more publications on congestion control research every year since their inception. We highlight, unfortunately, a critical threat to congestion control research—the lack of a community-wide consensus on evaluating CCA designs.

In this chapter, we address our first research objective by analyzing the current state of congestion control evaluations. Notably, the methodologies used to assess various CCA designs significantly differ from one another. Designers often pick evaluation scenarios (e.g., topology) and network conditions (e.g., workload) that best represent their deployment environments. Similarly, many prior work often prioritize metrics that best highlight the novel features of their designs. While these curated set of “tests” might be well-justified,[1] the lack of a consensus on evaluating CCAs poses a huge risk for both research as well as practice. The cherry-picked tests, for instance, do not present a *complete* evaluation, which makes it hard to determine whether the key insights from a prior work generalize to other scenarios that the designer did not consider. The differences in evaluation scenarios and metrics also make it non-trivial to perform comparative evaluations of one CCA against others. We find (and show in §3.2) that evaluators often do not fully disclose the details of their testbed and/or release their “tests,” which exacerbates the already arduous task of conducting rigorous and comprehensive comparative evaluations of a CCA against prior work. The solution to remedy these issues is rather simple: embrace an open, reproducible, standardized approach to evaluating CCAs.

Is not standardizing CCA evaluations an “old hat”? Floyd’s memo on the metrics for evaluating CCAs was the earliest attempt to *standardize* CCA evaluations [29]. Researchers have since introduced several new metrics (e.g., “harm” [27] and “slowdown” [28]). Without a unifying framework, it is difficult to argue where one metric fares better than another; unsurprisingly, not all metrics are widely adopted, with some researchers even questioning the utility of innovative metrics (e.g., [115]). The idea of evaluating CCAs in terms of trade-offs between a range of metrics [29]

[1]Researchers must indeed choose tests that accurately characterize their novel design elements and compare them to relevant prior work to publish their design.

seems to have been largely forgotten, and many studies often do not present the rationale behind their evaluation choices. There has been renewed interest in establishing a common set of criteria for characterizing the behavior of CCAs (e.g., [30, 116]). Enumerating a list of metrics or scenarios that everyone must or should follow is, however, unlikely to reach quick consensus or adoption: People will disagree about what is included or excluded, and an exhaustive list will place an undue burden on future work (thus, disincentivizing adoption). Such efforts may fracture the community with each faction adopting a different "benchmark" [117].

What tools facilitate reproducible evaluations? It is hard to reproduce the results from prior work, let alone testing prior designs under new conditions and scenarios. CCA designers do *not* typically make their implementations (in their entirety) publicly available or explicitly declare all their design choices and assumptions. Many prior work do not sufficiently describe their testbed or evaluation setup. New CCAs, in turn, limit their evaluations to a subset of prior work and use evaluation scenarios that facilitate such comparisons, and may also fall trap to repeating the mistakes of past work. While there has been work on designing tools and testbeds to enable reproducible CCA evaluations (e.g., TEACUP [97], FLENT [98], and Pantheon [90]), their scope has been rather narrow and adoption (perhaps as a consequence) quite limited. While the Pantheon made significant strides towards enabling open, reproducible evaluations, it had several limitations. The emulator did not support running flows using different CCAs concurrently to profile their interaction dynamics. Our own attempts to extend the platform revealed several shortcomings: It struggled to scale beyond a few tens of flows, did not support multiple bottleneck scenarios, and did not allow dynamic adjustment for various configuration parameters, e.g., RTT of a flow.

What can we learn from other areas of research? Establishment of a community-wide consensus on what constitutes a fair and rigorous benchmarking of techniques in a field has often served as a catalyst for ground-breaking innovations in that field. George Miller's *WordNet*, an electronic lexical database, has had a profound influence on research in the fields of computational linguistics and natural language processing [118], since any benchmarking of algorithms and techniques in these fields relied on WordNet. Deng et al.'s *ImageNet* laid the foundation for benchmarking machine-learning algorithms (e.g., object recognition and image classification) and paved the way for the current advancements in the field of deep learning and AI [119]. Similarly, TPC benchmarks [120] and SPEC [121] are the *de rigueur* for characterizing the performance of transaction-processing environments or databases and compute-intensive environments, respectively. These benchmarks, and several others, were effective and influential in advancing the concerned fields because they established a clear consensus on the evaluation criteria. They enabled an unambiguous "yard stick" to track progress and facilitated reproducible evaluations. Congestion-control research has not had such a benchmark, since there is no clear consensus on what such a benchmark should *cover*.

The rest of this chapter is organized as follows. In §3.1 we discuss prior work in CCA evaluation approaches. In §3.2 we review the status quo in congestion control research and highlight a reproducibility crisis and a clear lack of consensus concerning CCA evaluations. We follow this up with a few case studies to demonstrate the implications of these shortcomings on research as well as on practical deployments in §3.3 and summarize our findings in §3.5

3.1 Related Work

The plethora of CCAs coexisting on the Internet [19–21] has encouraged the community to think about the need for rigorous evaluations prior to deployment. Early efforts to formalize the assessment of CCAs include Sally Floyd's memo [29] and the 2014 Common TCP Evaluation Suite draft [30]. It is hard to carry out the extensive set of tests that such memos suggest, and most work, hence, cherry-pick what they want to test and how.

CCA designers often use network simulators, such as NS-3 [95], OMNeT++ [122], and GloMoSim [123], to model real hardware behavior and evaluate CCA performance. The choice of evaluation scenarios and parameters, however, largely remains at the discretion of the algorithm designer. Similarly, tools such as TEACUP [97], Mahimahi [124], FLENT [98], and the Pantheon [90] are some of the platforms developed to help ease the evaluation of CCAs in a reproducible way. Another related line of work is on reverse engineering and formal verification of congestion control behaviour [24, 125–127]. This line of work focuses on finding network environments where CCAs perform well/badly, and when badly, find reasons why. This helps in providing performance guarantees for a new CCA. Other intended outcomes are to find bugs in the implementation of a CCA. However, pointing out where a CCA performs badly, or its implementation bugs does not help the next designer know *how* to compare their proposed new CCA with the existing one. The work in [128] provided a comprehensive empirical evaluation of CCAs, marking an important step toward understanding CCA behavior across multiple dimensions.

Establishing consensus with regards to CCA evaluations may also mitigate inconsistencies brought about by user-space CCAs. Recent work in [22, 71] presented *QUICBench*, a benchmarking tool to evaluate how well CCA implementations in QUIC stacks match their corresponding TCP kernel implementations. *QUICBench* uncovered significant non-conformance in several implementations, largely attributed to the absence of a systematic approach to implementing both the QUIC stacks and their associated CCAs [22, 71]. This lack of consistency has resulted in performance disparities and fairness issues among different QUIC implementations' CCAs. The establishment of standardized CCA evaluation guidelines would encourage that such performance disparities are documented before such stacks are deployed.

Table 3.1: *Survey of prior work on CCAs published between 2014 and 2024.*

Venue	#pubs.	Percentage of publications				
		Art. Avail.	Art. Func.	Art. Rep.	In-text	Online
ATC	6	33	17	17	50	0
CoNEXT	10	0	0	0	60	0
EuroSys	2	0	0	0	50	0
HotNets	19	0	0	0	32	0
IFIP	12	0	0	0	42	0
IMC	17	6	0	0	41	0
INFOCOM	22	0	0	0	14	0
NSDI	14	0	0	0	21	50
PAM	5	0	0	0	20	0
SIGCOMM	29	21	10	7	31	14
SIGMETRICS	3	0	0	0	33	33
ToN	29	0	0	0	21	0
Total(%)	**100**	**5**	**2**	**2**	**29**	**7**

3.2 The Status Quo

In line with Research Objective 2(§1.1), we examine how the absence of consensus in CCA evaluations presents significant challenges for the broader networking community. Despite the rich literature on congestion control, there is an ongoing reproducibility crisis concerning research on CCAs, and the lack of a clear consensus on how to evaluate CCAs hinders the pace of research. In the context of CCAs designed for the public Internet, these challenges affect not only the researchers but also network operators (e.g., concerning deployment decisions) and end users (e.g., concerning performance and fairness).

3.2.1 A reproducibility crisis

Many prior work on CCAs do not either make *all* of their implementation artifacts publicly available or clearly document their testbed setup and/or evaluation approach. In this section, we quantify the reproducibility crisis in congestion control research through a manual survey of prior work.

We surveyed a decade of prior work on CCAs published at well-known networking venues since 2014 to determine what fraction of them publish their artifacts (Table 3.1). In Table 3.1, the columns "Art. Avail.", "Art. Func.", and "Art. Rep." correspond to the ACM artifact-review badges "Artifacts Available", "Artifacts Evaluated and Functional", and "Artifacts Reproduced," respectively. Similarly, the columns "In-text" and "Online" refer to papers without explicit badges where we found artifact URLs in the text or by searching for it online, respectively. We included any CCA-related work, whether it introduced a new CCA or simply evaluated an existing design. We labeled a work as reproducible if it had an explicit badge indicating the availability of artifacts. For publication venues that do not provide such badges, we looked for an artifact URL in the paper or manually searched for it online. We did not, however, evaluate available artifacts, and we only provide the percentage of publications with the ACM "Artifacts Reproducible" or "Artifacts Functional" badges

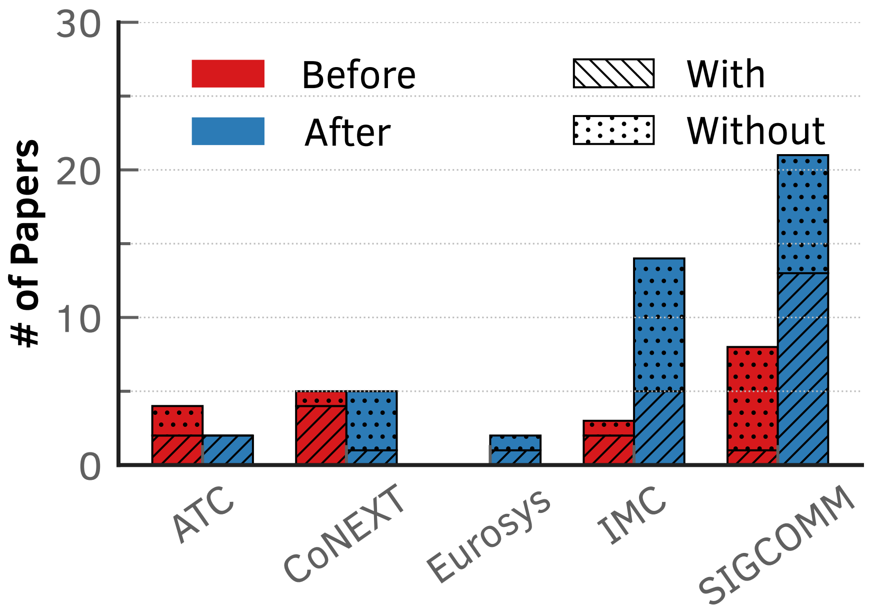


Figure 3.1: *Number of reproducible CCAs publications, i.e., with published artifacts, before and after the introduction of artifact evaluation badges at various networking venues.*

for completeness. Among the 168 prior work that we surveyed, *only* 36% of them had published artifacts.[2] Besides, among the prior work with publicly available artifacts, only a few (2%) are deemed functional or reproducible. This issue of reproducibility is indeed applicable broadly to networking research at large [129] and prevalent in other areas of natural sciences [130].

This widespread lack of openness hinders future work on CCAs from performing a rigorous benchmarking of the designs. Such openness is crucial for the community to evaluate CCA designs before they are deployed in the global Internet. The mismatches between Google's claims concerning BBR's behavior [35, 57, 58, 62, 131] and those observed by independent researchers [1, 25, 114, 132], for instance, serve as a good case study. We observe, nevertheless, that the introduction of artifact-evaluation badges has had a net positive effect (Figure 3.1). In venues such as EuroSys, IMC, and SIGCOMM, it has significantly improved the reproducibility of CCA studies. Shockingly, we note that the number of reproducible CCA work in CoNEXT has dwindled since the introduction of artifact badges. Perhaps this finding hints at the need for revisiting the criteria for publishing congestion control research and including additional incentives for improving reproducibility; program chairs could, for instance, only consider papers with public artifacts as eligible to be nominated for any awards.

[2]The percentage of publications with artifact URLs in text (column "In-text") subsumes those with the "Artifact ∗∗∗" badges.

3.2.2 Lack of Consensus

Even if we set aside the reproducibility concerns, the evaluation scenarios, conditions, and even metrics used in prior work differ vastly, often with little or no justifications on these choices. We reviewed the state-of-the-art work on CCAs and present the differences in the evaluation in Table 3.2. Per this table, there is hardly any consensus on the evaluation parameters. Even basic settings such as number of contending flows or the presence of small (or *mice*) flows greatly vary from one CCA evaluation to another. CCA designers employ diverse evaluation criteria for assessing their CCA's capability to co-exist harmoniously with flows using other CCAs. The criteria include varying RTTs, buffer sizes, and network topologies, as well as characterizing CCA using different metrics such as throughput, retransmission rate, Jain's fairness index (JFI), and queue usage.

The primary reason for this lack of consensus may be because different applications have different performance requirements, and no single scenario, condition, and metric can, hence, capture how well a CCA caters to the applications' needs. Furthermore, since an application's traffic (or flow) will encounter many other flows using different CCAs and experiencing vastly diverse network conditions, CCA designers may focus on metrics that explicitly measure a CCA's ability to coexist with other flows or handle the traffic (or workload) dynamics (e.g., JFI, *fness* [112], and *Harm* [27]).

While it is reasonable to optimize CCAs for specific applications or scenarios, CCA designers must not limit their evaluations to conditions that cast a positive light on their designs. We have the responsibility to highlight the costs or ramifications of our design choices: What did we trade off, and how? Such an open and rigorous evaluation necessitates that we reach a consensus on the objectives of such evaluations. While there are a few ongoing efforts on establishing a "standard" approach for benchmarking or evaluating CCAs [29, 116, 117], such efforts seem to dictate one or more benchmarks or evaluation suites (consisting of a diverse set of metrics). We are not belittling such efforts, but simply remarking that they might simply open up a new issue—dealing with numerous benchmarks or evaluation suites, which retain the current challenges in comparing CCAs with one another and offer no incentives for adoption. Any standardization effort should provide a framework to unify prior approaches while still providing a means to evolve the benchmarking approach. An open and standardized framework will help the community not only objectively evaluate their approach but also analyze its trade-offs and systematically identify opportunities to evolve their designs when deployment environment or network conditions change.

3.3 Case Studies

We examine the implications of the lack of consensus on CCA evaluations for congestion control research as well as practice. To this end, we examine some edge cases

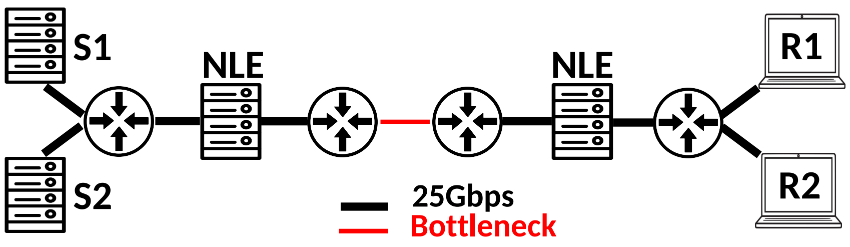


Figure 3.2: *A simple network testbed for evaluating CCA behavior.*

(in terms of network scenarios and conditions) that are critically important but often overlooked in CCA evaluations. We demonstrate how some state-of-the-art CCA designs perform poorly in these edge cases to highlight the implications the lack of consensus in CCA evaluation causes.

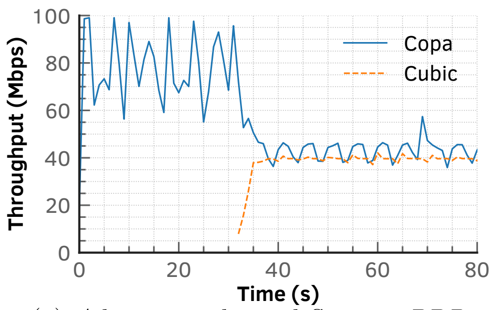
We investigated three modern CCAs deployed on the public Internet that reveal unexpected performance challenges along dimensions that the designers of those CCAs did not explicitly evaluate (i.e., in the respective publications), or did so in some limited capacity. We selected Copa [15] (used by Facebook [32]), PCC Vivace [92] (used by a leading US OTT streaming provider [143, 144]), and Sage [18] (a recently released learning-based CCA). Using a simple experimental setup, we highlight several new efficiency and fairness insights into these CCAs' behaviors, shedding light on aspects that merit further investigation.

3.3.1 Environment

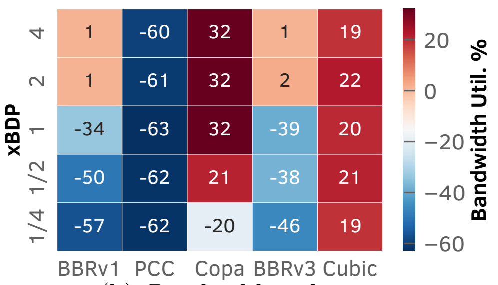
We used a testbed consisting of 10 Dell R6515 blade servers, each consisting of 16 cores and 128 GiB of RAM (Figure 3.2). We used Broadcom NetXtreme BCM5720 25 GbE NICs and 25 Gbps DACs for interconnecting the servers.[3] We used `netem` [145] on intermediate nodes which we designated as network latency emulator (NLE) between the end hosts, and `tc-tbf` [146] on the NLE to configure the bottleneck bandwidth and buffer size. The base RTT between sender-receiver pairs in the testbed was 1.5 ms on average. We ran Linux kernel version 5.4 (Debian 10) on the servers. The senders and receivers are KVM VMs running on the respective servers with Ubuntu 22.04.3 LTS (Copa and PCC), Debian GNU/Linux 12 (BBRv3), and Ubuntu 18.04.6 LTS with Linux kernel v4.19 (Sage—we could not get the suggested Ubuntu 16.04 to work with Sage, possibly due to software updates).

We set our bottleneck bandwidth to 100 Mbps, the link delay to 100 ms and varied the bottleneck queue as a function of BDP (i.e., $\{1/4,\ 1/2,\ \ldots,\ 16\} \times \text{BDP}$). We ran each experiment for 300 s with 3 repetitions. The parameter values we used in our tests merely represent a range of different scenarios that we chose to highlight *important* scenarios that were missed in the initial evaluations. Nevertheless, these

[3] We set our bottleneck to 100 Mbps hence lower speed NICs/DACs can be used



(a) *Adapting to limited flows, 1xBDP*



xBDP	BBRv1	PCC	Copa	BBRv3	Cubic
4	1	-60	32	1	19
2	1	-61	32	2	22
1	-34	-63	32	-39	20
1/2	-50	-62	21	-38	21
1/4	-57	-62	-20	-46	19

(b) *Bandwidth utilization*

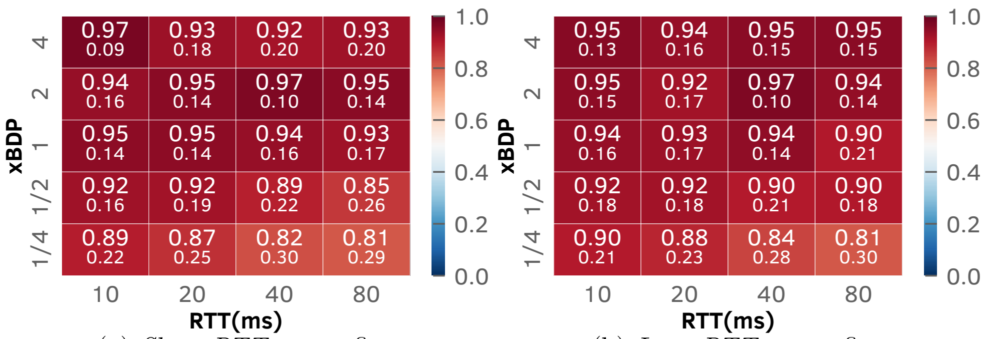
Figure 3.3: *Bandwidth utilization in inter-CCA scenarios.*

parameter choices were influenced in part by the prior work on standardizing CCA evaluations [29, 30] and prior CCA evaluations.

3.3.2 Copa's response to bandwidth fluctuations

In this case study, we analyze how Copa adapts to rapidly changing network conditions. One notable example of such variability in the Internet is the fluctuation in available bandwidth, which can occur due to dynamic routing changes or traffic demands imposed by different applications. Copa [15] uses "mode-switching" in practice, e.g., switching between its default behaviour and a 'TCP Competitive mode' when it observes a standing queue every 5 RTTs. This helps it to effectively compete with aggressive flows that may fill up the bottleneck queue. It is therefore critical to evaluate how effectively it manages fluctuations in available bandwidth, and to assess its ability to accurately detect and appropriately respond to such changes.

To this end, we start a long-lived Copa flow and let it saturate the 100 Mbps link for 32 s. We then start a competing rate-limited Cubic flow of 40 Mbps and run this for 50 s as shown in Fig. 3.3a. We calculate the median bandwidth utilization, $BwUtil$, of the Copa flow using simple performance gap analysis as follows: $BwUtil = (\frac{MaxBw - UtilBw}{MaxBw}) \times 100$, where $MaxBw$ is the maximum achievable bandwidth and $UtilBw$ the utilized bandwidth by the flow. In Figure 3.3b, positive numbers mean underutilization of flows' bandwidth share, and negative numbers the opposite. We observe that Copa almost always left a lot of bandwidth unused (up to 32%) when competing with a rate-limited Cubic, except at lower queue size configurations, where it was able to use more than it's share of bandwidth (20% at 1/4xBDP). This may be due to Cubic experiencing high losses at the low queue size. Copa's tendency to underutilize available bandwidth has been attributed to its overestimation of queuing delay in prior work [24, 147] or when faced with frequent RTT fluctuations [148] (i.e., in wireless or satellite networks). The results in this case study indicate that, despite Copa's mode-switching in order to co-exist with buffer-filling flows, it overestimates the queueing delay and penalizes itself, and ultimately leaves a portion of the link's bandwidth unused. Rate-limited flows are not uncommon in the Internet, particularly among "real-time" applications where throughput demands are typically finite, and



(a) *Short RTT starts first* (b) *Long RTT starts first*

Figure 3.4: *Normalized Queue utilization in multi-RTT scenarios.*

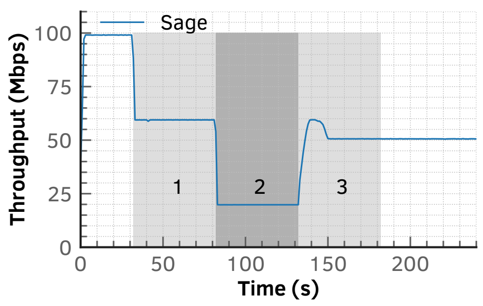
may even include periods of inactivity where no traffic is transmitted [149], thus evaluations should consider such scenarios.

3.3.3 PCC's ability to cope with dissimilar flows

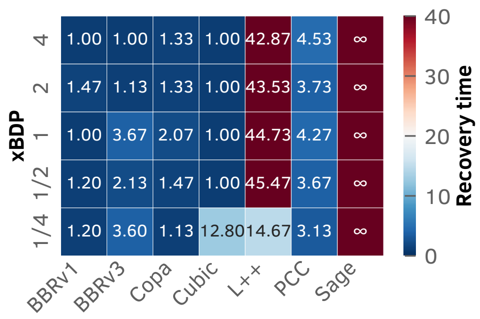
We investigate PCC Vivace [92], which is part of the PCC framework introduced in §2.3.5. We consider flows that experience different RTTs and arrive at the bottleneck at different times. Network paths in the Internet often comprise a diverse set of links: Transmission mediums, queue sizes, and processing delays are all factors that may cause flows to experience different RTTs. Although such RTT differences between flows can introduce unfairness, we observe that CCA evaluations typically focus on flows with equal RTTs (e.g., [26, 139]). Since CCAs rely on the implicit (e.g., packet loss or delay) or explicit (e.g., ECN) signals to detect congestion and react to it by reducing the sending rate; RTT differences essentially imply different reaction times for different senders. To quantify the impact of RTT differences, we run two PCC Vivace flows experiencing different RTTs. We configure one flow with an RTT of 160 ms, and vary the RTT of the other flow through various values–10, 20, 40, 80 ms–each constituting a separate experiment. We start the first flow and wait for 15 s before starting the second flow. In Fig. 3.4, we present the normalized queue usage when the long RTT flow (e.g., 160 ms) competes with a short RTT flow (shown on the x-axes). We observe significantly high queue occupancies across all queue configurations, which we see whether the short RTT or long RTT flow starts first, in Figures 3.4a and 3.4b, respectively. Given the concerted effort that has been invested in reducing bloated buffers in the Internet [48, 150], evaluations that highlight such performance caveats are indispensable.

3.3.4 Sage's adaptation to bandwidth changes

In this section, we investigate Sage's ability to cope with the unpredictability of bandwidth availability on a bottleneck link. Sage is an emerging, learning-based CCAs



(a) *Sage with dynamic bandwidth changes, 1×BDP*



(b) *Time (in s) to recover after Phase-3*

Figure 3.5: *Characterizing a CCA's ability to cope with dynamic bandwidth changes.*

that uses insights from prior CCAs to automatically develop better-performing congestion reaction and/or avoidance policies [18]. Flows on the Internet may experience rapid fluctuations in available bandwidth along their path due to as routing changes or transitions between different network types (e.g., from WAN to network edge mobile infrastructure). It is imperative, hence, to characterize how well a CCA can cope with such changes.

We start one long-lived (i.e., elephant) flow and allow it saturate the bottleneck (100 Mbps) for 32 s.[4] We then repeatedly change the bandwidth capacity of the link (as shown in Fig. 3.5a) and sustain each change for 50 s as follows. From $t = 32$ seconds to $t = 82$ seconds (*phase 1*), we change the capacity to 60 Mbps for 50 s; for the next 50 s (*phase 2*), we decrease it to 20 Mbps at $t = 82$, before finally increasing it again to 60 Mbps for another 50 s (*phase 3*). We effect these bandwidth changes or limitations using Linux's `tc` to change the bottleneck capacity dynamically in three phases (refer Fig. 3.5a). We run experiments for different CCAs and characterize how well each copes with the dynamic bandwidth changes along the path for comparison purposes. In Fig. 3.5b, we show the time taken by each CCA to reach 90% of the available throughput following the bandwidth reduction in *phase 3* (e.g., 100 Mbps). To this end, we determine the moving average of throughput using a 5-second window.

Most CCAs adapt quickly to the bandwidth change, typically within 2 seconds on average. LEDBAT++ struggles, however, to recover after phase 3: It requires about 40 seconds on average to adapt. In *phases 1 and 2* (omitted due to space constraints), we observe that Sage adapts well to bandwidth changes in a short period of time. It, nevertheless, completely fails to adapt to the available bandwidth after *phase 3* until the experiment ends, leaving the link severely underutilized.[5] In the Internet, bandwidth along a network path between two hosts can quickly change during the life of a flow. Such changes may even be typical in cellular or WiFi networks where one of the hosts could be a mobile end-user device. Whether a CCA can cope with such changes by quickly discovering available capacity and converging to it crucially

[4]We use 32 seconds to avoid interfering with BBR's RTT probing phases [35].
[5]Our evaluation used the 7-day pre-trained model provided in the Sage repository [18].

determines its suitability for use in the Internet. Our investigations, in this section, show that while many of the modern CCAs are able to cope with such dynamic bandwidth changes in a timely manner, a recent learning-based CCA still does not recover until the end of our experiment (300 s) despite it exploiting the wealth of insights from the rich body of prior work on CCAs.

3.4 End-host Configuration Effects

Evaluating CCAs is inherently hard due to the vast parameter space that must be considered (see §3.2). One often overlooked dimension is the configuration of the end-hosts used in the evaluations, which may significantly influence the behavior of a CCA and, consequently, the evaluation outcomes. Below, we share the lessons we learned during our efforts to reproduce prior work, emphasizing the importance of properly addressing these overlooked aspects.

TCP Segmentation Offloading (TSO) and Large Receive Offload (LRO) were designed to reduce the CPU's network processing load by offloading it to the NIC. Enabling TSO can, however, cause the NIC to send a burst of traffic [151], potentially filling up the bottleneck buffer and adversely affecting CCAs such as BBR or Copa, which heavily rely on accurate RTT estimations. Although TSO auto-sizing [90, 152] can mitigate traffic burstiness of high-bitrate flows, we emphasize that the choice of TSO has implications for inter-CCA performance comparisons. Similarly, the tick rate (`HZ` [153]) of end-host kernels influence the time-stamping accuracy for delay-based CCAs.

To achieve high throughput on end-hosts, both the sender and receiver must be configured with adequate buffers. These buffers are managed via the parameters `tcp_mem`, `tcp_rmem`, and `tcp_wmem`, and they have implications for maximum achievable throughput [154]. TCP's initial congestion window also has non-trivial performance implications: A large initial `cwnd` may introduce burstiness, whereas a small initial `cwnd` can unnecessarily extend flows' completion time [155, 156].

Many prior work do not discuss their choices for these parameters, even though they have non-trivial implications on the outcomes of a CCA evaluation [6].

3.5 Summary

In this chapter, we shed light on the widespread lack of consensus concerning CCA evaluations. Despite the existence of guidelines on evaluation scenarios and conditions, most CCA evaluations substantially diverge from them. While we observe a growing number of venues that explicitly encourage and accept papers aimed at reproducing prior research in so-called "Replicability and Reproducibility Tracks" [157,

[6] In our evaluations, we chose optimal settings obtained through empirical analysis. We selected values that demonstrated robust performance across different CCAs.

158], we believe program chairs can do more to incentivize publication of reproducible studies.

Additionally, we present several examples to illustrate how overlooking edge cases can obscure CCA performance insights. These findings, along with observations made from prior research, underscore the need for an open, rigorous, and reproducible evaluation framework for CCAs. In the next chapter, we address this challenge head-on by consolidating the diverse evaluation methodologies used in the last four decades into fundamental questions that distill the objectives of CCA benchmarking. This structured approach can facilitate the automation of CCA evaluations (e.g., by using recent advances in machine learning) and substantially lower the burden on researchers.

Table 3.2: *An overview of the diverse evaluation scenarios, conditions, and metrics used in prior work on CCAs.*

Ref.	#flows	mice?	stag.?	BW (Mbps)	RTT (ms)	buffer	topo.†	Plat.‡	Tput. - Gput.	RTT	JFI	cwnd.	Q usage-delay	Loss/Retx. rate	Others
[47]	2	✓	✓	1-400	8-324	1 BDP, 2 MB	$\mathcal{D}$	TB	✓	✗	✗	✗	✗	✗	Link util., intra-proto. fairness, RTT fairness
[35]	1,5,8	✗	✓	0.128, 10-100	10, 40, 100	0.15-10 MB	$\mathcal{D}$, $\mathcal{W}$, $\mathcal{C}$	—	✓	✓	✗	✗	✓	✓	—
[15]	1,3,10, 20	✗	✓	1-100	2–800	1 BDP	$\mathcal{D}$, $\mathcal{W}$, $\mathcal{DC}$, $\mathcal{S}$	TB, Sim.	✓	✓	✓	✗	✓	✓	RTT fairness, FCT
[92]	1,2,4	✗	—	10- 1000	10-800	7.5-30 KB	$\mathcal{D}$, $\mathcal{S}$, $\mathcal{C}$	Emu.	✓	✗	✗	✗	✓	✓	—
[49]	2,4,8,16	✓	—	1.6, 2.4, 3.2	0-102	14 KB	$\mathcal{D}$, $\mathcal{W}$	Sim.	✓	✗	✗	✓	✓	✓	TCP window
[51]	4,8	✗	✗	50- 700	30- 240	100- 6000 pkts.	$\mathcal{D}$	TB	✓	✗	✗	✓	✓	✓	Bw. stolen
[133]	2,18	✗	✓	100, 10000	5,21, 61,101	100 MB	$\mathcal{D}$	TB	✓	✓	✗	✓	✗	✗	—
[89]	4,8,12	✓	✓	4.7- 50	4,200	1000 pkts.	$\mathcal{D}$, $\mathcal{C}$, $\mathcal{DC}$	Sim.	✓	✓	✗	✗	✓	✗	seq# timeline, norm. tput.
[94]	1-10	✗	✓	3- 300	4-400	3 KB- 96 MB	$\mathcal{W}$, $\mathcal{C}$	TB Emu.	✓	✓	✗	✓	✗	✗	score, training time, avg. norm. tput., avg. norm. delay avg. CPU util., norm. power friendliness ratio
[128]	2,3,4	✗	✗	10, 100	0-400	—	$\mathcal{D}$	TB	✓	✓	✓	✗	✗	✗	intra-proto. fairness, inter-proto. fairness, RTT fairness
[134]	10-200	✓	✗	2,10	20-590	1 BDP 125 pkts.	$\mathcal{D}$, $\mathcal{W}$, $\mathcal{C}$, $\mathcal{S}$, $\mathcal{P}$	Sim.	✓	✗	✓	✓	✗	✓	#hops
[135]	2,3	✓	✓	1-500	16-200	0.01, 0.1, 1 BDP	$\mathcal{D}$	TB	✓	✓	✗	✗	✓	✗	fairness ratio, mean FCT
[136]	2	✓	✗	1-250	16-320	0.01, 0.1 0.2, 1 BDP	$\mathcal{D}$	TB	✓	✓	✗	✓	✗	✗	Fairness ratio, convergence time
[137]	2	✓	✓	400	16-324	2 MB	$\mathcal{D}$	TB	✗	✗	✓	✗	✗	✓	link util., Coeff. of var., convergence time
[56]	1,2,4,6	✗	✓	1000, 10000	5,20, 40,80	2,20, 200 MB	$\mathcal{D}$	TB	✓	✓	✗	✗	✗	✓	—
[138]	1-6,19	✗	✓	10	40,80	1- 14 BDP	$\mathcal{D}$	Emu.	✓	✓	✗	✗	✗	✓	Inflight, Bw.
[139]	2	✗	✗	10-1000	5,10,25, 50,75,100, 150,200	0.1,1,10, 20,50 MB	$\mathcal{D}$, $\star$	Emu. TB	✓	✗	✗	✗	✓	✓	Emu., TB
[140]	2,10, 20,100	✗	✗	100,1000, 5000	10,100	0.05-2 BDP	$\star$	TB	✓	✗	✓	✗	✗	✗	—
[112]	2	✓	✓	10	50, Internet	0.5, 2 BDP	$\mathcal{D}$, $\mathcal{W}$	TB	✗	✗	✓	✗	✓	✓	—
[141]	2-50, 1k-5k	✗	✓	100 10000	20,100, 200	1 BDP	$\mathcal{D}$	TB	✓	✗	✓	✓	✗	✓	—
[142]	1-30	✓	✓	100,300	50	1 BDP	$\mathcal{D}$	Emu.	✓	✓	✗	✗	✗	✗	link util.
[107]	2-900	✓	✓	100,800, 970,1000	0,50,220	—	$\mathcal{W}$, $\mathcal{P}$	TB	✓	✗	✗	✗	✗	✓	link util.

† $\mathcal{D}$: Dumbbell, $\mathcal{P}$: Parking lot, $\mathcal{C}$: Cellular, $\mathcal{S}$: Satellite, $\mathcal{W}$: WAN, $\mathcal{DC}$: Datecenter, $\star$: Star
‡ Testbed: TB, Emulator: Emu., Simulator: Sim.

4 A Recipe for Benchmarking Congestion Control Algorithms

There is increasing diversity in CCAs deployed in the Internet [19–21]. This heterogeneity is only likely to increase with the continued adoption of HTTP/3 [69, 70], which will, in turn enlarge the deployment footprint of QUIC. As the variety of CCAs in use continues to widen, the need for systematic and rigorous comparisons across designs becomes increasingly critical. Without such comparisons, it is difficult to ascertain where (i.e., in what aspects) a new CCA design excels and why. Before we deploy these innovative CCA schemes, it is imperative that we rigorously evaluate how they fare in network conditions representative of those experienced in the public Internet compared to existing schemes.

As we noted in Chapter 3, there is often little to no consensus in CCA evaluations in practice. Furthermore, there is a widespread lack of well-specified, open, and reproducible evaluations, which poses a critical threat for congestion control research. This chapter addresses this challenge in line with our third research objective by proposing a framework to enable systematic, rigorous, and comprehensive evaluations of CCAs. The framework focuses on benchmarking or evaluating CCAs designed specifically for the public Internet.

We raise the level of discourse on CCA benchmarking or evaluations to focus on the *purpose* of any such evaluation, i.e., the coverage of purposes of the tests we intend to run on a CCA. To this end, we introduce a structured CCA evaluation "recipe" that both *unifies* and *extends* existing methodologies. The recipe is grounded in identifying the fundamental questions that a CCA designer or reviewer expects any such evaluation to answer. We broaden these questions to capture the variability and complexity of network conditions in the public Internet. We distill the objectives of CCA evaluations into three key questions concerning a CCA's behavior:

1. *How does a CCA cope with the performance unpredictability of the end-to-end path?*
2. *How does a CCA adapt to cross-traffic?*
3. *How does a CCA handle workload dynamics?*

These three questions capture the key behaviors of a CCA that we strive to characterize and we demonstrate how answering these three questions suffices to obtain crucial insights into CCA behavior in a wide range of network conditions. We do not dictate metrics, or benchmarks, or scenarios for an evaluation, but our recipe demonstrates how to generate the *performance profile* of a CCA and, in the process, shows how evaluation choices affect the characterization of a CCA's behavior. We used the evaluation recipe to highlight issues in the latest BBR version, BBRv3 (see Chapter 5), and the community recognized the findings with an award [1]. These preliminary studies clearly demonstrate the utility of our CCA evaluation recipe.

While there has been prior efforts to achieve consensus within the networking community on CCA benchmarking, these efforts have largely been ignored by CCA designers as we previously discussed in §3.2. This stems, in part, from the fact that prior approaches do not always clearly articulate the underlying intent of CCA evaluations. Moreover, the extensive list of metrics and tests prescribed by these efforts make CCA evaluations time consuming and arduous. Our recipe can be integrated with existing tools, platforms, emulators, and simulators, thereby contributing to ongoing efforts towards the standardization of CCA evaluations. Furthermore, the recipe has the potential to inspire further innovation in automated platforms and benchmarking tools, enabling them to systematically address the common, fundamental questions that every evaluation should attempt to answer.

The contributions of this chapter are as follows.

- We present a benchmarking recipe by focusing on the objectives (or intents) of CCA evaluations, rather than the tests that comprise an evaluation. It distills, adapts, and expands decades of prior work on CCA evaluations into three succinct fundamental questions concerning a CCA's behavior. We motivate the rationale behind our recipe and demonstrate the utility of answering these questions.
- We benchmark several CCAs using our recipe, and we discover novel insights and reveal (previously unknown) shortcomings in their performance. Evaluating CCAs using our recipe offers designers quick insights into the shortcomings of CCA, allows operators to compare and contrast CCAs and, crucially, and understand the implications of introducing a new CCA in the public Internet. Our benchmarking recipe reveals, for instance, that when contending flows arrive at different times at a bottleneck link—a common scenario in the public Internet—PCC and BBRv1 converge to their fair shares quickly regardless of bottleneck buffer sizes. Had this recipe been available to the designers and operators, it might have informed the deployment plan of BBRv3. Our recipe also shows that Sage, a learning-based CCA, which by design learns the optimal behavior from decades of prior work, struggles to converge in the typical bottleneck buffer setting in the public Internet.

4.1 Related Work

Early attempts to formalize CCA evaluations include Sally Floyd's memo [29] and the 2014 Common TCP Evaluation Suite draft [30] (recently revised by [23]). However, a lack of consensus regarding evaluation approaches, has led many CCA designers to selectively choose evaluation scenarios and metrics. This cherry-picking often results from the impracticality of performing the extensive tests suggested in those memos.

Platforms such as Mahimahi [124] and the Pantheon [90] took the initial steps towards addressing the lack of reproducibility in CCA evaluations. These tools, alongside other platforms like TEACUP [97] and FLENT [98], facilitate more consistent and reproducible benchmarking efforts. Similarly, emulators and simulators such as NS-3 [95], OMNeT++ [122], and GloMoSim [123] complement these efforts and can be integrated with our CCA benchmarking recipe to help standardize evaluation practices. An orthogonal line of work focuses on reverse engineering and formal verification of CCA behavior, where proposals such as CC-Fuzz [126] and Abagnale [127] seek to identify where CCAs perform badly, and why. Nevertheless, the choice of evaluation scenarios and parameters often remain at the discretion of CCA designers, resulting in little or no overlap across evaluations.

The work in [128] provided initial comprehensive evaluations of CCAs, marking a significant step toward understanding CCA behavior across various dimensions. Other research has focused on CCA behavior concerning its ability to ensure fair bandwidth allocation for services on the Internet [159]. While earlier CCA evaluations, such as [47], incorporated recommendations from the Evaluation Suite draft, recent studies often lack comprehensive evaluations and often regurgitate commonly evaluated scenarios that often fail to catch CCA limitations such as those shown in [128].

The increasing adoption of HTTP/3 will further shift a significant portion of Internet traffic to be served over QUIC [69, 70]. This transition may introduce more diversity in Internet CCAs, since it makes it easy for designers to design, adapt and deploy userspace CCAs [22, 71]. Our CCA benchmarking recipe offers a valuable framework for evaluating the conformance of such userspace CCAs. Succinctly, we build upon the foundational efforts to provide high-level questions that seek to emphasize the intent behind CCA evaluations, and align with recent calls by the IETF to provide updated guidance for evaluating new CCAs [23].

4.2 Motivation & Insights

There is no clear agreement within the networking community on what constitutes a rigorous benchmark of a CCA [5, 29]. Our survey of evaluation parameters used in some of the well-known CCA schemes (refer Table 3.2) reveals little overlap, if any. This lack of agreement makes it hard for us to compare a CCA against others, which is crucial for both gaining confidence in a specific design or refining it during development. We cannot determine if one CCA scheme fares better than some others because of the choice of metrics used for evaluation, or assumptions made about the operating environment, or a combination of such factors. Deployment conditions may change, and operating environments (e.g., infrastructure) typically evolve over time; such changes make it infeasible for us to ascertain what aspects of a CCA's behavior needs to be re-evaluated, and how. In the absence of a common "recipe" for evaluating CCAs, we cannot reliably characterize the interaction dynamics between CCAs, which

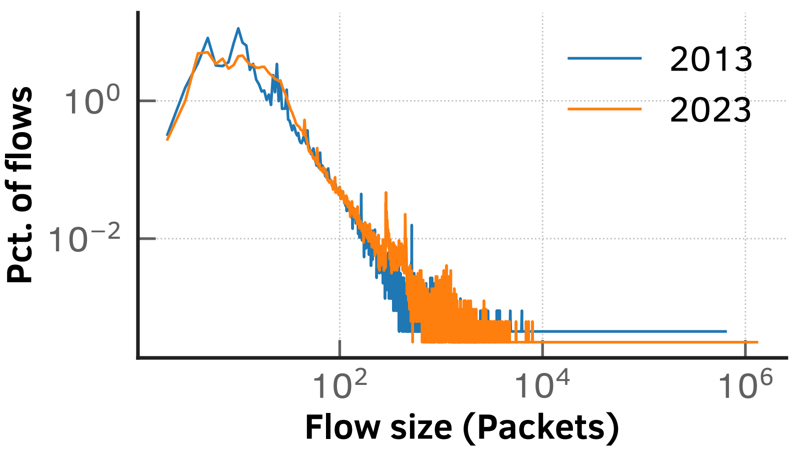


Figure 4.1: *In the public Internet, flow sizes vary substantially, exhibiting heavy tails.*

matter particularly when deploying a CCA in the public Internet (e.g., [1, 25]). Lastly, differences in evaluations undermine our confidence in prior work and constrain our ability to leverage insights from them.

Rather than prescribe a list of tests that everyone must run, on which obtaining a consensus has proven elusive, we shift our focus toward the underlying purpose of these evaluations. Our approach aims to nudge the community toward reaching a consensus on CCA evaluations by first focusing on *intent* of any such evaluations. To this end, we reviewed decades of prior work on CCAs and analyzed how they were evaluated. We then clustered the various tests or experiments into common themes or dimensions, and identified the high-level questions, concerning the performance or behavior of a CCA, that the tests in each cluster are attempting to answer. We believe these questions encapsulate *critical* CCA evaluation objectives, and posit that they serve as a foundation for advancing the networking community towards standardized CCA evaluations. In the remainder of this section, we present these questions and discuss the insights that informed their development.

① ***How does a CCA cope with the performance unpredictability of the end-to-end path?***

In the Internet, flows may arrive at a bottleneck link via paths that differ vastly in their performance characteristics (e.g., delays and loss). Paths between two arbitrary hosts in the Internet may traverse through datacenters, Internet service providers (ISPs), and Internet exchange points (IXPs); the performance and configuration of network elements (e.g., buffer sizes [160–162] and AQMs) in (independent) networks along these paths may differ substantially. The differences in bottleneck buffer sizes [35, 47, 91], delays or RTTs [15, 89], and network topologies [15, 94] used for evaluating CCAs (Table 3.2) indeed reflect the differences in the deployment environments (e.g., data centers and WAN). Most importantly, even if a CCA is optimized for a deployment environment, that environment will likely evolve or change over time; that CCA may also be deployed to a different target environment. Either case has substantial implications for the behavior of that CCA.

② *How does a CCA adapt to cross traffic?*

The cross-traffic encountered by a flow along the path between a source and destination may lead to bandwidth fluctuations over time due to several factors: (a) Variations in the arrival times of contending flows at the bottleneck, or interactions with flows experiencing different RTTs (due to, for instance, routing changes and outages [162, 163]); (b) The traffic generated by different applications that share a path with any given flow will vary over time depending on the application type; web-browsing applications, for instance, tend to be bursty and latency sensitive in nature, while video streaming tends to be latency sensitive and bandwidth intensive; and (c) The flows contending on a bottleneck link will likely be using different CCAs, each of which may react differently to latency and bandwidth changes along the path; indeed whether the CCA of a flow is in steady state or not has implications for its ability to both infer and react to changes in available bandwidth. A CCA cannot be evaluated, hence, in *isolation*, for instance, using a specific type of application, or in competition with flows using the same or similar CCAs. A CCA's compatibility with other CCAs, in particular, crucially determines its feasibility for use in the public Internet. Additionally, flows may traverse different hops along their path, where they contend with varying sets of competing flows—a scenario that the simple, widely used dumbbell topology cannot model. The "parking lot" topology [102, 103, 109], in contrast, presents a setup where flows traverse through several contention points that arise at the links where they contend for bandwidth with cross-traffic.

③ *How does a CCA handle workload dynamics?*

Different traffic workloads subject a network (specifically, the bottleneck link) to different kinds or levels of load [113]. CCA evaluations that use long-running (or "elephant") flows (e.g., [26, 90, 128]) typically focus on the steady-state behavior of the CCA. Since elephant flows contribute most of the traffic volume in the Internet [164, 165], these evaluations typically reveal the performance experienced by throughput intensive applications. The MAWI data set, which constitutes network traffic data sampled from a transit link [67], reveals that flow sizes vary a lot and exhibit a heavy-tailed distribution (Fig. 4.1). The similarities between the flow-size distributions sampled ten years apart, first in 2013 and then in 2023, shows that the heavy-tailed nature of flow sizes is consistent over time. A significant fraction of them are short-running (or *mice*) flows, which have fewer than 10 to 12 packets; such flows terminate well before they progress to the (typical) congestion avoidance phase. It is infeasible to congestion control such mice flows, and they can saturate a network link at any time [113]. Workloads that capture the properties of diverse traffic conditions are, hence, crucial for evaluating the ability of a CCA to cope with the highly dynamic nature of traffic in a network. Especially in the Internet, where we do not have much control over flow arrivals or types or sizes, such evaluations could be critical for avoiding the introduction of a new CCA wreaking havoc on existing traffic in the Internet [114].

4.3 Using the CCA Benchmarking Recipe: The Plan

Our recipe for benchmarking a CCA is rather simple and straightforward: Any rigorous benchmarking approach *must* answer all three fundamental questions concerning its behavior (discussed in §4.2). While the recipe mandates the objectives, it allows the community to debate the choice of metrics or experiments for answering each of these questions. Essentially, the recipe is a *meta* benchmark that unifies all benchmarking approaches with respect to the objectives that they must satisfy.

The application of this benchmarking recipe to any CCA results in a practical and rigorous *performance profile* of that CCA that can be readily compared to a performance profile of any other CCA. To elucidate how to design a benchmarking approach following this recipe, we offer simple and practical guidelines on evaluation scenarios that answer all three questions. First, for a robust characterization of a CCA's ability to cope with the performance unpredictability of the end-to-end path, we recommend that evaluations experiment with both dumbbell and parking-lot topologies. Evaluations can vary the delays, bandwidths, and bottleneck buffer sizes in such topologies to cover a range of scenarios observed in prior work, even if this range extends beyond the ideal conditions for a specific CCA design.

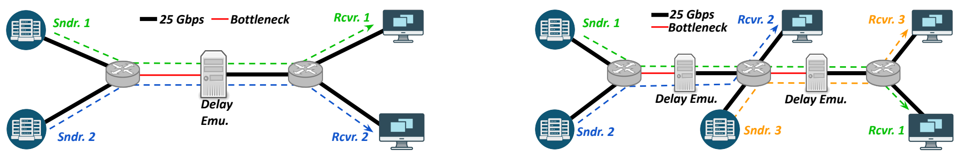
Second, a CCA's core objective is to quickly discover the available bandwidth at a bottleneck link. Since bandwidth changes at a bottleneck may be effected by both elastic factors (e.g., competing flows using same or different CCAs) and non-elastic factors (e.g., static topology changes, or short-lived flows that cannot be congestion controlled), we recommend evaluations to account for both these factors. Evaluations should also use competing flows with different CCAs that react differently to various implicit (e.g., loss and delay) as well as explicit (e.g., ECN) congestion signals for understanding a CCA's compatibility with other CCAs.

Lastly, we recommend using workloads generated by sampling real network traffic to characterize a CCA's performance when competing with flows of different sizes and assess its ability to cope with dynamic traffic changes (i.e., due to dynamic flow arrivals and departures at a bottleneck link). Evaluations should not skip or skimp on answering these questions by arguing that the evaluation scenarios are not applicable or relevant for a concerned CCA.

4.4 Using the CCA Benchmarking Recipe: The Prep

Below, we describe a minimal network infrastructure for using our benchmarking recipe and evaluating any CCA in a range of scenarios. Later, we use this setup for benchmarking various CCAs using our recipe and present our findings.

Testbed To characterize the behavior of a CCA using our benchmarking recipe, we designed a custom testbed consisting of eight Dell R6515 blade servers and four APS Networks BF6064X-T switches. Each server had 16 cores, 128 GiB of RAM, and one or two Broadcom NetXtreme BCM5720 25 GbE NICs, as required. We ran Linux



(a) *Single bottleneck* (b) *Multiple bottlenecks*

Figure 4.2: *Network testbed environments for evaluating the performance of CCAs in both single-bottleneck and multi-bottleneck scenarios.*

(Debian 10) with kernel version 5.4 on all the servers. We used 25 Gbps DACs for interconnecting the servers and four switches. The servers functioned as either end hosts or traffic shapers, while the switches only forwarded the traffic. To emulate a diverse range of network conditions in both single and multiple bottleneck scenarios, illustrated in Fig. 4.2a and Fig. 4.2b as described in [30], we used the traffic control utility `tc-tbf` [146] and `netem` [145] to filter, throttle, and redirect traffic.

Configurations We realized the single and multiple bottleneck scenarios, illustrated in Fig. 4.2a and Fig. 4.2b, using Linux's traffic control (`tc`) utility. More concretely, we used `tc` to filter, throttle, and redirect traffic and emulated a wide range of network conditions. We also varied the capacities of bottleneck links using `tc` as required for different experiments. We designated an intermediate node between the end hosts as the network latency emulator (NLE) to avoid issues with TCP small queues [166, 167] and used `netem` [145] for introducing one-way delays. Additionally, we configured `tc-tbf` [146] on the NLE to configure the bottleneck bandwidth and buffer size. Unless otherwise stated, we added half of the configured delay to the link before the bottleneck and half to the one after the bottleneck. The base RTT between sender-receiver pairs in our testbed, without any added delay, was 1.5 ms on average.

Choice of CCAs for validation We used 7 different CCAs for our evaluations: Cubic, BBRv1 and v3 [62], Copa [15], PCC [92], LEDBAT++ [168], and Sage [18]. We deployed VMs on each of the servers and installed each CCA on a separate VM. This setup allowed us to deploy a CCA without affecting the installation of another on the same (physical) machine (and due to limited hardware). BBR is widely deployed in Google's infrastructure [62], and it is already used for delivering a significant amount of Internet traffic [19]. Though BBRv1 and BBRv3 are two versions of the same CCA, they differ substantially in design and behavior. We also chose these versions to demonstrate how the recipe can easily highlight shortcomings in their designers' (publicly presented) evaluations [58, 62, 169]. We ran BBRv3 on a VM running Debian 12 with Linux kernel v6.4.0, which is the version provided by Google [170]. Copa and PCC are both user-space implementations; we only have to install the necessary binaries to use them. We, nevertheless, opted to use VMs even for these two CCAs to make it easy to control the sending rates of (individual) flows, as required in some experiments. The Copa and PCC VMs both used Ubuntu 22.04.3 LTS with Linux kernel v5.15, while the Sage VMs used Ubuntu 18.04.6 LTS with Linux kernel v4.19 (we could not get the suggested Ubuntu 16.04 to work with Sage, possibly due to software updates). For PCC, the (default) MSS of 1472 bytes introduced fragmentation issues in our testbed. We modified the PCC implementation, hence, to reduce the MSS value to 1448 bytes and circumvent the fragmentation issue. Similar to Copa and PCC, we used Ubuntu 22.04.3 with a patch [171] for the LEDBAT++ VM (denoted as L++ in all figures for brevity).

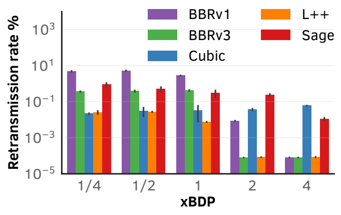
Workloads We used several different workloads for characterizing the performance of CCAs. In line with prior work, we used iPerf2/iPerf3 [172] for generating long-lived (i.e., "elephant") flows. Specifically, we used iPerf2 and iPerf3 [172] for generating long-lived (i.e., "elephant") flows for Cubic, BBRv1, BBRv3, LEDBAT++ and Sage. For Copa and PCC, we used the respective binaries for traffic generation. These persistently backlogged flows help in analyzing the steady-state behavior of a CCA. We mixed these elephant flows with short-lived (i.e., "mouse") flows, represented by a `SYN`-flood traffic, to create synthetic workloads. Basically, we generated a `SYN` flood by repeatedly opening and tearing down TCP connections, immediately after the 3-way handshake completes, using the **`hping`** (v3) utility. To generate a substantial volume of traffic with the SYN flood, we configured the utility to use 400 bytes of per-packet payload. For emulating real-world traffic conditions, we generated a workload by sampling flow sizes from a MAWI trace [67] generated on January 5, 2023. We sampled the flow sizes and connection times (of traffic associated with well-known port numbers) from this trace using Zeek [173]. We then scaled up the interconnection times by a factor of 5 to adapt the workload to our bottleneck, which has less capacity than the MAWI infrastructure. We then provided these samples as input to the Harpoon workload generator [174]. We created three workloads by configuring Harpoon to randomize the selection of flow sizes and inter-connection times. These three workloads help us evaluate whether the CCA behavior is affected by the randomness in the workload. The parameter values used in our experiments do *not* represent an exhaustive list of network scenarios, but capture the range of different scenarios influenced in part by the prior work on standardizing CCA evaluations [29, 30] and in part by evaluations presented in prior work on CCAs. Unless otherwise stated, we ran each experiment 3 times and report the statistics.

4.5 Using the CCA Benchmarking Recipe: The Steps

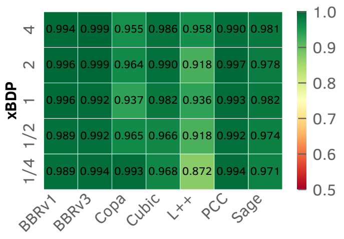
In this section, we present a simple approach for answering the three key questions posed by our recipe and discuss new efficiency and fairness insights into different CCAs.

4.5.1 How does a CCA cope with the performance unpredictability of the end-to-end path?

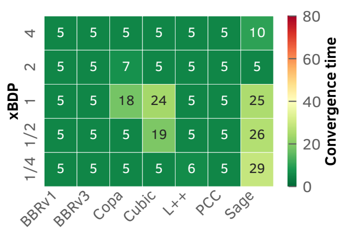
In this section, we discuss how to analyze a CCA's ability to cope with the performance unpredictability of the end-to-end path (§4.2). We present a twofold approach that exercises a CCA in a wide range of real-world conditions and characterizes its



(a) *Retransmission rate*



(b) *Avg. Jain's fairness index*



(c) *Avg. convergence times (s)*

Figure 4.3: *Bottleneck buffer size has (a) implications for the number of retransmissions a CCA's flow may experience, but (b) does not hinder the CCA from obtaining its fair share of capacity. However, convergence time, (c), varies per CCA.*

behavior using metrics commonly used in prior work. As an initial step, we focus specifically on analyzing the behavior of two long-lived flows using the same CCA in two dimensions: paths with varying buffer sizes and those with rigid, instantaneous bandwidth changes. At each step, we highlight the key takeaways and insights from our approach.

4.5.1.1 Buffer size

It is quite challenging to determine the optimal size of bottleneck buffers [160, 161, 175–180]. Buffer sizes are, hence, likely to differ between links, depending on the link type (e.g., inter-AS or intra-AS), network where they are located (e.g., eyeball or core), network operator's policies, and other factors [162]. To characterize the effect of buffer sizes on a CCA's performance, we run two long-lived flows in a single-bottleneck (i.e., dumbbell topology) setting with 100 ms RTT and a bottleneck capacity of 100 Mbps.

Loss-based CCAs such as Cubic experience only a marginal effect, if any, to changes in buffer sizes (Fig. 4.3a): There is little to no change in the (Cubic) sender's retransmission rates. Delay-based CCAs such as BBR, in contrast, seem highly sensitive to bottleneck buffer sizes. BBRv1 performs poorly compared to Cubic, Sage and BBRv3, especially at shallow buffers. Although BBRv3 shows lower retransmissions than BBRv1, they are overall still higher than what is seen in loss-based algorithms, i.e, Cubic, pointing to BBR's inefficiencies when attempting to keep queue occupancies low, especially at shallow buffer scenarios, as can be observed in Fig. 4.3a. The performance difference between BBRv1 and BBRv3 stems from the fact that the former is agnostic to losses, while the latter takes them into account for adjusting the sender's rate [62]. Our recipe is able to pinpoint clearly that such inefficiencies are due to BBR overshooting when probing for bandwidth, and this issue is more exacerbated in shallow buffer settings. LEDBAT++ is designed specifically for "delay-insensitive", background traffic [52], and backs off when it observes high delays. This behaviour seemingly helps it to avoid retransmissions, leading to much better performance than either BBR across all buffer configurations.

Our recipe also facilitates analysis of emerging, learning-based CCAs, such as Sage [18]. Sage uses a data-driven approach to learn from existing CCAs and automatically develop better-performing CCA policies. We show that Sage performs poorly with respect to mitigating retransmissions; it performs as poorly as the BBR variants.[1] Instead of providing an overall ranking of a CCA such as in [18], our approach focuses on identifying specific network scenarios and providing an "apples-to-apples" comparison between CCAs. We omit Copa and PCC from this plot, nevertheless, since they are both implemented in user space on top of UDP, making it difficult to calculate retransmissions accurately.

To further analyze CCA behavior, we plot the Jain's fairness index (JFI) for all the CCAs as a function of buffer size in Fig. 4.3b). With the exception of LEDBAT++, buffer-size changes do not seem to hinder a CCA from obtaining its fair share of the bottleneck capacity. Additionally, our recipe highlights the danger in prioritizing one metric over another which can obscure the underlying behavior of a CCA. While average JFI and convergence values (e.g., Fig. 4.3c) seem acceptable (except for Sage, which demonstrates poor convergence at shallow buffer levels), assessing CCAs through the lens of retransmission rates can be particularly informative for delay-sensitive applications. Following the recipe allows designers to have a broad perspective of a CCA's behavior.

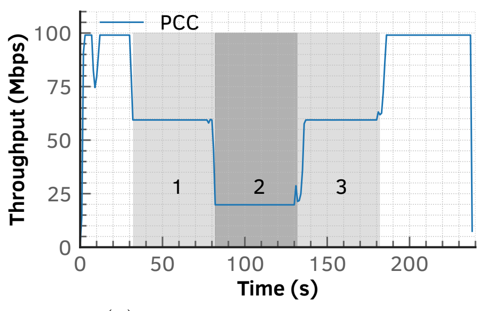
Takeaways. *It is impractical to assume that buffers of routers along a path would all be configured in a consistent manner. While changes in the bottleneck buffer sizes do not substantially alter the fairness properties of most CCAs, it has a pronounced impact on their convergence times (to their fair shares) and retransmissions. We can use our recipe to also pinpoint the shortcomings in BBRv3's design, and perform head-to-head comparisons of its convergence time with those of LEDBAT++, PCC, Sage, and others.*

4.5.1.2 Instantaneous bandwidth changes

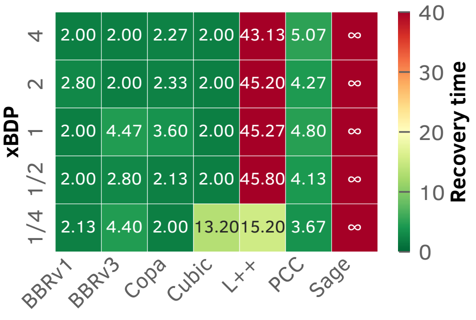
Flows on the Internet may experience rapid fluctuations in available bandwidth along their path due to factors such as routing changes or transitions between different network types (e.g., from WAN to network edge mobile infrastructure). It is imperative, hence, to characterize how well a CCA can cope with such changes. To this end, we start one long-lived (i.e., elephant) flow and allow it saturate the bottleneck (100 Mbps) for 32 s.[2] We then repeatedly change the bandwidth share of the flow and sustain each change for 50 s as follows. We effect these bandwidth changes or limitations using Linux's `tc` to change the bottleneck capacity dynamically in three phases. From $t = 32$ seconds to $t = 82$ seconds (*phase 1*), we change the capacity to 60 Mbps for 50 s; for the next 50 s (*phase 2*), we decrease it to 20 Mbps at $t = 82$, before finally increasing it again to 60 Mbps for another 50 s (*phase 3*) (e.g., Fig. 4.4a).

[1]Our evaluation used the 7-day pre-trained model provided in the Sage repository [18]

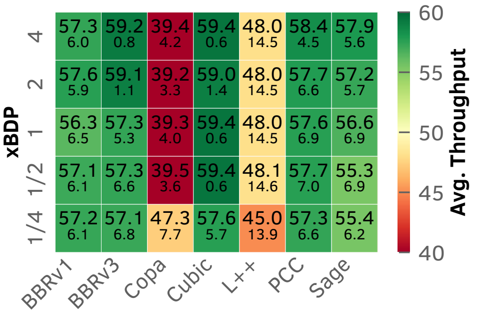
[2]We use 32 seconds to avoid interfering with BBR's RTT probing phases [35].



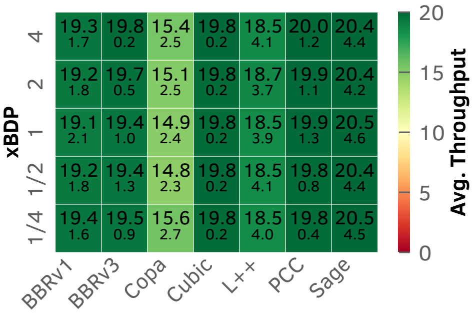
(a) *Changing bw using* `tc`



(b) *Recovery time after Phase 3*



(c) *Avg. throughput in phases 1, 3*

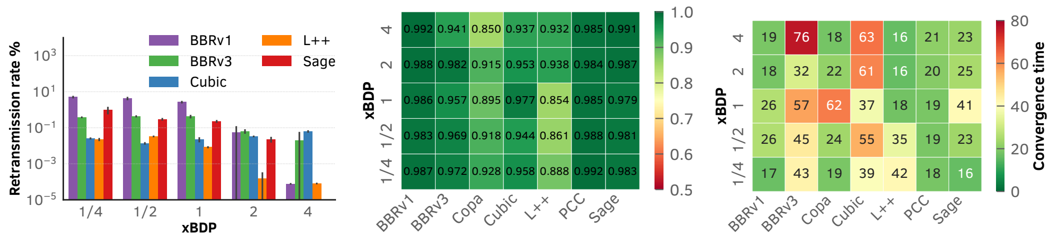


(d) *Avg. throughput in phase 2*

Figure 4.4: *CCA behavior under strict, dynamic bandwidth changes. Maximum throughput is set to* 60 Mbps *in phases 1 and 3, and* 20 Mbps *in phase 2.*

We run experiments for the different CCAs and characterize how well each copes with dynamic bandwidth changes along the path.

In Fig. 4.4b, we show the time taken by each CCA to reach 90% of the available throughput following the bandwidth reduction in *phase 3* (e.g., 100 Mbps). This is determined by calculating the moving average of throughput using a 10-second window. Most CCAs adapt quickly to the bandwidth change, typically within 2 seconds on average. However, LEDBAT++ struggles, requiring about 40 seconds on average to adapt. While Sage adapts well to bandwidth changes in *phases 1 and 2*, it fails to adapt to the available bandwidth after *phase 3* until the experiment ends, leaving the link severely underutilized. Similarly, and despite it's fast adaptation, Copa also exhibits substantial link underutilization, as shown in Figures 4.4c and 4.4d. For *phases 1 and 2*, where the bottleneck capacity was set to 60 Mbps, we observe that Copa struggles to fully utilize the available bandwidth, achieving about 33% less throughput than expected (25% less for *phase 3*). This behavior may be attributed to Copa's tendency to overestimate queuing delay, leading to self-penalization, as previously reported [24]. Copa's inability to converge to the correct bandwidth changes and corresponding underutilization warrants attention from those considering it for public deployment, as such bandwidth fluctuations are common on the Internet. For the BBR variants, we observe quick convergence to the bandwidth changes induced



(a) *Retransmission rate* (b) *Avg. Jain's fairness index* (c) *Avg. convergence time (s)*

Figure 4.5: *Staggering the arrival of flows at a bottleneck link by as little as 15 s significantly alters the behavior of a CCA.*

by `tc` in each of the three different phases; and they consistently achieve throughput that matches the limit induced by `tc`.

Takeaways. *In the Internet, bandwidth along a network path between two hosts can quickly change during the life of a flow. Such changes may even be typical in cellular or WiFi networks where one of the hosts could be a mobile end-user device. Whether a CCA can cope with such changes by quickly discovering available capacity and converging to it crucially determines its suitability for use in the Internet. Our investigations are able to show which of the modern CCAs are able to cope with such dynamic bandwidth changes.*

4.5.2 How does a CCA adapt to cross traffic?

Even if buffer sizes, bottleneck capacity, and (path) RTT remain constant throughout a flow's lifetime, the presence of dynamic cross-traffic can significantly influence the performance of a CCA. Scenarios such as flows arriving at the bottleneck at different times, contending flows with different RTTs, flows carrying traffic generated by diverse applications, and flows using different CCAs are a *norm* in the Internet. Investigating such scenarios is crucial to gaining valuable insights into the performance of a CCA under realistic conditions, and the second question in our recipe is designed to cover these investigations. Answering this question involves running experiments designed to characterize a CCA's behavior in the presence of various types of cross-traffic.

4.5.2.1 Staggered flows

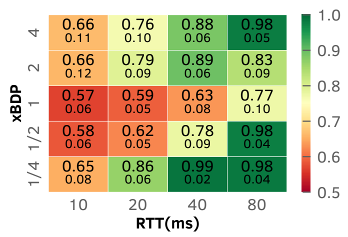
The evaluation of a CCA under the simplistic condition that all flows start at the same time is unrealistic in the Internet. Instead we *stagger* the flow arrivals by a small time period (i.e., 15 s) to characterize how the arrival of a flow affects existing flows in the network. We then calculate retransmissions, the JFI, and the moving average of JFI over 5 s windows to report the times it took the flows using the same CCA, to reach a fairness threshold of 0.9 (which we assume to represent equitable sharing). The level of retransmissions observed by CCAs at shallow buffers is consistent whether flows start simultaneously (§4.5.1) or at staggered intervals. However, at large buffers, the BBR variants exhibit an increase in retransmissions (Fig. 4.5a), particularly BBRv3. This aligns with prior findings that BBRv3 struggles to converge to a fair throughput share in large-buffer scenarios when flows are staggered [1], resulting in the high retransmissions observed here. Furthermore, BBR's convergence times at deep buffers are worse than those of Cubic, showing overall longer convergence times compared to both BBRv1 and PCC. While BBRv1 demonstrated relatively better convergence times, BBRv3 seems to have regressed in this aspect. If we focus our attention to Fig. 4.5c, we observe that all CCAs struggle substantially to achieve the fairness threshold, compared to when flows start simultaneously (Fig. 4.3c), which is also reflected in the slightly decreased fairness shown in Fig. 4.5b.

Takeaways. Simply staggering the arrival of flows at a bottleneck link substantially changes the behavior of a CCA. This observation has crucial implications for deploying a CCA in the public Internet, where such staggered arrivals will be typical. In such condiutions, BBRv3's long convergence times esp. in large buffers may be a cause for concern that warrants further investigation, given the existence of deep buffers in the public Internet [160–162], and its ongoing adoption of BBR across the public Internet [19, 21, 131]. More importantly, our recipe shows that CCAs struggle to converge quickly when flow arrivals are staggered, a scenario that is often ignored in CCA evaluations. Our recipe clearly indicates that if such conditions are typical, PCC fares well compared to the rest regardless of bottleneck buffer sizes.

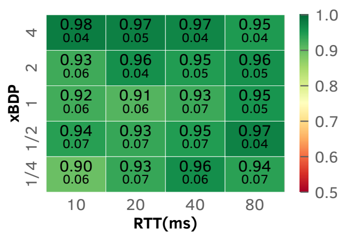
4.5.2.2 Dissimilar flows (with different RTTs)

Network paths in the Internet often comprise a diverse set of links: Factors including transmission mediums, queue sizes, and processing delays may cause flows to experience different RTTs. Although such RTT differences between flows can introduce unfairness, CCA evaluations typically focus on flows with equal RTTs (e.g., [26, 139]). CCAs rely on the implicit (e.g., packet loss or delay) or explicit (e.g., ECN) signals to detect congestion and react to it by reducing the sending rate; RTT differences essentially imply different reaction times for different senders. To quantify the impact of RTT differences across flows on a CCA's behavior, We run two flows using the same CCA but experiencing different RTTs. We configure one flow with an RTT of 160 ms, and vary the RTT of the other flow through various values—10, 20, 40, 80 ms—each constituting a separate experiment. We start the first flow and wait for 15 s before starting the second flow.

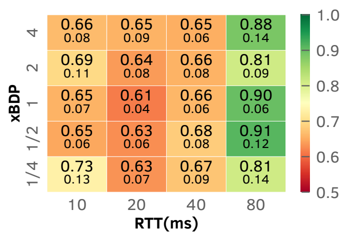
Cubic, PCC, and LEDBAT++ flows achieve high fairness, whereas BBRv3 and Copa perform poorly (Figs. 4.6 and 4.8). Sage only exhibits low fairness when the competing flow's RTT is much lower (e.g., 10 ms). While Cubic demonstrates overall good fairness, it is at the cost of higher queue occupancies compared to Copa, LEDBAT++ and Sage, regardless of bottleneck buffer size (Fig. 4.7). Like Cubic, PCC, a learning-based CCA, shows high queue occupancies (generally exceeding those of Cubic). This behavior results in a significant variation in fairness, especially in large-buffer sce-



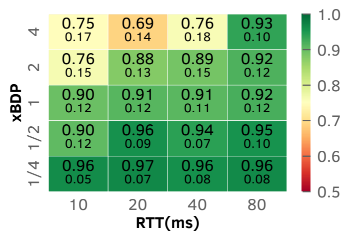
(a) *BBRv3*



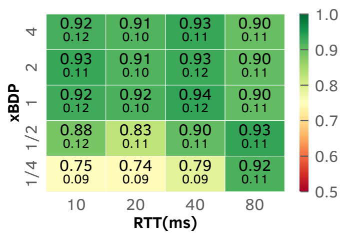
(b) *Cubic*



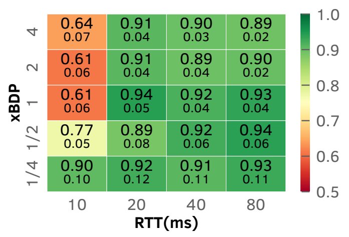
(c) *Copa*



(d) *PCC*



(e) *LEDBAT++*



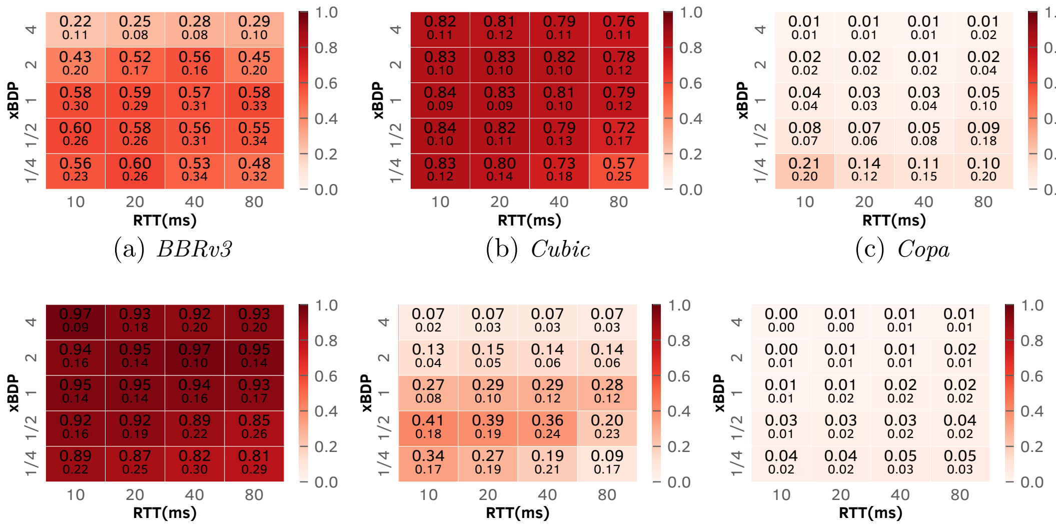
(f) *Sage*

Figure 4.6: *Contention between two flows with different RTTs when the short RTT flow starts first. The competing flow has 160 ms RTT, while the RTT of the first flow is varied through four different values (indicated along the X-axis). The second flow starts 15 s after the start of the first. Figures a-d shows the mean Jain's fairness index values and the bottom value in a smaller font represents the standard deviation.*

narios (top two rows), likely due to PCC's tendency to fill the queue (Figs. 4.7d and 4.9d).

Compared to other delay-based CCAs such as Copa and LEDBAT++, BBRv3 shows high queue occupancies. The BBR variants' general RTT unfairness is well-documented in prior work, especially when the gap between the competing flows' RTTs is large [1]. This unfairness stems from the longer-RTT flow inadvertently filling the queue due to BBR's 2xBDP-based in-flight strategy, leading to Head-of-Line (HOL) blocking for shorter-RTT flows [35, 56]. Nevertheless, BBR generally maintains lower queue usage compared to loss-based CCAs, i.e., Cubic. However, despite low queue occupancy being one of BBR's primary design objectives [35], our recipe highlights that in multi-RTT scenarios, BBR exhibits higher queue usage compared to modern delay-based CCAs such as Copa and LEDBAT++, particularly in shallow-buffer environments (Figs. 4.7 and 4.9). Overall, Copa, LEDBAT++ and Sage show the lowest queue utilization among the tested CCAs, demonstrating efficiency in queue usage, as shown in Figures 4.7c and 4.9c.

However, for Copa, as shown in Figures 4.6c and 4.8c, avoiding queueing delays leads to RTT unfairness. Our recipe highlights Copa's struggles in multi-RTT scenarios, with its performance only improving when the gap between the RTTs is relatively small (e.g., 80ms vs 160ms). Copa operates under the assumption of being able to observe an empty queue every 5 RTTs to establish a base delay, which it uses to calculate its sending rate. This mechanism enables it to effectively compete with



(a) *BBRv3* (b) *Cubic* (c) *Copa*

(d) *PCC* (e) *LEDBAT++* (f) *Sage*

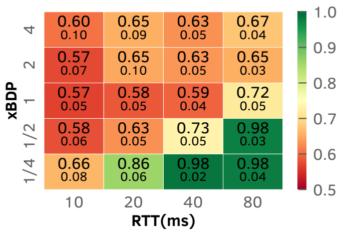
Figure 4.7: *Queue occupancy for two flows for when the short RTT flow starts first. Figures a-d show the mean normalized queue occupancy and the bottom value in a smaller font represents the standard deviation.*

buffer-filling, loss-based CCAs. However, in multi-RTT scenarios with significant RTT gaps between competing flows, this assumption can break down due to one flow filling the queue. Consequently, Copa may incorrectly switch to its ‘TCP competitive mode’, even when competing against other Copa flows [15], resulting in the RTT unfairness observed here.

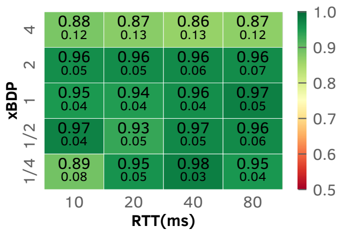
Takeaways. *Any reasonable evaluation of a CCA must consider flows with different RTTs, especially to characterize how the CCA might fare in the public Internet. The above experiments following our recipe reveal, for instance, that RTT differences substantially affect the fairness achieved by CCAs.*

4.5.2.3 Diverse traffic types

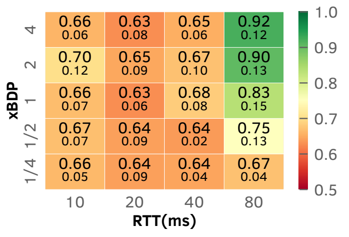
Network conditions (e.g., bandwidth) in the Internet may change rapidly and dramatically. How a CCA responds to such changes is a key measure of its performance in the public Internet, and should be thoroughly evaluated. Building upon the ‘bandwidth changes’ experiment described in §4.5.1, we start one long-lived (i.e., elephant) flow (say, “primary”) and allow it saturate the bottleneck (100 Mbps) for 32 s. As described earlier (Fig. 4.4a), the bandwidth allocation for the primary flow is adjusted across three phases, each lasting 50 s. In *phase 1* (from $t = 32$ to $t = 82$ seconds), we restrict the elephant flow to 60 Mbps for 50 s; *phase 2* (from $t = 82$ to $t = 132$ seconds), the elephant flow’s bandwidth share is further reduced to 20 Mbps for the next 50 s, and finally, in *phase 3* (from $t = 132$ to $t = 182$ seconds), the bandwidth share is restored to 60 Mbps for another 50 s. Following *phase 3*, the bandwidth restrictions are lifted,



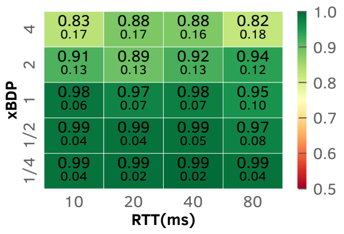
(a) *BBRv3*



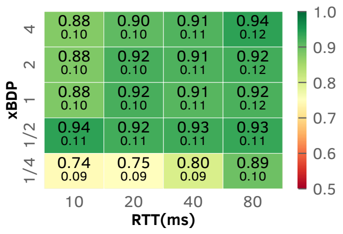
(b) *Cubic*



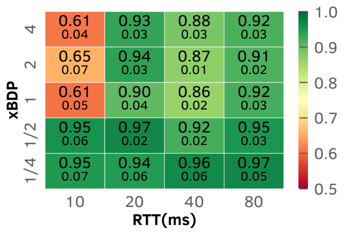
(c) *Copa*



(d) *PCC*



(e) *LEDBAT++*



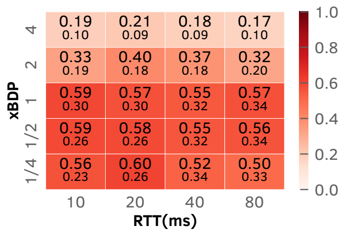
(f) *Sage*

Figure 4.8: *Contention between two flows with different RTTs when the long RTT flow starts first. The First flow has 160 ms RTT, while the RTT of the second is varied through four different values (indicated along the X-axis). The second flow starts 15 s after the start of the first. Figures a-d shows the mean Jain's fairness index values and the bottom value in a smaller font represents the standard deviation.*

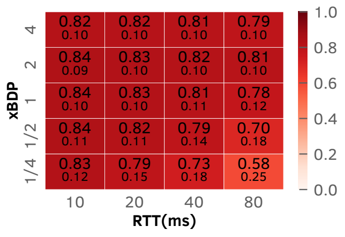
and the bottleneck capacity reverts to 100 Mbps for the remainder of the experiment. Unlike in §4.5.1, here we induce these bandwidth changes using two methods: by (i) starting another elephant flow using Cubic to represent an *elastic* cross-traffic (CT), and (ii) starting a flood of short-lived (i.e., mouse) flows using `hping` to represent an *inelastic* cross-traffic. Each method effects the bandwidth changes in three phases as described earlier. Finally, we repeat the experiments by changing the CCA of the primary flow and characterize how well different CCAs adapts to the dynamic cross-traffic along the path.

In Figures 4.10 and 4.11, we show the behavior of various CCAs when competing with different types of cross-traffic. As expected, introducing elastic cross-traffic (using Cubic) to induce bandwidth changes reveals that the behavior of the CCAs is significantly influenced by the configured buffer size [3]. At all phases, we note aggression towards the cross-traffic using Cubic from the BBR variants, and surprisingly, from PCC as well. BBR's unfairness towards loss-based algorithms is well-documented in prior work [1, 26, 56], and is more pronounced at shallow buffers, diminishing as the buffer size increases. PCC's behavior, however, is surprising; it uses almost 100% of the bottleneck capacity, leaving almost nothing for the Cubic flow. By contrast, LEDBAT++ demonstrates a more cooperative response to the Cubic flow, likely due to its design as a 'background traffic' CCA. Nevertheless, it exhibits greater variability compared to other CCAs, which can be attributed to its 'periodic slowdown'

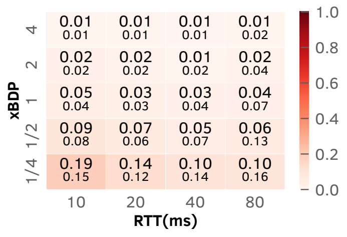
[3]Due to space constraints, we omit the results for 1/4xBDP; however, the trends closely align with that for 1/2xBDP across all phases.



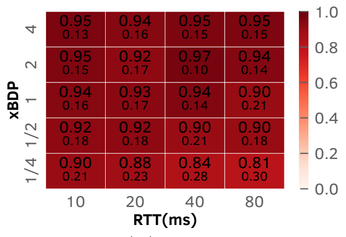
(a) *BBRv3*



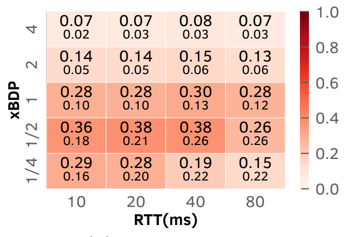
(b) *Cubic*



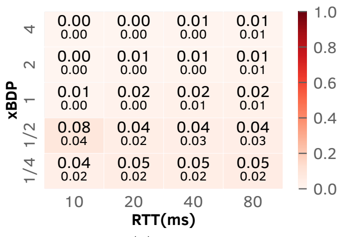
(c) *Copa*



(d) *PCC*



(e) *LEDBAT++*



(f) *Sage*

Figure 4.9: *Queue occupancy for two flows for when the long RTT flow starts first. Figures a-d show the mean normalized queue occupancy and the bottom value in a smaller font represents the standard deviation.*

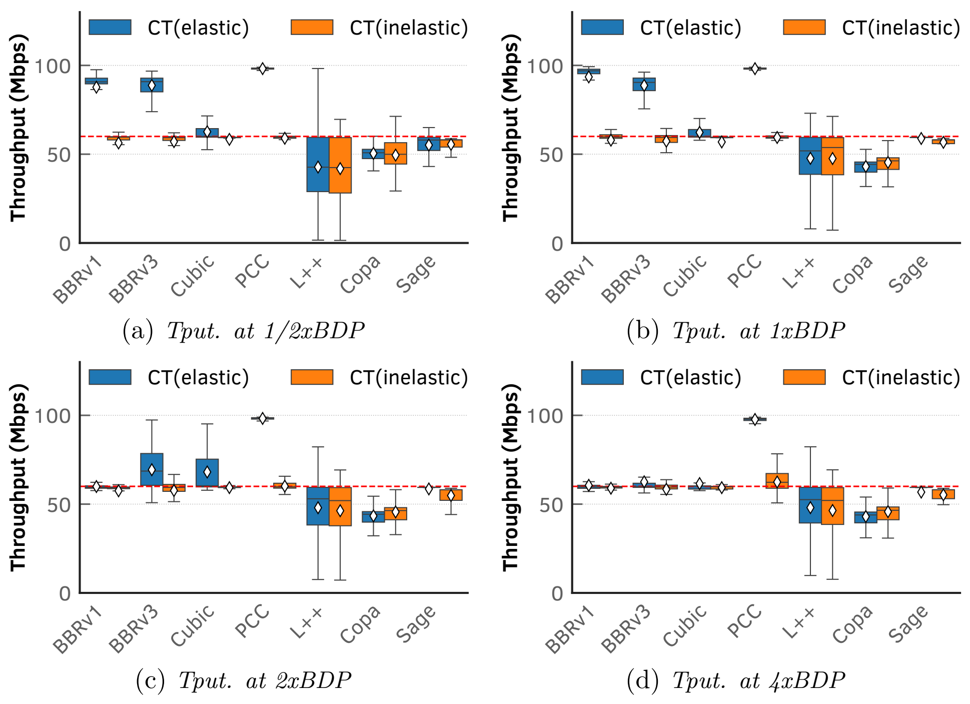
mechanism to probe for a base delay [168]. These experiments crucially quantify the extent to which various CCAs can share a bottleneck link equitably with the widely used CCA, Cubic.

Contending flows in the Internet may not always be cooperative. They may, for instance, be short lived and, hence, unable to being congestion controlled. When we use such short-lived flows to induce a significant volume of inelastic cross-traffic, all CCAs accurately respond to the bandwidth change, regardless of the buffer size at the bottleneck. Besides LEDBAT++, the median throughput of the primary flow in response to an inelastic cross-traffic matches the expected values in all three phases (refer Fig. 4.10 and Fig. 4.11)

Takeaways. *In the Internet, a flow may content with various types of cross-traffic, leading to changes in available bandwidth along a network path. Such changes may even be typical in cellular or WiFi networks where one of the hosts could be a mobile end-user device, and may also be effected by other competing elastic or non-elastic traffic. Whether a CCA can cope with such changes by quickly discovering available capacity and converging to it crucially determines its suitability for use in the Internet.*

4.5.2.4 A DoS scenario

Cross-traffic along a network path may also manifest in the form of denial-of-service (DoS) attacks. Evaluating a CCA's ability to recover under such extreme conditions is a critical component of a rigorous performance evaluation. However, most prior



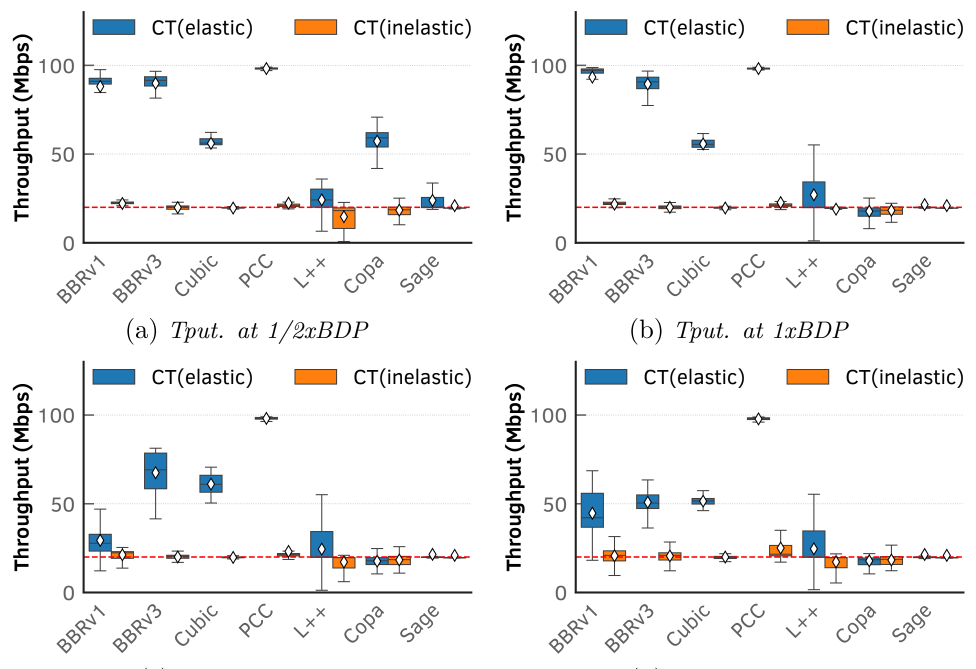
(a) *Tput. at 1/2xBDP* (b) *Tput. at 1xBDP*

(c) *Tput. at 2xBDP* (d) *Tput. at 4xBDP*

Figure 4.10: *The behaviour of various CCAs when faced with different types of cross-traffic in phases 1 and 3. The ideal bandwidth for the elephant (60 Mbps) is indicated in red.*

evaluations tend to overlook such scenarios. Our recipe demonstrates that many modern CCAs exhibit significant delays in recovering from such severe disruptions, even when the interruptions are brief. To this end, we leverage the same mice scenario (inelastic traffic) described earlier but modify it to deliberately disrupt the elephant flow. We do this by generating mice traffic in short, high-rate bursts. Specifically, we let a long-lived (i.e., elephant) flow saturate the link at 100 Mbps for 32 s. We then start mice flows using `hping` and run them for 10 s in 3 distinct phases: At $t = 32$ with mice traffic capped at 40 Mbps, at $t = 82$ with mice traffic capped at 80 Mbps, and finally at $t = 132$ with mice traffic capped 120 Mbps. We then calculate the time it takes the elephant flow to recover across all three phases of the DoS attacks. We first determined the average throughput of the elephant flow prior to the onset of the DoS attack, then computed the moving average of the throughput after the DoS attack using a 10-second window. We consider the elephant flow to have recovered when its post-DoS moving throughput average reaches 90% of the pre-attack throughput.

Figure 4.12a illustrates the phases during which the DoS attacks occur, while Figures 4.12b, 4.12c, and 4.12d, present the recovery times of various CCAs after the DoS traffic leaves. As expected, all CCAs struggle to recover from these severe DoS attacks, although the delay-based CCAs (with the exception of LEDBAT++) fare much better. When the DoS traffic takes up 80 Mbps and 120 Mbps, we observe that LEDBAT++ cannot recover back to its pre-attack throughput, while for Sage this behavior is observed at the lowest mice traffic cap (e.g., 40 Mbps). Cubic, being a



(a) *Tput. at 1/2xBDP* (b) *Tput. at 1xBDP* (c) *Tput. at 2xBDP* (d) *Tput. at 4xBDP*

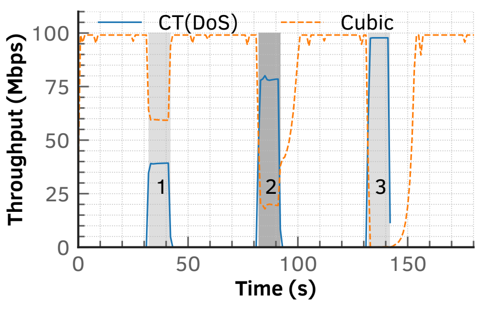
Figure 4.11: *The behaviour of various CCAs when faced with different types of cross-traffic in phase 2. The ideal bandwidth for the elephant (20 Mbps) is indicated in red.*

loss-based protocol, struggles to recover at especially the shallow buffers, presumably due to the losses experienced by the elephant flow during the attack, leading to a drastic reduction of `cwnd`. Overall, we observe that the recovery takes longer with more intense DoS traffic (e.g., 120 Mbps) and all CCAs exhibit longer recovery times, especially at shallow buffers.

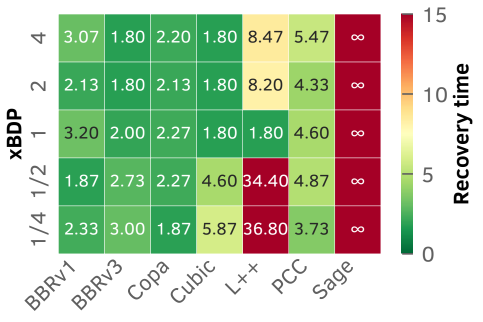
Takeaways. *CCA evaluations rarely assess an algorithm's behavior under extreme cross-traffic conditions to examine how it responds to such severe interruptions and the subsequent recovery time. However, such scenarios are not uncommon in the modern Internet and warrant thorough investigation.*

4.5.2.5 Topologies

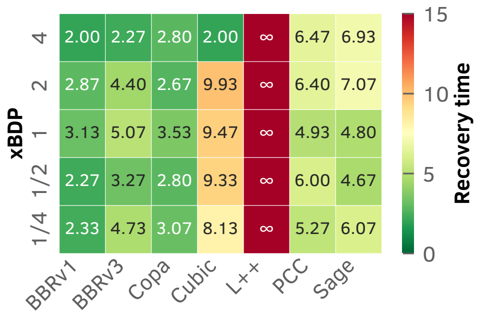
To scrutinize the performance of CCAs in various real-world network conditions, our recipe includes evaluations in both single and multiple bottleneck scenarios. To this end, we conduct an experiment where we run a long-lived flow–using `iperf`–to compete with an identical flow, having the same RTT and using the same CCA, in a dumbbell topology. We then replicate the experiment in a scenario with multiple bottlenecks, where the long-lived flow contends with two other flows across two distinct bottlenecks, resembling the multiple bottleneck described in [30]. The expected fair share for our flows in an ideal environment is 50% in both the topologies. In this experiment, we set the RTT to 100 ms for all flows, irrespective of whether they



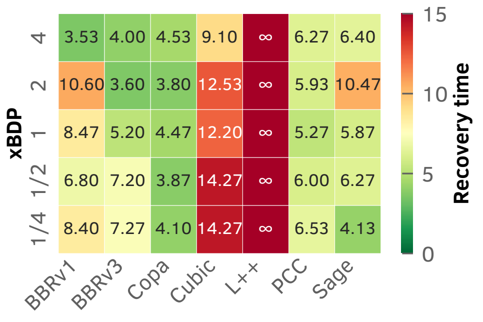
(a) *DoS traffic vs. Cubic*



(b) *Phase 1*



(c) *Phase 2*

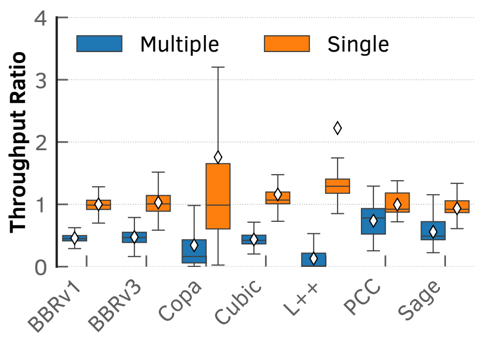


(d) *Phase 3*

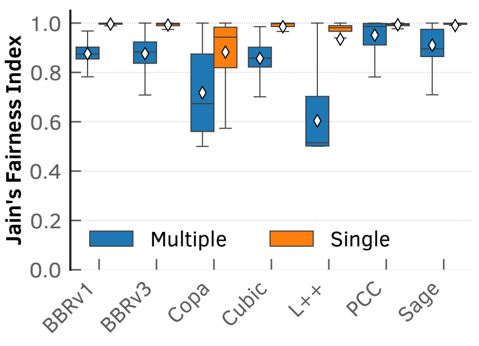
Figure 4.12: *Phases during which a CCA competes with an inelastic DoS traffic (a) where in phase 1, the DoS traffic is capped at (40 Mbps); phase 2 at (80 Mbps), and phase 3 at (120 Mbps). (b-d) Time to recover to 90% of pre-DoS throughput.*

traverse one or two bottlenecks. We run the flows for 300 seconds, excluding the initial 10 seconds ("warm up" phase) and the final 10 seconds from the results. We repeat the experiments three times and report the results. We employ two metrics to compare and contrast the behavior of the seven CCAs when competing with similar (i.e., same-CCA) flows.

The first metric is the throughput ratio. In the single bottleneck scenario, we define it as the ratio of the throughput achieved by the first flow to that of the second flow. In the multiple bottleneck scenario, however, we calculate this ratio by using the average throughput of the second and third flows as the denominator. The ideal value for the throughput ratio in both scenarios is one, signifying equal throughput for all flows throughout the experiment. Fig. 4.13a depicts the throughput ratio in the two scenarios (or topologies). In the single bottleneck scenario, all CCAs (on average) attain the fair value of one, but the narrative differs significantly in the multiple bottleneck scenario. BBRv1, BBRv3, Cubic, and Sage achieve a low mean throughput ratio of 0.5. Said differently, with these CCAs, the flow passing through two bottlenecks achieves approximately half the throughput compared to the other two flows on average—such unfair sharing may have strong implications for end-users' quality of experiences. Copa and LEDBAT++, compared to these four CCAs,



(a) *Throughput ratio*



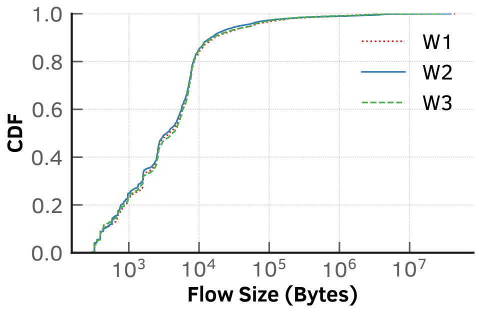
(b) *Jain's fairness index*

Figure 4.13: *(a) Throughput ratio (b) Jain's fairness index (JFI) of two flows when they compete in a dumbbell topology (with a single bottleneck) and in a parking lot topology (with multiple bottlenecks).*

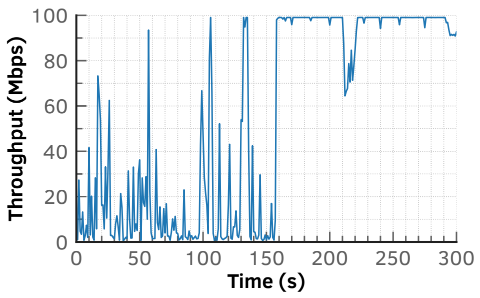
perform even worse, presumably due to difficulties in estimating the correct delay from the multiple bottlenecks. PCC, on the other hand, fares well compared to the rest: It achieves a median throughput ratio that is quite close to one; even at the 90th percentile, the flow encountering multiple bottlenecks achieves a throughput similar to that of the other two flows.

The second metric is the Jain's Fairness Index (JFI). In the single bottleneck scenario (i.e., dumbbell topology), we calculate the JFI between the two flows. In the multiple bottleneck scenario (i.e., parking lot topology), we determine the average of the JFIs between the long flow (traversing multiple bottlenecks) and each of the two other flows (traversing only a single bottleneck). Fig. 4.13b confirms our inferences from the earlier throughput ratio analyses, albeit it offers a slightly different perspective. JFI serves as a robust indicator for demonstrating whether flows share resources equitably, while the throughput ratio reveals which flows receive more bandwidth in case of an inequitable bandwidth sharing. Per this plot, Copa and LEDBAT++ struggle to achieve an equitable sharing between flows in a multiple bottleneck scenario, though our experiment design ensures that all the flows experience similar RTTs. Copa's poor performance, therefore, stems from its inability to help the long flow (traversing multiple bottlenecks) cope with more losses than the other two. It also exhibits a large variation in JFI across the different runs. PCC and Sage fare well compared to the rest of the CCAs, and BBRv3, surprisingly, fares poor compared to the BBRv1, the older version of the same CCA.

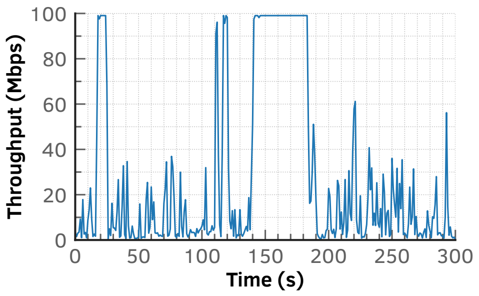
Takeaways. *Both the single bottleneck scenario (i.e., the dumbbell topology) and (less-widely used) mutiple bottleneck scenario (i.e., parking lot topology) are crucial to evaluate the real-world performance of a CCA. Our preliminary investigations with a parking lot topology shows significant changes in CCA performance, even if we ensure that all flows experience the same RTT.*



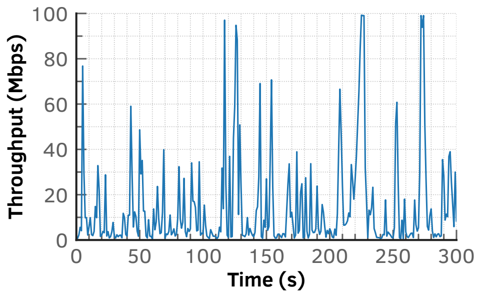
(a) *MAWI trace flows*



(b) *W1, 1xBDP*



(c) *W2, 1xBDP*



(d) *W3, 1xBDP*

Figure 4.14: *Although the three workloads follow the same distribution in (a), total throughput observed differs depending on the random seed chosen; the workload may fill the link, i.e., in (a), have long filling spikes i.e., in (c) or hardly fill the link at all, i.e., in (d). Figures (b), (c) and (d) show total throughput for Cubic at 1xBDP.*

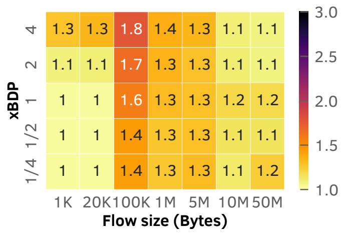
4.5.3 How does a CCA handle workload dynamics?

Next, we highlight the importance of considering different traffic workloads when evaluating CCAs. Link utilization differs between networks on the internet hence it is important to evaluate how different CCAs cope with different link utilization levels with a multitude of flow sizes. We start our experiments with a realistic workload, and then we add dynamic RTT per flow to create the closest environment to the Internet.

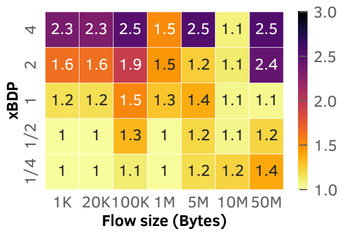
4.5.3.1 Examining varying workload intensities

We consider three different workloads representing three levels of link utilization to evaluate CCA behaviour. We generate the three workloads from the same MAWI [67] distribution by varying the random generator seed. The generated flow sizes in all three workloads follow the same distribution as shown in Fig. 4.14a, and their link utilization levels can be seen in Figures 4.14b, 4.14c and 4.14d respectively. We ran the three workloads for the TCP CCAs only, namely BBRv1 [4], BBRv3, Cubic and LEDBAT++. Running the same type of workload scenarios using the UDP-based

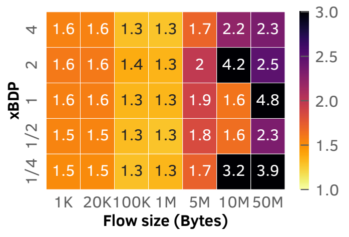
[4]We omit the BBRv1 results here due to space constraints.



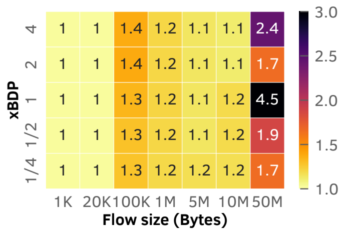
(a) *BBRv3 - W1*



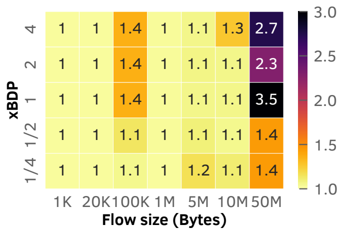
(b) *Cubic - W1*



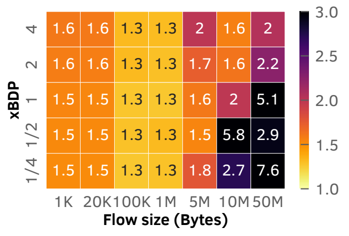
(c) *LEDBAT++ - W1*



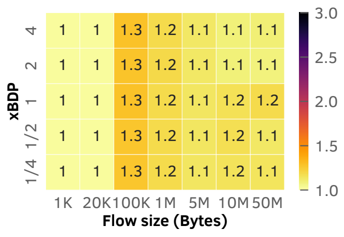
(d) *BBRv3 - W2*



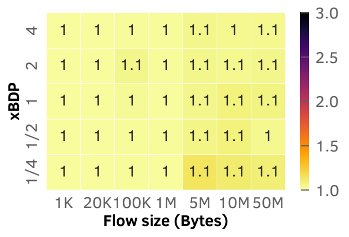
(e) *Cubic - W2*



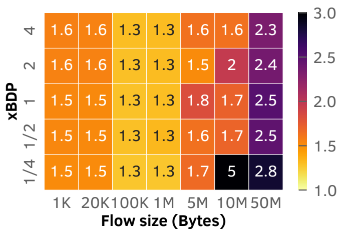
(f) *LEDBAT++ - W2*



(g) *BBRv3 - W3*



(h) *Cubic - W3*



(i) *LEDBAT++ - W3*

Figure 4.15: *FCT Slowdowns for workloads with different traffic intensities: Workload 1 includes large files that fill link, Workload 2 is less intense compared to Workload 1, and Workload 3 has the least intensity.*

CCAs (i.e., PCC and Copa) as well as Sage proved to be challenging and impractical. Nevertheless, we show that the behaviour of a CCA greatly varies when different traffic workloads are considered.

As described in §4.4, we use Harpoon [174] as the traffic generator and vary the bottleneck queue as a function of BDP. We set the link delay to 100 ms and run each experiment for 300 seconds. We calculate the FCT slowdown for each workload as in [28] as follows: $FCT\_Slowdown = \frac{oFCT}{eFCT}$. The $oFCT$ is the observed FCT and the $eFCT$ is obtained as shown in equation 4.1 and based on [181]; where A is the link capacity, N the flowsize, $icwnd$ the initial congestion window, and RTT the minimum RTT observed on the link.[5]

[5]In our testbed, end-to-end RTT has some fluctuations so we picked minimum observed RTT for our calculations.

$$eFCT = \begin{cases} RTT + \frac{N}{A}, & N \leq icwnd \\ FCT_{ss}, & icwnd < N \leq BDP \\ FCT_{p.ss} + \frac{N-(BDP)}{A}, & N > BDP \end{cases} \tag{4.1}$$

with $FCT_{ss} = (1+\lfloor(\log(\frac{N}{icwnd}))\rfloor)\times RTT$ and $FCT_{p.ss} = (1+\lfloor(\log(\frac{BDP}{icwnd}))\rfloor)\times RTT$.

In Fig. 4.15, we show the median FCT slowdown for selected CCAs across the three workloads. The contrast in FCT slowdown results between the workloads highlights the importance of evaluating CCAs in different link utilization scenarios. Workload 1, as seen in 4.14b, includes large files that fill up the link. This is reflected in the FCT slowdown, especially for Cubic (4.15b), since it is a buffer-filling protocol. LEDBAT++ (4.15c) also suffers, as long flows back off for short flows; however, even short flows are impacted by the long flows. The BBR variants see slightly better slowdown values, with BBRv3 (4.15a) seeing the least reduction in FCT across all flow sizes[6].

Workload 2, in contrast, exhibits less intense traffic, as shown in 4.14c; however, large files are drastically impacted by small flows in all four CCAs. Unlike in workload 1, where the BBR variants achieve low slowdowns for almost all flow sizes, Fig. 4.15d show a higher slowdown for the largest bucket of flow sizes for BBRv3, while small flows experience minimal slowdown. The Cubic results in 4.15e display a similar behavior. In contrast to BBRv3 and Cubic, LEDBAT++ maintains the same trend observed in workload 1.

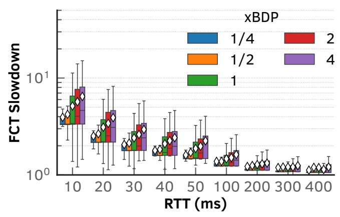
Workload 3, as shown in 4.14d, has the least intensity, which is reflected in the slowdown of BBRv3 and Cubic. By only using workloads with traffic profiles similar to that of workload 3 where the link is underutilized, we may be led to conclude that BBRv3 and Cubic have similar FCT slowdown values, as reflected in Figures 4.15g, and 4.15h. On the other hand, we observe that LEDBAT++ (4.15i) does not show significant improvement compared to the more intense workloads (i.e., workloads 1 and 2). Due to its design as a ‘scavenger’ algorithm, LEDBAT++ sacrifices the FCT for all flow sizes in order to be background traffic.

Takeaways. Depending on the intensity of a specific workload, CCA behaviour differs. For network administrators trying to decide which CCA is suitable for their network traffic, it is important that evaluations into a CCA’s performance provide insights that showcase such distinct behaviour.

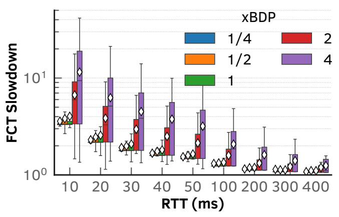
4.5.3.2 Multi-RTT workloads

Moving closer to real-world internet traffic, we repeat the experiments in the previous section with one major change: We assign different RTTs to the different flows in the workloads. We do the RTT assignments based on observed internet flow RTT distributions from the RIPE Atlas probes [182]. Based on the percentage of hosts observed to have a certain RTT or less, we assign that percentage of flows in our

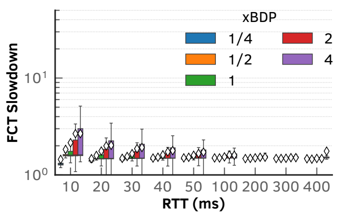
[6]BBRv1 shows a similar trend to BBRv3 in most cases, results omitted here for brevity.



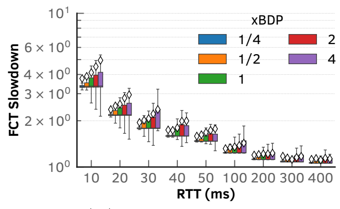
(a) *BBRv3 - W1*



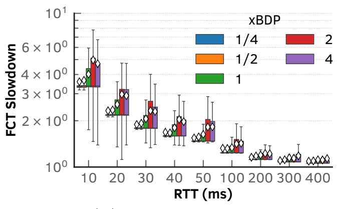
(b) *Cubic - W1*



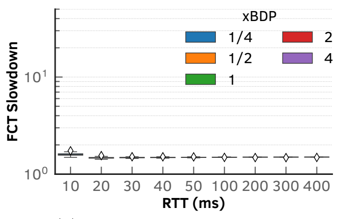
(c) *LEDBAT++ - W1*



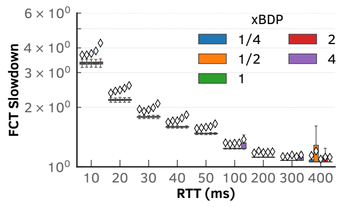
(d) *BBRv3 - W2*



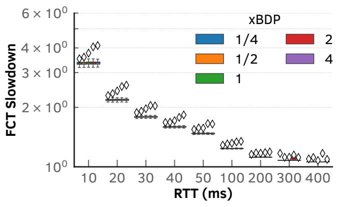
(e) *Cubic - W2*



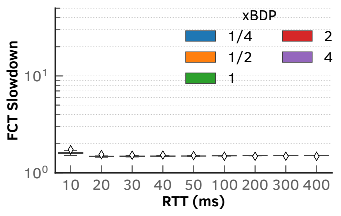
(f) *LEDBAT++ - W2*



(g) *BBRv3 - W3*



(h) *Cubic - W3*



(i) *LEDBAT++ - W3*

Figure 4.16: *Multi-RTT, FCT slowdown for workload 1, 2 and 3 per CCA.*

testbed with that specific RTT. For instance, if 26.82% of hosts were observed to have RTTs equal to or less than 10 ms, we assign 10 ms RTT to 26.82% of the flows in our experiment, and so on. We use Linux's `tc` utility to filter ports and assign RTTs to the different flows based on specific port ranges.[7] As before, we vary the queue as a function of BDP and run each workload for 300 seconds.

In Fig. 4.16 we show the FCT slowdown obtained by selected CCAs as a function of the queue size and RTT, for workload 1. As seen in the previous section, the effect of the workload's intensity on the FCT is obvious, with workload 1 presenting high overall slowdown values. It is interesting to note that Cubic's average FCT slowdown is much less compared to BBRv3 when the queue size is 1/4x, 1/2x and 1x of BDP, across almost all RTT configurations. This is because at low queue sizes, Cubic does not get a chance to fill the buffer before receiving a congestion signal and consequently backing off. However, as expected, Cubic's FCT slowdown increases as the queue size increases (Figures 4.16b). While this observation is also visible for BBRv3 in Fig. 4.16a, it is not as exaggerated.

At low RTTs, BBRv3 uses the low RTT observed as a signal that the link is free, and thus pushes more data into the network, resulting in congestion and increased FCTs, as can be seen in Fig. 4.16a. When RTTs are high, BBRv3's estimated BDP used to

[7]We configured our testbed such that all TCP ports in the range 1000 - 65535 were usable.

calculate its sending rate is too high, leading to a somewhat bloated queue. Both low and high RTTs serve as a double-edged sword to BBRv3's FCT slowdown.

The results of this experiment with LEDBAT++ appear somewhat different, as shown in Fig. 4.16c. For both low RTT and high RTT flows, LEDBAT++ exhibits significantly lower FCT slowdown compared to BBRv3 and Cubic. Our further analysis revealed that LEDBAT++ was able to send only about 10% of the total flow count sent by both Cubic and BBRv3. This difference is primarily due to LEDBAT++ further underutilizing the bottleneck link as a result of its backoff mechanism, where all flows yield to one another.

Takeaways. In workloads of flows with varying RTTs, we observe unexpected wins for Cubic in terms of lower FCT slowdown at low buffer sizes. Another interesting trend is that the FCT slowdown decreases as the RTTs increase, the effect of which is not easily detectable in evaluations that only consider workloads consisting of flows with the same RTTs.

4.6 Summary

Despite the extensive body of work on congestion control, spanning about four decades, we do not yet have a clear consensus on how to benchmark congestion control algorithms (CCAs). In this Chapter, we highlighted the implications of a lack of consensus on CCA benchmarking on future research.

We addressed the issue head on by presenting a CCA benchmarking "recipe" that distills and adapts past work on evaluating CCAs into three fundamental questions concerning the behavior of CCAs: *How does a CCA cope with the performance unpredictability of the end-to-end path? How does a CCA adapt to cross-traffic? How does a CCA handle workload dynamics?* We demonstrated how answering these questions offers a rigorous characterization of a CCA's behavior in a wide range of network conditions.

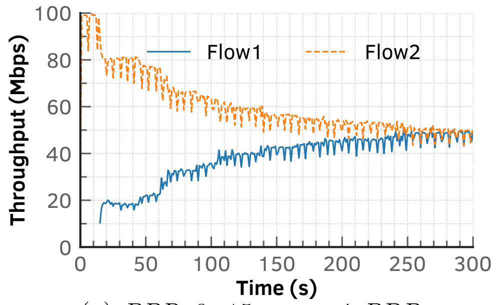
5 Case Study 1: BBR Congestion Control

BBR is a relatively new congestion control algorithm (CCA) that takes a proactive approach in detecting congestion on a network path. Instead of relying on signals that are weakly correlated with congestion (e.g., packet loss and transient queue delay), BBR characterizes a path using two parameters, bottleneck bandwidth and round-trip propagation time. This design allows BBR to converge with a high probability to Kleinrock's optimal operating point [34], and ensures maximized throughput and minimized delay and loss in a stable state. BBR is currently used by Google and several CDNs [19–21]. Since it was first introduced in 2016, BBR's authors have consistently presented evaluations to demonstrate its efficacy in various network scenarios. However, independent evaluations have showed BBR's other performance issues including unfairness to loss-based algorithms, high retransmissions, RTT unfairness, etc. [1, 26, 56, 128, 138, 139].

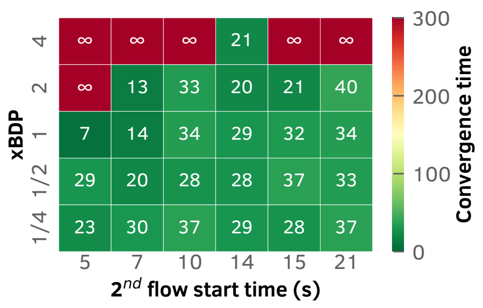
A majority of prior BBR evaluations consider its performance in network scenarios where flows start at the same; a simplistic condition that is unrealistic in the Internet. To illustrate the lack of generalizability in characterizing a CCA's behaviour that arises from such approaches, we consider a simple scenario with flows arriving at a bottleneck at different times in Figure 5.1a. In this experiment, we stagger the flow arrivals by several starting times–5, 7, 10, 14, 15, 21 s–each constituting a separate experiment, to characterize how the arrival of a flow affects existing flows in the network [1].

We assess the performance of BBRv1, BBRv3 and Cubic based on convergence, a key metric for evaluating the efficiency of a CCA in achieving stable and steady utilization of the bottleneck bandwidth [183]. Convergence further reflects the algorithm's ability to quickly adapt to newly available bandwidth or recover from failures. Although many prior works considered convergence times, definitions often differ between proposals [15, 92, 136, 137]. Further complicating the matter, some researchers argue that determining the point of convergence is highly influenced by the duration of the experiment [184]. Hence, since there is no consensus on the "appropriate" experiment duration, establishing an *exact* point at which we can say flows have converged remains challenging [184]. Moreover, no widely accepted standard exists for characterizing the oscillatory flow behavior post-convergence, particularly in both intra-CCA and inter-CCA scenarios. Nevertheless, we lean on the definition in [92]

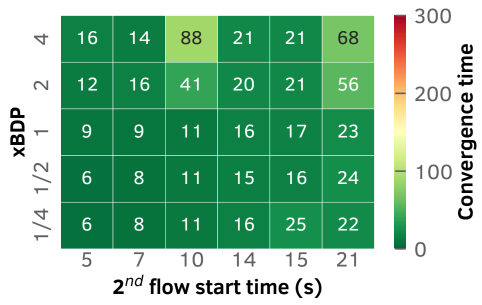
[1]It can be argued that starting the second flow at multiples of 5 s will conflict with the RTT probing time in BBRv1 and BBRv3, however, flow start times in the Internet are uncontrolled



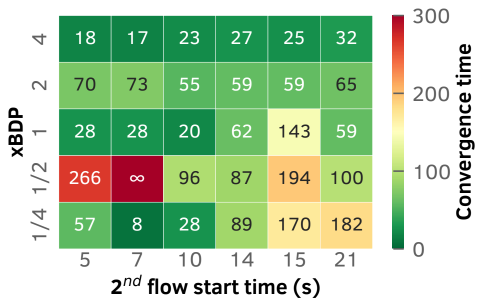
(a) *BBRv3, 15 s gap, 4xBDP*



(b) *BBRv3*



(c) *BBRv1*



(d) *Cubic*

Figure 5.1: *Convergence to a fair share is influenced by flow arrival.*

and define convergence time as time taken to reach a Jain's fairness index threshold of 0.95 (which we assume to represent equitable sharing) within 5 s windows.

If we focus our attention to the third row which corresponds to a bottleneck buffer set to BDP (a typical configuration) in Figures 5.1b, 5.1c and 5.1d, we observe that the time to reach the fairness threshold is highly influenced by the start time of the second flow. For BBRv3 (Fig. 5.1b), simply staggering the arrival of flows at a bottleneck link substantially changes it's behavior, much worse than that seen in BBRv1 (Fig. 5.1c) and almost similar to what is observed in the case of Cubic (Fig. 5.1d). Regardless of the second flow's start time, we observe that BBRv3 is often unable to reach convergence until the experiment ends, especially at large BDP configurations, compared to both BBRv1 and Cubic. This may be a cause for concern as BBRv3 is seeing increased Internet deployment [62], and flows in the Internet may have different starting times. Furthermore, there is still continued existence of deep buffers in the public Internet [160–162] and the ongoing adoption of BBR across the public Internet necessitates a more explicit emphasis on such performance observations [19–21, 131].

In this chapter, we take a deep dive into the behavior of the BBR variants and further demonstrate how a benchmarking recipe allows us to focus on often overlooked dimensions of evaluations. We demonstrate how the questions outlined in Chapter 4 serve to define a minimal set of *necessary* evaluations, guiding researchers to adequately consider corner cases in assessing a new CCA's efficiency, and enabling the discovery of new performance insights.

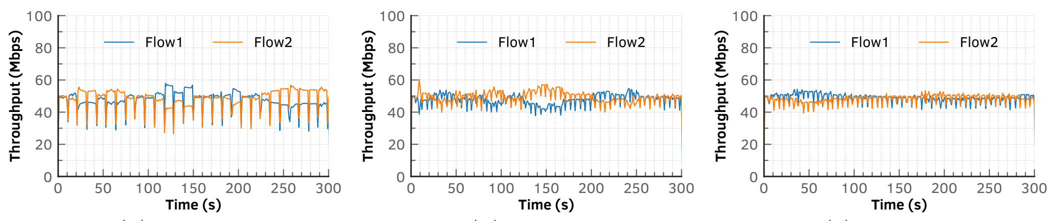
5.1 Related Work

Several independent studies have examined BBR's performance in various scenarios and compared it with that of other CCAs, partly perhaps because of how quickly Google has been transitioning large volumes of traffic to use BBR [62]. Many prior work such as [26, 56, 128, 138, 139] conducted comprehensive evaluations of BBRv1, focusing on its interactions with other well-known and widely used CCAs, e.g., NewReno, Cubic, and Vegas. Past evaluations also considered both similar and dissimilar flows (i.e., flows with varying RTTs or using different CCAs). Some investigated BBRv1's performance in both shallow and deep buffer scenarios [26, 56, 139], and some others analyzed the impact of extremely long delays, which are typical in satellite networks [185]. Prior studies identified several shortcomings with BBRv1, including unfairness to loss-based CCAs in shallow buffers, frequent retransmissions, high RTT unfairness as well as high queue occupancy. The scope of virtually all such evaluations was, however, limited to long-lived flows. Hurtig et al. tested BBRv1 with various buffer sizes and link bandwidths and used a workload comprising a mix of bulk as well as short transfers [186], but they used fixed sizes (instead of sampling them from observed traffic distributions) for long-lived and short-lived flows.

Active queue management (AQM) techniques are designed for mitigating congestion [187], and several prior work have studied the impact of deploying AQMs on the performance of different CCAs (e.g., [188]). In the case of BBR, studies have shown that AQMs such as FQ-CoDel improve the performance of both BBRv1 and BBRv2 in deep buffer scenarios [140, 189, 190]. We consider the analyses of AQM deployments on BBRv3 as orthogonal to this work.

BBRv2 uses ECN signals to detect queuing, and Tahiliani et al. [191] showed that BBRv2 reduces queue occupancy at the bottleneck, when ECN was enabled. Multiple studies demonstrated that BBRv2 has lower link utilization than BBRv1 owing to its conservative behavior during bandwidth probing and that both versions were unfair to loss-based CCAs in deep buffer scenarios [59–61, 192]. We did not evaluate the benefits of ECN for BBRv3 in this study, and leave that for future work.

Some recent work also attempted to address the BBR's shortcomings by implementing various improvements. Bi et al. [193], for instance, focused on BBRv1's frequent retransmissions issue. They proposed dynamically adjusting the in-flight data cap based on the estimated bottleneck buffer size from RTT samples as well as packet losses. BBRv2+ used path delay information for tuning the aggressiveness of bandwidth probing [192]. They used Mahimahi [124] for evaluating BBRv2+ in emulated WiFi and LTE networks. While we do not propose solutions for addressing the shortcomings in BBRv3, we believe our comprehensive evaluations and detailed analyses will pave the way for designing and validating different solutions.



(a) *BBRv1* (b) *BBRv2* (c) *BBRv3*

Figure 5.2: *Throughput achieved by two similar, synchronous (i.e., starting at the same time) flows when competing for bandwidth on a bottleneck with a deep buffer of size 16×*`BDP`*.*

5.2 Evaluation

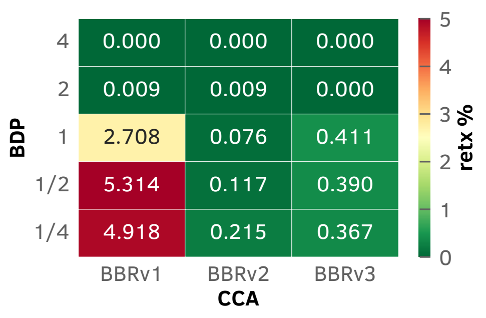
In line with our benchmarking recipe (Chapter 4), we now evaluate the three versions of BBR and Cubic in a wide range of network conditions. Our goal is to ascertain the ability of each CCA to share the bandwidth in a fair or equitable manner with other flows—regardless of whether the contending flows use the same CCA or a different one. To this end, we evaluate fairness and throughput performance in settings that encompass at least one dimension from the 3 questions described in Chapter 4.

5.2.1 Path performance unpredictability

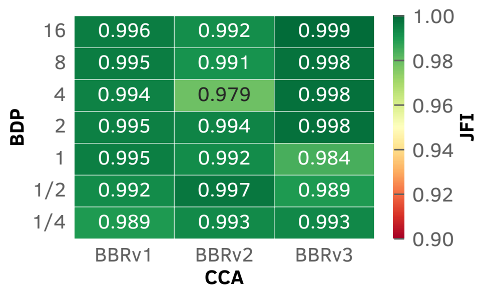
We begin our evaluation with a simple scenario where two similar flows, i.e., using the same CCA and experiencing the same RTT (of 100 ms), contend for bandwidth on a bottleneck (refer Fig. 3.2). We start both flows *at the same time* and let them run for a duration of 300 s and plot the throughput experienced by the flows as a function of time in Fig. 5.2. We repeated this experiment by varying the bottleneck queue size from one-fourth to 16 times the `BDP`, doubling the queue size once for each experiment. Overall, all BBR variants achieve an equitable bandwidth share when competing with a similar and synchronous flow. BBRv3 significantly reduces the throughput oscillations and allows the flows to converge much quicker than either of the earlier two versions. Although not shown here, our inferences hold across all buffer size settings.

Fig. 5.3a indicates whether the flows experience high retransmissions when competing with another similar, synchronous flow.[2] The heatmap reports the percentage of retransmissions (i.e., ratio of the aggregate number of retransmitted packets to the total number of packets delivered by both flows) experienced by the two flows; higher values indicate higher retransmissions, and, hence, lower values (in green) imply better goodput or link utilization. Per this figure, although BBRv1 experiences high retransmissions when competing with another BBRv1 flow started at the same time in low buffer settings, across all other settings, the CCAs converge quickly without incurring much loss. This observation of high retransmissions when using BBRv1 in

[2] We omit the results for $8 \times$ `BDP` and $16 \times$ `BDP` as we see virtually no retransmissions in such settings.



(a) *Retransmissions*



(b) *Jain's fairness index (JFI)*

Figure 5.3: *(a) Retransmissions experienced by and (b) Jain's fairness index (JFI) of two similar, synchronous flows competing for bandwidth on a bottleneck link as a function of bottleneck queue size.*

shallow buffers is in line with observations made by prior work [56, 138, 139]: They are due to BBRv1' aggressive probing behavior and its indifference to loss. The Jain's fairness index (JFI) for the flows in Fig. 5.3b, however, shows that all three versions equitably share the bottleneck with a similar and synchronous flow, regardless of bottleneck buffer size; BBRv3 shows only a marginal improvement, if any, over the prior two versions.

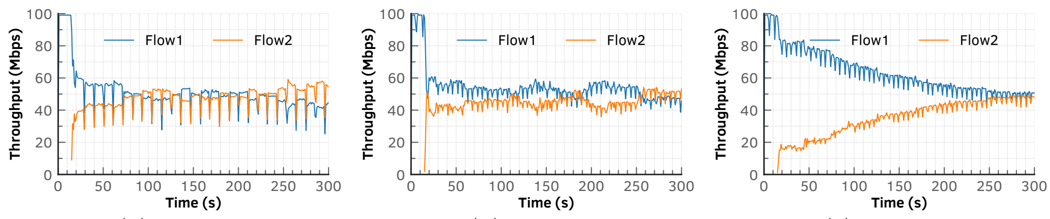
Takeaways. *BBR achieves an equitable sharing of bandwidth with similar synchronous flows, and BBRv3 address the well-documented issues of aggressiveness and unfairness in BBRv1.*

5.2.2 Adapting to cross traffic

5.2.2.1 Staggered flows

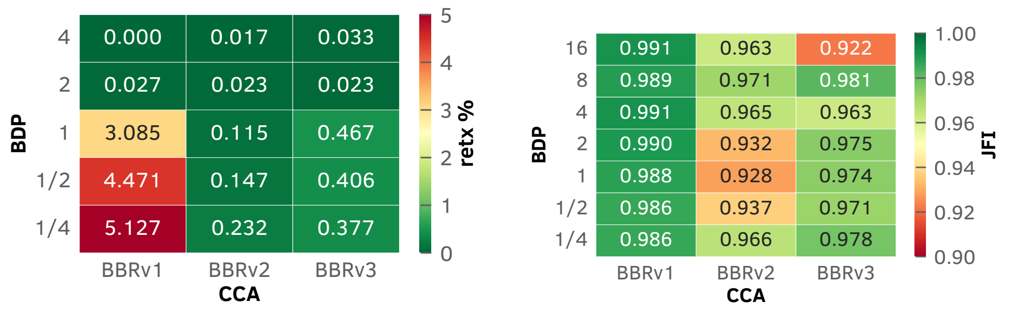
As with the simple scenario performed in this chapter's introduction, we take a deeper look at the staggered flows where the second flow starts 15 s after the first. In the Internet, flows may arrive at a bottleneck at random times, when other contending flows on the link have already started or reached their stable state. The staggering of flows simply emulates this typical traffic condition.

Per Fig. 5.4, BBR implementations do not converge quickly to an equitable share when flows do not start at the same time—quite unlike the case of similar and synchronous flows (§5.2.1). Similar to the 4xBDP case, we note that when a BBRv3 flow joins 15 s later at a bottleneck link with a 16 × `BDP` buffer, it takes *more than 4 minutes* for it to achieve an equitable bandwidth sharing. Such large buffers are not uncommon in the internet: Router manufacturers may (by default) provide large buffers [194], and administrators may configure larger buffers on transcontinental links to cope with the high RTTs [162] or improve video QoE [195].



(a) *BBRv1* (b) *BBRv2* (c) *BBRv3*

Figure 5.4: *Throughput achieved by two similar but staggered flows (i.e., second flow starts 15 s after the first) when competing for bandwidth on a bottleneck with a deep buffer of size 16×*`BDP`*.*

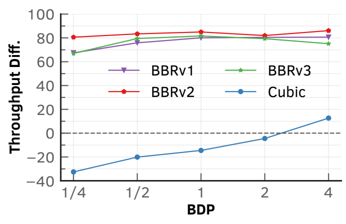


(a) *Retransmissions* (b) *Jain's fairness index (JFI)*

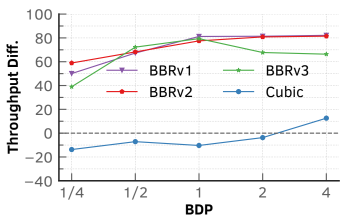
Figure 5.5: *(a) Retransmissions experienced by and (b) JFI of two similar but staggered flows (i.e., second one starts 15 s after the first flow) competing for bandwidth on a bottleneck link as a function of bottleneck buffer size.*

We observe that when flow starts are staggered by an interval of 15 s, the flows experience higher retransmissions (Fig. 5.5a) than when they are started at the same time.[3] The heatmap in the plot reports, as in §5.2.1, the percentage of retransmissions experienced by the two flows; higher values (in yellow and red) indicate higher retransmissions. BBRv1 in particular suffers high retransmissions in shallow buffer settings, again owing to its aggressive probing behavior and indifference to losses. A key change in BBRv2 (compared to BBRv1) was the addition of using loss as a congestion signal; this change enables it to reduce the loss rates in shallow buffer settings and avoid unnecessary retransmissions [169]. BBRv2 experiences, as a result, the lowest retransmissions of all three implementations. BBRv3 inherits BBRv2's changes and, as a consequence, experiences low retransmissions, which matches Google's claims about BBRv3 [62]. We observe that BBRv3 experiences more retransmissions than BBRv2 in shallow buffer settings, but the former achieves better fairness than the latter across a range of buffer sizes (Fig. 5.5b), with the exception of the largest buffer size (of 16-times `BDP`). BBRv1, which does not use packet loss to infer congestion, leads to higher fairness than BBRv2 and BBRv3, both of which take loss into con-

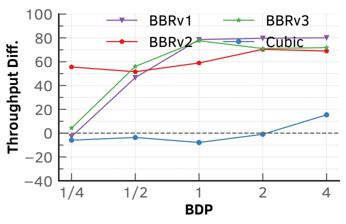
[3] As before, we omit the results for 8 times `BDP` and 16 times `BDP` as we see virtually no retransmissions in such settings.



(a) *TD; 160 ms vs 10 ms*

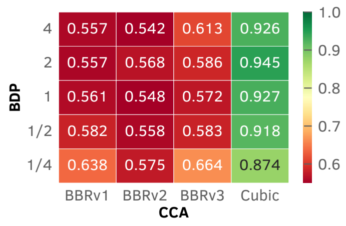


(b) *TD; 160 ms vs 20 ms*

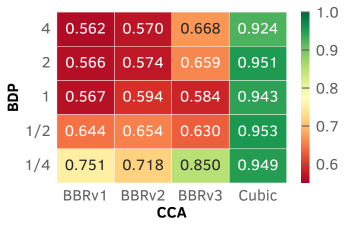


(c) *TD; 160 ms vs 40 ms*

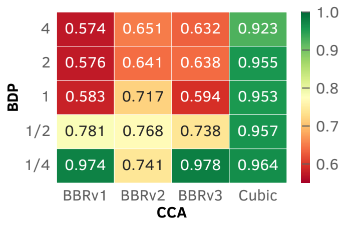
Figure 5.6: *Average throughput difference (TD); 160 ms flow starts first, then 15 s later the (10 ms, 20 ms, 40 ms) flow starts.*



(a) *JFI; 160 ms vs 10 ms*



(b) *JFI; 160 ms vs 20 ms*



(c) *JFI; 160 ms vs 40 ms*

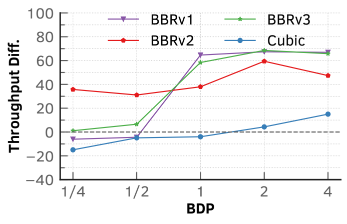
Figure 5.7: *Average Jain's fairness index (JFI); 160 ms flow starts first, then 15 s later the (10 ms, 20 ms, 40 ms) flow starts.*

sideration. These observations could partly be explained by BBRv3's less aggressive behavior compared to BBRv1 (compare BBRv3's `cwnd_gain` and `pacing_gain` to those of BBRv1 in Table 1 in [1]).

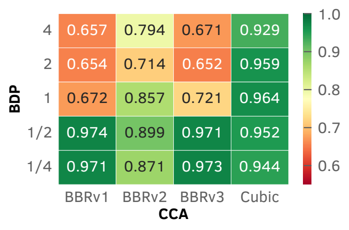
Takeaways. *While BBR's refinements seem to allow it reduce losses, with recent versions taking losses into account to control their probing, even the most recent version of BBR struggles to achieve high fairness when flow arrivals are staggered by a few seconds. This observation has crucial implications for BBRv3's adoption in the Internet, since in the public Internet arrivals of competing flows might often be staggered with respect to one another.*

5.2.2.2 Dissimilar flows: RTT fairness

Evaluations concerning fairness of CCAs in prior work (e.g., [26, 139]) usually consider flows with similar RTTs. Flows in the Internet typically have different RTTs, which make it challenging in terms of fairly sharing bandwidth when such flows interact at a bottleneck link, even if they all use the same CCA. The RTT dictates a CCA's reaction to bandwidth limitations or to other flows, since the signals typically used by a sender (e.g., packet loss or delay) for determining how much it can send and/or how fast, must traverse the path from the bottleneck to the receiver, which then echoes it back to the sender.

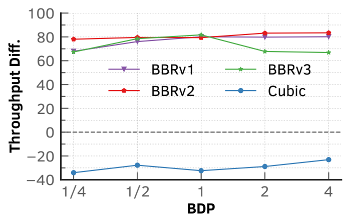


(a) *TD; 160 ms vs 80 ms*

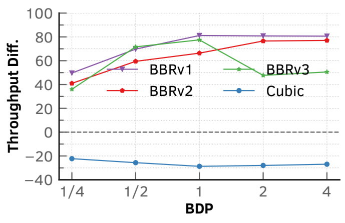


(b) *JFI; 160 ms vs 80 ms*

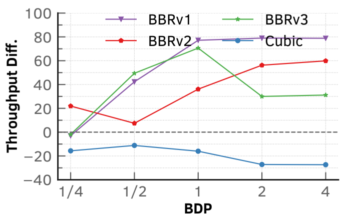
Figure 5.8: *Average throughput difference (TD) and Jain's fairness index (JFI); 160 ms flow starts first, then 15 s later the 80 ms flow starts.*



(a) *TD; 10 ms vs 160 ms*



(b) *TD; 20 ms vs 160 ms*



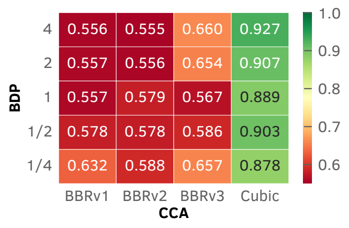
(c) *TD; 40 ms vs 160 ms*

Figure 5.9: *Average throughput difference (TD); short RTT flow (10 ms, 20 ms, 40 ms) starts first, then 15 s later the 160 ms flow starts.*

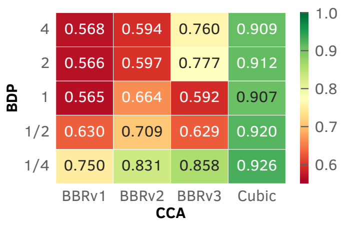
With this experiment, we evaluate the bandwidth share when flows using the same CCA traverse different routing paths, thus experiencing different RTTs, and sharing a common bottleneck along the way. We run two flows using the same CCA, where the first flow is our base flow with a fixed RTT of 160 ms and the second flows experiences a different RTT—one of $10, 20, 40, 80$ ms, each constituting a different experiment. In the first experiment scenario, the base flow, 160 ms, starts first and the second flow joins after 15 seconds (Fig. 5.6, Fig. 5.7 and Fig. 5.8). In the second scenario, the base flow joins 15 seconds after the flow with the shorter RTT (Fig. 5.9, Fig. 5.10 and Fig. 5.11). These scenarios help us characterize how the different CCAs behave when the short (or long) RTT flow joins the network, while the other flow has already reached a steady state; we can safely assume a flow has finished slow start after 15 seconds.[4] Specifically, we calculate the throughput difference between the base flow and the second flow, as follows: $BaseFlow_{AvgTput} - SecondFlow_{AvgTput}$. A positive value implies that the base flow obtains the majority of the bandwidth, while a negative value indicates that the second flow receives higher bandwidth than the first. Additionally, we also compute the Jain's fairness index to quantify the CCA's fairness.

When we start the long-RTT flow first, all versions of BBR favor the long-RTT flow (Fig. 5.6); it receives much higher bandwidth than the second flow. This bias towards the long-RTT flow (starting first) diminishes a little when the difference in flow RTTs

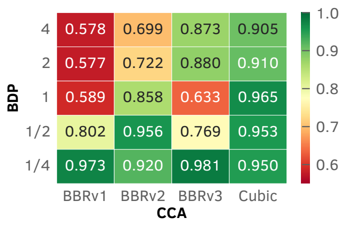
[4]We do not disturb a flow while in slow start to ensure that it does not exit slow start prematurely, which would result in link underutilization.



(a) *JFI; 10 ms vs 160 ms*

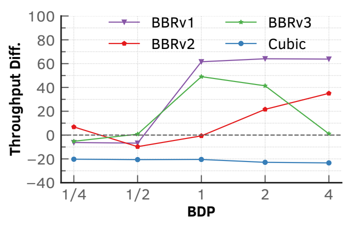


(b) *JFI; 20 ms vs 160 ms*

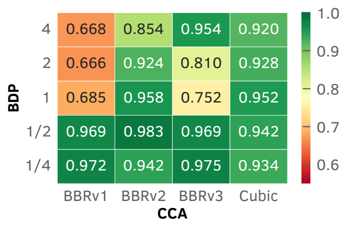


(c) *JFI; 40 ms vs 160 ms*

Figure 5.10: *Average Jain's fairness index (JFI); short RTT flows (10 ms, 20 ms, 40 ms) start first, then 15 s later the 160 ms flow starts.*



(a) *TD; 80 ms vs 160 ms*



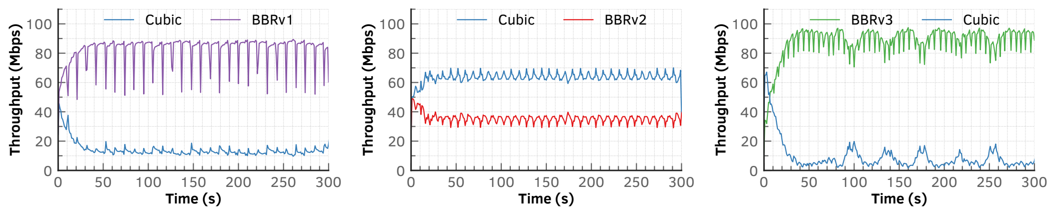
(b) *JFI; 80 ms vs 160 ms*

Figure 5.11: *Average throughput difference (TD), 80 ms flow starts first, then 15 s later the 160 ms flow starts.*

decreases (Fig. 5.6c); BBRv2 and BBRv3 (which is largely similar to BBRv2) perform much better than BBRv1 when RTT differences between the base flow and second flow decreases (Fig. 5.8), perhaps owing to their less aggressive design, i.e., by reacting to loss signals. These observations also hold if we start the short-RTT flow first and then the long-RTT (or base) flow (Fig. 5.9 and Fig. 5.11). Our inferences are in agreement with prior work that demonstrated BBR favoring long-RTT flows over short-RTT flows [56, 128], which is quite different from what has been observed in case of AIMD mechanisms [47, 196]; our results confirm that BBRv3's behavior does not vary substantially from BBRv2 with respect to preferring long-RTT flows over short-RTT flows. Cubic, unlike BBR, favors the second (short-RTT) flow over the base (long-RTT) flow, even though the second flow starts after the base flow, across almost all buffer settings.

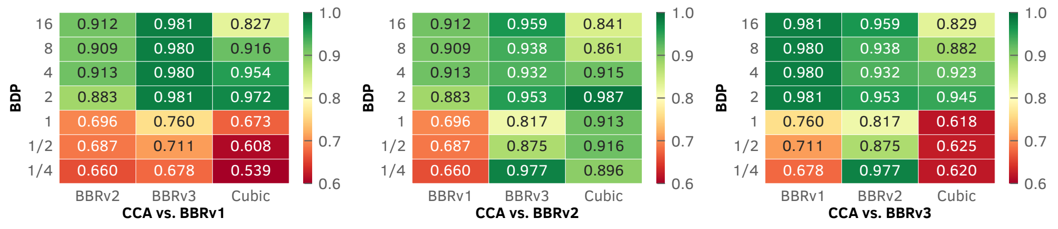
All BBR versions exhibit high RTT unfairness when the RTT difference between the base and second flow is significant, regardless of which flow starts first (Fig. 5.7, Fig. 5.8, Fig. 5.10, and Fig. 5.11). Unlike BBR, Cubic achieve high fairness across all settings, regardless of how we size the bottleneck buffer. BBR's bias towards long-RTT flows is less pronounced in shallow buffers (if the RTTs of the flows do not differ substantially), where the short-RTT flow is able to compete for a better bandwidth share with the long-RTT flow.

Fairness is improved for all BBR versions if the short RTT flow starts first, as it is able to grow its `cwnd` before the long RTT flow starts. Additionally, the smaller the



(a) *BBRv1* (b) *BBRv2* (c) *BBRv3*

Figure 5.12: *Throughput of a BBR flow when competing for bandwidth with a Cubic flow on a bottleneck link with a 1×BDP buffer.*



(a) *BBRv1 vs. others* (b) *BBRv2 vs. others* (c) *BBRv3 vs. others*

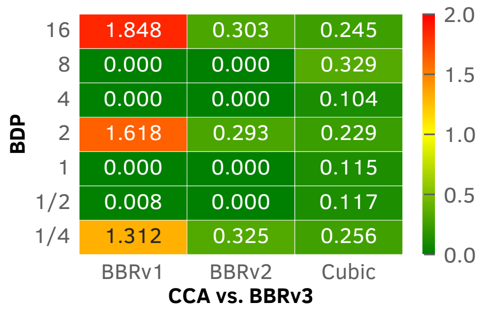
Figure 5.13: *Jain's fairness index (JFI) of two flows using different CCAs and competing on a bottleneck with a 1×BDPbuffer.*

gap between the two flow's RTT values, the better the fairness as both flows are able to estimate the bottleneck bandwidth and RTT at almost the same rate (Fig. 5.8 and Fig. 5.11). When BBR flows with different RTTs compete for bandwidth, the short-RTT flow is likely to observe the increase in queuing faster than the long-RTT flow (since the rate at which a sender can observe such signals depend on the RTT). Furthermore, since BBR determines the delivery rate based on such signals, the short-RTT flow may consistently slow down in response to any observed queuing before the long-RTT flow reacts. When flows eventually synchronize, short-RTT flows regain some of the bandwidth share, but the situation reverts to benefiting the long-RTT flow again as they probe for bandwidth, due to the long-RTT flow estimating a higher BDP value.

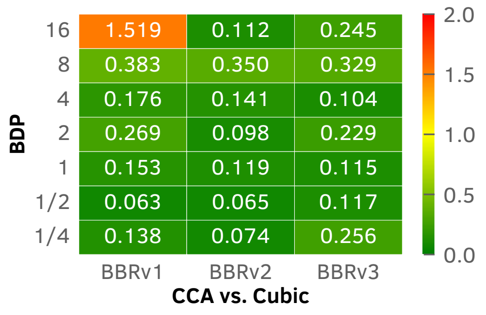
Takeaways. BBR's bias towards long-RTT flows is well-known, and our evaluations confirm that BBRv3 improvements or optimizations do not change the status quo. BBR offers poor fairness across flows that differ substantially in RTT, which might be typical in the Internet. Cubic, the widely used CCA in the Internet, unlike BBR, offers higher fairness than BBR in nearly all our evaluations scenarios with flows with different RTTs.

5.2.2.3 Dissimilar flows: Inter-CCA fairness

We now evaluate BBRv3's ability to share bandwidth equitably when competing with flows using CCAs other than BBRv3. With more than 40% of current Internet



(a) *BBRv3*



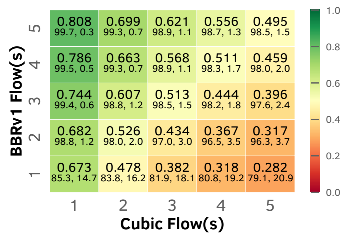
(b) *Cubic*

Figure 5.14: *Retransmissions when (a) a BBRv3 flow and (b) a Cubic flow compete against flows using other CCAs.*

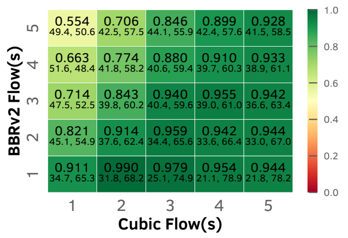
traffic estimated to be delivered using BBR [19], it is quite important to characterize how BBRv3 behaves when competing with non-BBRv3 flows already prevalent in the Internet. Such an inter-CCA characterization is also timely and crucial for network operators and generally the networking community, since Google intends to submit BBRv3 soon for inclusion in the Linux kernel [62]. To characterize BBRv3 flows' ability to co-exist with non-BBRv3 flows, we run two (co-existing) long-lived flows, each using a different CCA; we start these flows at the same time. We set the RTT to 100 ms and the bandwidth to 100 Mbps, and we vary the bottleneck buffer size as a function of BDP.

BBRv1 has long been reported to be extremely unfair to loss-based CCAs such as Cubic [26, 56, 128, 139]. Fig. 5.12a, hence, simply confirms this well-established claim. BBRv2 seems to fare well than BBRv1 (Fig. 5.12b), since unlike the former, the latter takes packet loss into consideration. BBRv2 improves upon BBRv1 with respect to its ability to share bandwidth with loss-based CCAs, even allowing Cubic to obtain more than its fair share of bandwidth when competing with BBRv2 flows. BBRv3, which Google claims will quickly converge to fair share, performs even slightly worse than BBRv1 (Fig. 5.12c). Even if flows experience same RTTs, BBRv3, in its current form, offers no room for Cubic flows to co-exist; any RTT differences between the flows may only exacerbate this situation.

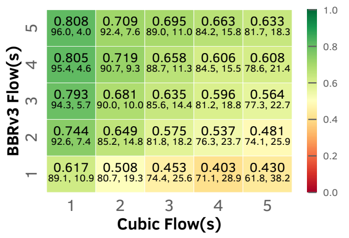
Fig. 5.13 shows the Jain's fairness index for the competing flows to characterize BBR's inter-CCA fairness. In shallow-buffer scenarios, BBRv1 is quite unfair to all others. No other CCA, not even other versions of BBR, is able to obtain a fair share when competing with BBRv1 (Fig. 5.13a); increasing bottleneck buffer sizes, nevertheless, seems to alleviate this situation substantially. This behavior is presumably due to BBRv1 only backing off when its large in-flight data cap (3×`BDP`) is exceeded (refer Table 1 in [1]), ignoring packet loss unlike BBRv2 and BBRv3. Unlike BBRv1, BBRv2 performs quite well (with high JFI values) across all bottleneck buffer settings (Fig. 5.13b) primarily because of its ability to reach to packet loss. It can equitably share bandwidth with Cubic, making it safer than BBRv1 for deployment in the public Internet where the latter is quite prevalent. BBRv3, despite being only a



(a) *BBRv1 vs. Cubic*



(b) *BBRv2 vs. Cubic*



(c) *BBRv3 vs. Cubic*

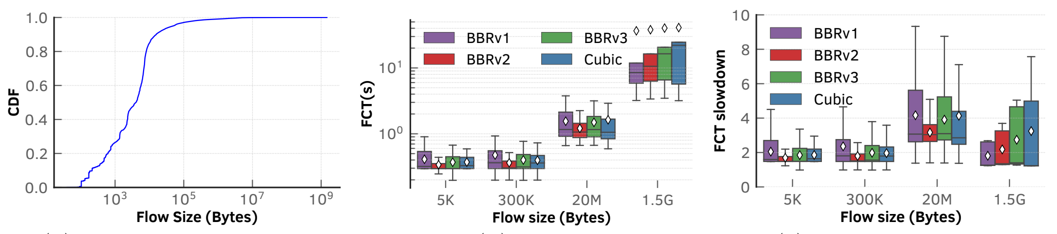
Figure 5.15: *JFI when 1-5 Cubic flows compete against 1-5 (a) BBRv1, (b) BBRv2 and (c) BBRv3 flows.*

minor revision of BBRv2, performs quite poorly, especially against Cubic (Fig.5.13c). In deep-buffer settings, Cubic is able to send more data into the network before experiencing congestion (i.e., packet loss); naturally, it obtains a higher bandwidth compared to shallow-buffer settings and, as a result, the fairness index improves.

Overall, while the design of BBRv3 (which it inherits from BBRv2) represents a substantial progress (in creating a fair and safe CCA), the performance tweaks [62] or optimizations (refer Table 1 in [1]) that Google has introduced in BBRv3 are exactly the opposite. With regards to Google's claims about reduced packet loss, we observe frequent retransmissions (Fig. 5.14) when BBRv3 competes against Cubic, comparable almost to those when BBRv1 competes against Cubic in both deep and shallow-buffer settings.

Thus far we focused on a scenario where a single BBR flow competes with a single Cubic flow for bandwidth on a bottleneck. Previous studies showed that even when competing with several Reno or Cubic flows a single BBRv1 flow was able to grab most of the link bandwidth [26, 141]. We revisit this evaluation where we pit a single BBR flow against several Cubic flows. More specifically, we conduct a series of experiments where we vary the number of BBR flows as well as competing Cubic flows from 1 through 5, and characterize the fairness (measured via JFI) across the competing flows in each experiment (Fig. 5.15). As before, we set the link bandwidth to 100 Mbps, the RTT to 100 ms, and the buffer size to 1×BDP(i.e., 1250 KB). We start all the flows at the same time and run them for 360 s. Fig. 5.15 shows the JFI of the experiments with the three BBR versions competing against Cubic. The top value in each cell is the mean JFI value over time, concatenated over three experiment runs, while the bottom value show the mean throughput shares of BBR and Cubic.

In Fig. 5.15a, JFI values become smaller (i.e., fairness worsens) as we move from top to bottom or left to right. This observation confirms the observations from prior work that even a single BBRv1 flow can outcompete multiple Cubic flows [26, 141]. BBRv2's design completely reverses this behavior (Fig. 5.15b); when we increase the number of Cubic flows fairness improves, emphasizing once again that we can safely deploy BBRv2 in the public Internet. BBRv3, in contrast, resurrects BBRv1's aggressive behavior (Fig. 5.15c); it offers lower fairness than BBRv2, although the



(a) *flows in MAWI trace* (b) *FCTs* (c) *FCT slowdowns*

Figure 5.16: *(a) Distribution of flow sizes in the MAWI trace; Flow completion times (FCTs) and FCT "slowdown" for various flow sizes across different CCAs.*

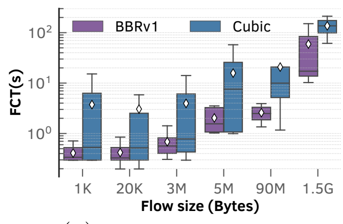
Cubic flows seem to be able to obtain a higher bandwidth when competing against BBRv3 than when competing against BBRv1.

Takeaways. *While BBRv2 makes significant improvements towards achieving equitable bandwidth sharing when competing against loss-based CCAs, BBRv3 seems to undo most, if not all, such improvements. BBRv3 does not equitably share bandwidth with loss-based CCAs such as Cubic, and its behavior in some instances is even worse than that of BBRv1. Despite its ability to react to packet loss, it remains highly unfair to Cubic flows, in shallow buffers.*

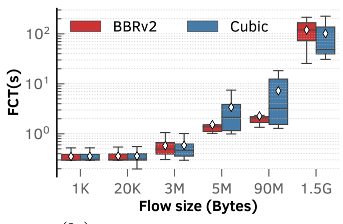
5.2.3 Real-world workloads

We now turn to using realistic traffic workloads to quantify BBR's performance in real-world network conditions. Specifically, we use a distribution of flows comprising both short-lived (i.e., mouse) and long-lived (i.e., elephant) flows. We sample the size of these flows from an empirical distribution of flow sizes observed in the Internet (Fig. 5.16a). The CDF in Fig. 5.16a depicts the flow size distribution from a MAWI trace [67]. We use the Harpoon [174] traffic generator to sample the flow sizes and connection times from this trace data. We fixed the random seed used in the sampling to ensure that the sampled distribution of flow sizes remains the same across the evaluations of all CCAs; we can then compare the performance of the same set of flows across different CCAs and determine which CCA performs the best. We conduct two distinct experiments using the topology shown in Fig. 3.2 with the bottleneck bandwidth set to 100 Mbps, RTT to 100 ms, and the buffer size to `BDP`. In the first, we set all flows to use the same CCA, and in the second, we configure half of the flows to use one of the BBR versions and the other half to use Cubic. We then analyze the flow completion times (FCTs) as well as the FCT "slowdowns" [28] computed by normalizing the measured FCT by the theoretically optimal FCT obtained by taking into account the flow size, the bandwidth, and the RTT.

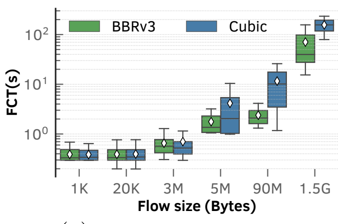
When all flows use the same CCA. In case of BBRv1, the FCTs of short flows, i.e., flows smaller than 300 KB, are comparatively longer than those of the long flows (Fig. 5.16b). The FCT slowdowns for the longest flows in Fig. 5.16c when compared to those of the smaller flows clearly show that BBRv1 prioritizes long flows over their



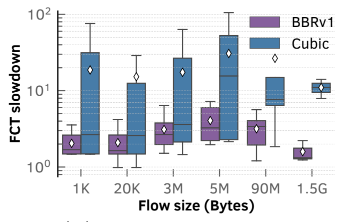
(a) *BBRv1 vs. Cubic*



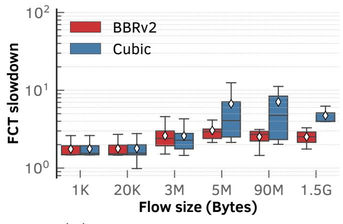
(b) *BBRv2 vs. Cubic*



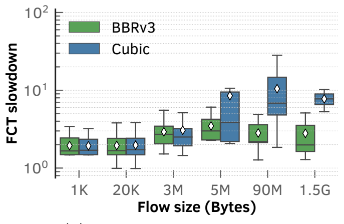
(c) *BBRv3 vs. Cubic*



(d) *BBRv1 vs. Cubic*



(e) *BBRv2 vs. Cubic*



(f) *BBRv3 vs. Cubic*

Figure 5.17: *FCTs and FCT "slowdowns" for flows of varying sizes split between two CCAs.*

shorter counterparts. BBRv2, in contrast, offers lower FCT for short flows than for long flows. Lastly, BBRv3 offers smaller FCT slowdowns for short flows than for long flows, but its FCT slowdowns are consistently higher than those of BBRv2.

When flows are split equally between two CCAs. To evaluate how BBR behaves when competing with Cubic (similar to the earlier evaluations in §5.2.2.3) in realistic traffic conditions, we configure half of the flows in the workload to use one of the BBR versions and the other half to use Cubic. As before, we plot the FCTs and FCT slowdowns of the flows grouped into buckets of different sizes in Fig. 5.17. The FCTs and FCT slowdown values corresponding to the experiment where BBRv1 flows compete with Cubic (Fig. 5.17a and Fig. 5.17d) are not surprising; BBRv1 is quite unfair to Cubic, as amply demonstrated in §5.2.2.3, and its ability to seize bandwidth aggressively from Cubic flows results in substantially smaller FCTs and FCT slowdowns for BBRv1 flows than Cubic flows, regardless of flow size. BBRv2 is far less aggressive to Cubic than BBRv1 (Fig. 5.17b and Fig. 5.17e). Short flows of both CCAs experience similar FCT slowdowns, while long flows using BBRv2 fare better than those using Cubic. BBRv3's performance is similar to that of BBRv2, except in case of long flows. (Fig. 5.17c and Fig. 5.17f) Long flows using Cubic experience higher FCTs and FCT slowdowns when competing against BBRv3 than against BBRv1.

Takeaways. The evaluations confirm the highly aggressive and unfair behavior of BBRv1 towards loss-based CCAs such as Cubic. BBRv2's design significantly improves that status quo, making Cubic flows achieve nearly their fair share when competing with BBRv2 flows. While BBRv3 behaves similar to BBRv2 for short flows allowing both Cubic and BBRv3 flows to experience similar FCT slowdowns, long flows using BBRv3 experience much smaller FCT slowdowns than those using Cubic.

5.3 Summary

The bottleneck bandwidth and round-trip propagation time (BBR) algorithm is a relatively new congestion control algorithm (CCA). BBR's design eschews the typical congestion signals (e.g., transient queue delays and losses) used in the vast majority of prior work. It measures instead bottleneck bandwidth and round-trip times on each `ACK` and paces the sender to retain the delivery rate close to the bottleneck bandwidth and avoid any queue build up. Several studies, however, demonstrated this initial design to be highly unfair to loss-based CCAs such as Cubic. Google responded to the criticisms by evolving the design and releasing newer versions of BBR. BBRv3, released in July 2023, is the most recent version in the evolution of BBR.

Given the increasing volume of traffic that Google has been transitioning to use BBR and its current efforts to include BBRv3 in the Linux kernel, we turned our attention to performing a systematic and rigorous evaluation of BBRv3 in a range of realistic network conditions and traffic scenarios using our CCA benchmarking recipe. Specifically, we checked whether Google's claims of BBRv3's fairness towards other CCAs hold water in our evaluations.

While BBRv3 has evolved substantially compared to the earlier versions, in our evaluations BBRv3 struggles to achieve an equitable bandwidth sharing with competing flows. Even when competing with similar flows (i.e., flows using BBRv3) a small difference in arrival times causes BBRv3 to incur substantial delays in bandwidth convergence. Despite the revisions, optimizations, and bug fixes, BBRv3's highly competitive behavior stifles Cubic on a bottleneck link, particularly in shallow buffer settings. Our benchmarking recipe provides insights on the *crucial* implications that BBRv3's adoption and deployment in the public Internet may have on wider network traffic performance.

6 Case study 2: Network-assisted Congestion Control

A fundamental and long-standing challenge in networking is effective mitigation of congestion on a bottleneck link. The network landscape is continually evolving (examples include programmable data planes, low-Earth-orbit satellite networks, 5G and WiFi 7 for cellular and wireless networks, and plethora of novel applications with varying performance demands), and this evolution, in turn, ushers in innovative CCA proposals from the networking community. The evolution in networking technologies and applications (or systems) also introduce new challenges for CCA design.

Today, an "ideal" CCA must cater to a multitude of objectives including ensuring fair or equitable sharing of network resources (e.g., [141, 197, 198]), reducing delays or RTTs, maximizing throughput, minimizing flow completion times (FCTs), avoiding starvation, tackling TCP Incast (e.g., [35, 78, 92, 199, 200]), and adapting to diverse network settings (e.g., datacenter [14, 201–203] and wireless networks [16, 17]). In this chapter, we use the benchmarking recipe (refer to Chapter 4) to characterize the behavior of a network-assisted CCA design that caters to such evolving networks. By exploiting recent advancements in programmable data planes [204], we are able to design a framework that facilitates designing a multi-objective CCA for the public Internet.

In reviewing prior work on CCAs, we identify four key challenges that any congestion control signaling or notification mechanism, specifically designed for the public Internet, should meet.

Responsiveness The primary goal of a network congestion notification is *prompt* signaling. Quickly notifying a sender of *impending* congestion is crucial for reducing (TCP) timeouts and retransmissions (e.g., DCTCP [78]). Moreover, recent work shows that a *sub-RTT* congestion feedback is essential for guaranteeing high utilization and minimizing FCTs [14].

Stability Detailed telemetry data can curb oscillations in a sender's throughput and support consistent, low queue usage at the bottleneck. If we quickly deliver such rich telemetry data to a (traffic) sender, we can minimize the likelihood of packet drops from buffer overflows as well as network under-utilizations from queue underflows.

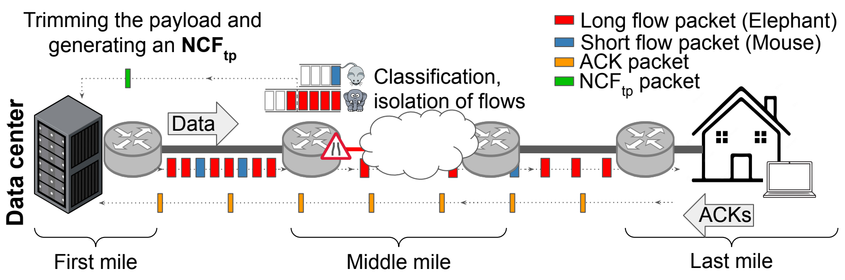

Trimming the payload and generating an $\mathbf{NCF}_{tp}$
Classification, isolation of flows
🟥 Long flow packet (Elephant)
🟦 Short flow packet (Mouse)
🟧 ACK packet
🟩 NCF_{tp} packet
Data center
Data
ACKs
First mile
Middle mile
Last mile

Figure 6.1: *Illustrating the use of `NCF` in congestion scenarios where a rich, sub-RTT feedback can be immensely beneficial.*

Protocol independence The congestion notifications (e.g., ECN) are agnostic to the transport protocol. Any upgrade to this old scheme, incorporating rich feedback, should still strive to be independent of transport protocol. TCP and its variants, and even the modern QUIC implementations can then leverage the signaling, as required, and contribute to improvements in the performance of diverse applications using these transports.

Incremental deployment Most prior work on innovative congestion notification mechanisms focused on data centers or enterprise networks. The public WAN or Internet, in contrast, is not within the control of a single provider or organization. The adoption of any congestion signaling mechanism must allow for incremental or selective (i.e., strategic) deployments of the solution.

We present a simple and robust congestion signaling or notification mechanism, called *network-assisted congestion feedback* (`NCF`), that addresses all these four objectives. At its core, `NCF` exploits the programmability in today's switches to distinguish long (i.e., *elephant*) flows from short (i.e., *mice*) flows (Fig. 6.1). We then isolate the traffic corresponding to the two flow types using the support for priority classes and queues that are available even in commodity networking hardware. When a bottleneck experiences congestion, `NCF` meticulously constructs a congestion feedback signal and sends it *only* to the elephant flow (senders) for throttling them. All of these tasks are performed in the data plane, at line rate, by exploiting the capabilities of a programmable switch; we refer to these data-plane implementations as `NCF`'s data-plane logic ($\texttt{NCF}_{\texttt{dl}}$).

`NCF` provides a quick, i.e., sub-RTT, congestion feedback to the senders of an elephant flow. While prior work on generating explicit congestion signals focused on data centers, we exploit the availability of programmable data planes in the public WAN (e.g., in IXPs and access networks). Even if many end users experience congestion in the last mile, delivering the congestion notification from the last hop prior to the customer premise equipment (CPE) instead of the end-user devices themselves could substantially shorten the control loop delay; such a notification would, for instance, avoid the high delays that are typical in WiFi connections, the common mode of Internet connectivity for end-user devices. Furthermore when congestion occurs far away from users, closer to content providers and origin servers (e.g., [162, 205, 206]),

a quick signaling could substantially shorten the otherwise high-delay round-trip-feedback path over the public Internet.

NCF generates a rich and explicit congestion feedback in the form of NCF telemetry packets (NCF_{tp}). The data conveyed via $\texttt{NCF}_{\texttt{tp}}$ includes link speed, queue size, and an estimate of the count of elephant flows. This rich telemetry data about the conditions at a bottleneck can be exploited by the sender to adapt the congestion window quickly and precisely, at *sub-RTT* reaction times (Fig. 6.1).

The design of NCF is transport-protocol agnostic. Unlike explicit congestion notifications and related work, NCF does not require receiver support; $\texttt{NCF}_{\texttt{tp}}$ packets are sent directly to the sender by the data plane. The sending mechanism can then adapt its response based on the telemetry data delivered by $\texttt{NCF}_{\texttt{tp}}$ as we demonstrate through a simple CCA scheme, which we label NCF's sender logic ($\texttt{NCF}_{\texttt{sl}}$). Our approach enables novel CCA designs for the public Internet that can offer substantial performance and fairness by using the precise, sub-RTT feedback offered by NCF, and $\texttt{NCF}_{\texttt{sl}}$ is simply one point in this solution space.

Our approach can be incrementally or strategically deployed, as required. A content provider serving a large customer base might experience congestion close to the edge of their serving infrastructure (e.g., at upstream links from the "edge" servers) [205]. To alleviate the issue, we can deploy NCF in the first mile, at the content provider or in the upstream network. Similarly, an eyeball AS may deploy NCF to mitigate congestion in the last mile and facilitate fair sharing of network resources by various application traffic. We can also deploy NCF at strategic locations such as colocation facilities and IXPs, which host the points of presence (PoPs) where traffic from a content provider's infrastructure connect to various ISP networks [207–213]. Any network can deploy NCF without requiring support or coordination from another network. The deployments may also catalyze or motivate other networks to adopt NCF, as each deployment may essentially move the bottleneck elsewhere along the path traversed by application traffic to adopt NCF.

Our signaling mechanism may resemble the idea of source quench (SQ) [214] or choke messages [215], but it is *without* the issues (e.g., [216, 217]) that resulted in the deprecation of those old ideas [218]. SQ was weak and unfair, for instance, because it sent one (ICMP) message per congested packet without regard to flow type; besides, it carried little information about the bottleneck's characteristics. NCF instead focuses on elephant flows that can react to congestion and carries enough information concerning the bottleneck to the sender. The sender, hence, can back off immediately to its fair share; NCF, hence, obviates the need for heuristics to approximate the extent of congestion. SQ messages are also easy to filter or spoof. NCF, in contrast, generates valid, low-overhead ACK-like packets that middleboxes are unlikely to filter, and it preserves existing TCP sequence and acknowledgment numbers, which makes spoofing harder. We also allay security concerns, e.g., switch overload, by generating $\texttt{NCF}_{\texttt{tp}}$ purely in the data plane and mitigate DDoS vulnerabilities through safeguards such as nonces, verification, and damping. Likewise, in the data center context, prior work have investigated generating precise congestion feedback using programmable hardware (e.g., XCP [76], HPCC [81], FastTune [201], Bolt [14]) and quickly sending this feedback using `ACK`s traveling towards the sender (e.g., FastTune [201], ExpressPass [202]). While the idea of eliciting explicit network support for congestion control originated about two decades ago (e.g., [76, 77, 219]), we recently resurrected this idea, on account of recent advances in networking, in a preliminary version of this work at a workshop in 2019 [3]. Indeed the most recent work Bolt [14] followed up our call for the community to rethink explicit congestion signaling. The scope of the signaling or feedback in prior work has been, nevertheless, rather narrow (e.g., in data centers, or requiring support at every hop along the path, or with sweeping changes across the network stack), which renders the approach either insufficient or impractical for deployment at scale in the public Internet.

The contributions of this chapter are as follows.

- Using the benchmarking recipe, we show the benefits of a new congestion control framework (`NCF`) for designing CCAs for the public Internet. `NCF` exploits a rich and explicit congestion signal, $\texttt{NCF}_{\texttt{tp}}$, that we generate using widely available programmable data planes.
- We show, using different benchmarks and across diverse network conditions, that `NCF` ensures high fairness while minimizing queue usage in various scenarios (§6.2.2), and offers these benefits even when competing flows have dissimilar RTTs and regardless of bottleneck buffer sizes (§6.2.2.2). Compared to three widely used CCAs, `NCF` offers the lowest FCTs to short flows even in challenging multi-bottleneck scenarios (§6.2.2.4).

6.1 Related Work

The extensive body of prior work on congestion control, as showcased in Chapter 3, highlights the numerous CCA schemes designed to target diverse network environments, meet various application requirements, and leverage advancements in network infrastructure. Huang et al. [220] provide a comprehensive survey of this space.

Traditional CCA proposals, however, suffer from two key shortcomings: They depend on imprecise signals (e.g., loss and delay), and rely on the receiver to reflect the signal back to the sender. This dependency on the receiver results in a delay of at least one RTT before the sender can apply corrective measures to its delivery rate. Cubic [47], the default Linux kernel CCA, for instance, detects congestion by inferring losses based on received `ACK`s. It is, as a result, typically slow to react to congestion and consequently fills up the bottleneck queue [221]. BBR, a relatively modern CCA, in contrast, combines window-based and rate-based mechanisms to minimize queue build-up [35]. BBR has seen widespread adoption [19–21], and our evaluation of `NCF` compares its performance to BBR, Cubic, and NewReno [222], an earlier loss-based CCA.

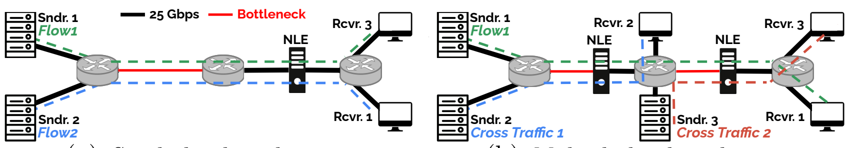
The idea of eliciting explicit network support for congestion control derives its roots from DECBit [72], RED [73], and ECN [74]. These approaches broadly rely on two techniques to signal congestion: mark one or more bits in the packet headers that the receiver will reflect back to the sender (e.g., ECN [74]), or pre-emptively drop packets before the buffer overflows (e.g., RED [73] and CoDel [150]). The congestion signals used are quite narrow in scope (for instance, they do not indicate the degree of congestion), and some require receiver support. Recent advancements, such as the Low Latency, Low Loss, Scalable throughput (L4S) framework [85], improve congestion quantification and reduce notification delays, but their deployment still requires support from the receivers. L4S combines a scalable CCA (e.g., TCP Prague [86]), Accurate ECN (AccECN) [87], and a dual-queue AQM [88] that isolates L4S and non-L4S traffic at the bottleneck. The solution, however, does not offer flow-type isolation and sub-RTT signaling.

Other early work on in-network solutions for congestion control pursued a "clean-slate" approach. XCP, for instance, used a congestion header to communicate flow state (e.g., `cwnd` and RTT) between routers and receivers [76], while RCP required routers along the path to compute per-flow rates [77]; both required support at each hop along the path. More recent efforts, however, have shifted focus to datacenter environments, (e.g., DCTCP [78] and TIMELY [79]). TIMELY [79], for instance, relies on accurately estimating RTT; feasible in datacenter environments where end hosts may support hardware time stamping. The advent of programmable switches has further led to exploring novel uses for In-band Network Telemetry (INT) data provided by these switches. HPCC, for instance, enriches the congestion signal with rich data such as the current load on the switch egress port, but it, unfortunately, relies on receivers to reflect this signal back to senders [81]. `NCF` is inspired by prior work, but extends them in non-trivial ways: It does not require receiver support, offers sub-RTT signaling, and is suitable for the public Internet.

Related approaches investigated various methods of directly notifying senders of congestion, albeit in the datacenter context. QCN [75] was an early effort to provide direct feedback to the sender, though its congestion notifications were restricted to the L2 domain. FastLane, in contrast, requests switches to send high-priority notifications directly to senders as soon as a packet drop is detected, however, waiting until a packet drop might already be too late [80]. RoCC [83] and BFC [84] also use sub-RTT congestion signaling at intermediate nodes. RoCC tracks elephant flows at the switch and calculates each flow's fair rate using the queue size, but was designed specifically for RDMA traffic.[1] BFC is a per-hop, per-flow control mechanism that requires dedicated queues per-flow for optimum performance [83]. It also requires upstream switches to pause queues when a bottleneck sends congestion signals, which is impractical in WAN environments. Bolt uses precise congestion information from the switch to proactively control a sender's rate [14]. Although conceptually similar to `NCF`, Bolt is designed for datacenters (with microsecond delays), relies on support from endpoints (for proactive ramp-up) and does not take into account that mice flows cannot be congestion controlled, though they can easily fill up queues [113, 223].

More recently, Zhuge [16] shortened the control loop in wireless networks by modifying the access point (AP) to inform the senders of the delay experienced by its packets

[1]We could not evaluate RoCC since it is not publicly available.



(a) *Single bottleneck setup* (b) *Multiple bottleneck setup*

Figure 6.2: *Network testbed for performing head-to-head comparisons of* `NCF` *against other CCAs.*

at the AP. Zhuge is, nevertheless, designed to handle congestion in the last mile and their approach of estimating packet delays is not applicable to the WAN scenarios that we consider.

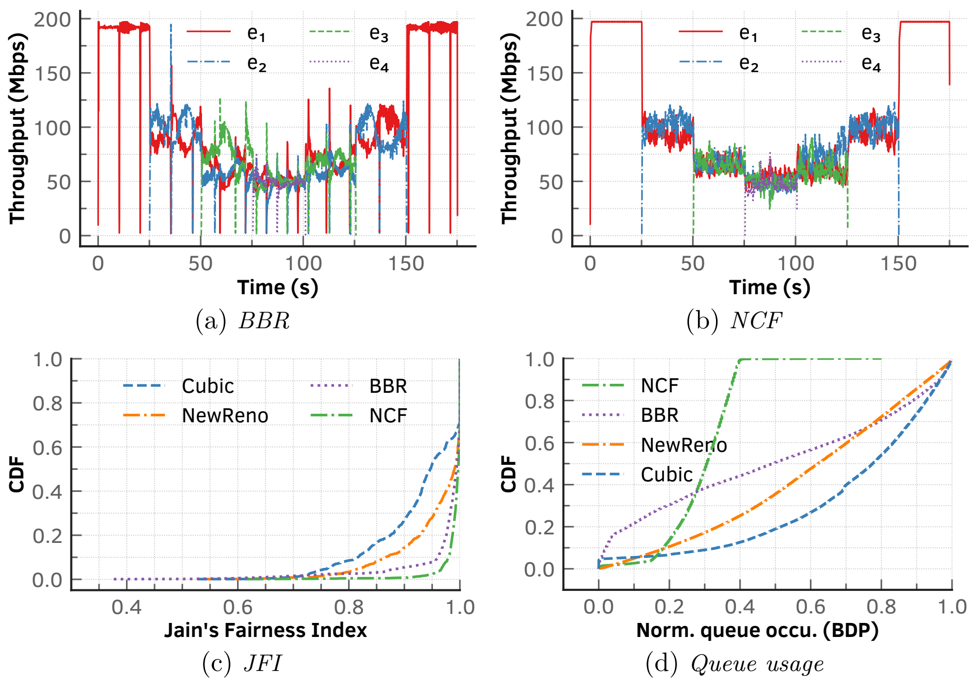
6.2 Evaluation

We realized the `NCF` prototype using Intel's Tofino1 chipset [194]. Below, we discuss how we used the prototype in line with the benchmarking recipe to characterize `NCF`'s performance and demonstrate its benefits. Although we do not evaluate `NCF` based on every question in the recipe, the parts we do explore still offer valuable insights into how our framework can guide designers or evaluators of similar congestion control approaches towards well-scoped, objective-driven analyses.

6.2.1 Experimental setup

Testbed We built the testbed using three APS Networks BF6064X-T programmable switches with Intel's Tofino1 chip in a dumbbell topology as shown in Fig. 6.2. We used eight Dell R6515 blade servers each with 16 cores and 128 GiB of RAM, running Linux kernel version 5.4 (Debian 10). We configured some as traffic senders, a few as network latency emulators (NLEs), and the rest as receivers. We installed one or two Broadcom NetXtreme BCM5720 25 GbE NICs, as required, and used 25 Gbps DACs for interconnecting the servers and switches. We varied the capacity of the bottleneck link (marked in red in Fig. 6.2) as needed for the different experiments using the traffic shaping feature of the Tofino switches. The base RTT between sender-receiver pairs in the testbed, without any added delay, was less than 0.06 ms. We, nevertheless, introduced different one-way delays using DEMU [224], a DPDK-based solution with low overhead and high accuracy, at the NLE(s). We evaluated various CCAs in both single bottleneck (Fig. 6.2a) and multiple bottleneck (Fig. 6.2b) settings and using a range of bottleneck buffer configurations.

Traffic generators and workloads We implemented $\texttt{NCF}_{\texttt{s1}}$ using F-Stack [225], a `DPDK` based framework to bypass the kernel overhead. We generated elephant flows using `iperf` (v2). For large-scale evaluations, we generated workloads based on the data from Parsonson et al, [226] which follow the widely used traffic traces from Facebook [227]. In particular, we derived the flow size and inter-arrival time distributions



(a) *BBR*

(b) *NCF*

(c) *JFI*

(d) *Queue usage*

Figure 6.3: *When we stagger the arrivals and departures of four elephant flows* (e_1, e_2, e_3, e_4) *at a bottleneck link of 200 Mbps by 25 s, (a) the flows struggle to achieve their fair shares in BBR, while (b) they converge quickly to their fair shares in* `NCF`*;* `NCF` *offers (c) higher fairness and (d) smaller bottleneck buffer occupancy than the other CCAs.*

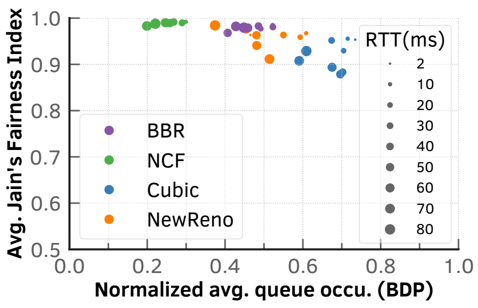
(Weibull and Lognormal, respectively) from the traces, and then used Harpoon [174] to generate traffic with 400 concurrent senders using samples from these empirical distributions.

Choice of CCAs We compare `NCF` against the two widely used CCAs, namely CUBIC [47] and NewReno, as well as the relatively modern BBRv1 [35] from Google (henceforth referred to as just BBR in this chapter). Our choice of CCAs is influenced by the availability of CCA implementations on F-stack.

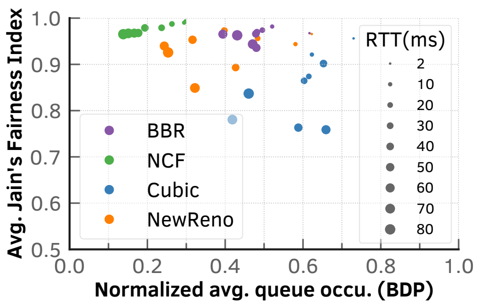
6.2.2 Adapting to cross traffic

6.2.2.1 Staggered flows

We now evaluate `NCF`'s ability to facilitate an equitable bandwidth sharing between flows. To this end, we run a staggered-flow experiment (similar to that of HPCC [81]), where we introduce four flows, one after another, in a single-bottleneck setting (Fig. 6.2a). More precisely, we add a new flow to the testbed every 25 s until we reach four concurrent flows, and then remove one flow every 25 s until we have only one left. The four flows are *similar*: They use the same CCA and have same (20 ms) RTT. We



(a) *JFI vs Queue usage; 25 s*



(b) *JFI vs Queue usage; 5 s*

Figure 6.4: *NCF scores high on fairness with minimal queue occupancy across a range of bottleneck link delays, even if we reduce the arrival & departure intervals from* 25 s *to* 5 s.

measure how well the flows share the bottleneck bandwidth (of 200 Mbps) as they join and leave.

The staggered flows struggle to acquire their fair share when using BBR (Fig. 6.3a), both during flow arrivals and departures. More importantly, changes in the number of active flows substantially affect the throughput of existing flows, and, consequently, the bottleneck queue occupancy (Fig. 6.3d). Cubic and NewReno show similar behavior. NCF, in contrast, allows the flows to converge quickly to their fair shares (6.3b). We also observe substantially smaller throughput fluctuations in NCF flows than those of the other CCAs. We periodically (i.e., every 80 ms) compute the JFI for concurrent flows, and the CDF of the JFI of the flows over time (Fig. 6.3c) shows that flows share bandwidth more equitably when using NCF than when using other CCAs. NCF also minimizes the bottleneck queue usage (Fig. 6.3d): The median queue occupancy for BBR, NewReno, and Cubic are at least 66% more than that of NCF.

To analyze the implications of latency on the fairness and bottleneck queue occupancy, we repeat the staggered-flow experiment across a range of bottleneck delay settings (Fig. 6.4a). NCF scores high on the Jain's fairness index with minimal queue use compared to all other CCAs. BBR also scores high on fairness, but with queue occupancies nearly twice as large as that for NCF. NewReno and Cubic show significant variation in fairness and queue occupancy, and perform poorly at low delays. We then reduce the time between flow arrivals and departures—to 5 s instead of the 25 s used earlier—to evaluate how the CCAs perform when network traffic volume (across the bottleneck link) changes quickly. We observe that NCF still scores high on fairness with low queue occupancies (Fig. 6.4b). The other CCAs perform much worse on fairness—with NewReno and Cubic scoring sometimes as low as 0.85 and 0.75, respectively, on fairness—when traffic volume changes rapidly.

Takeaways. *NCF enables fair sharing of the bottleneck bandwidth across flows while minimizing bottleneck buffer occupancy, across a wide range of bottleneck link delays and regardless of whether flows arrive at and depart from the bottleneck link rapidly or gradually.*

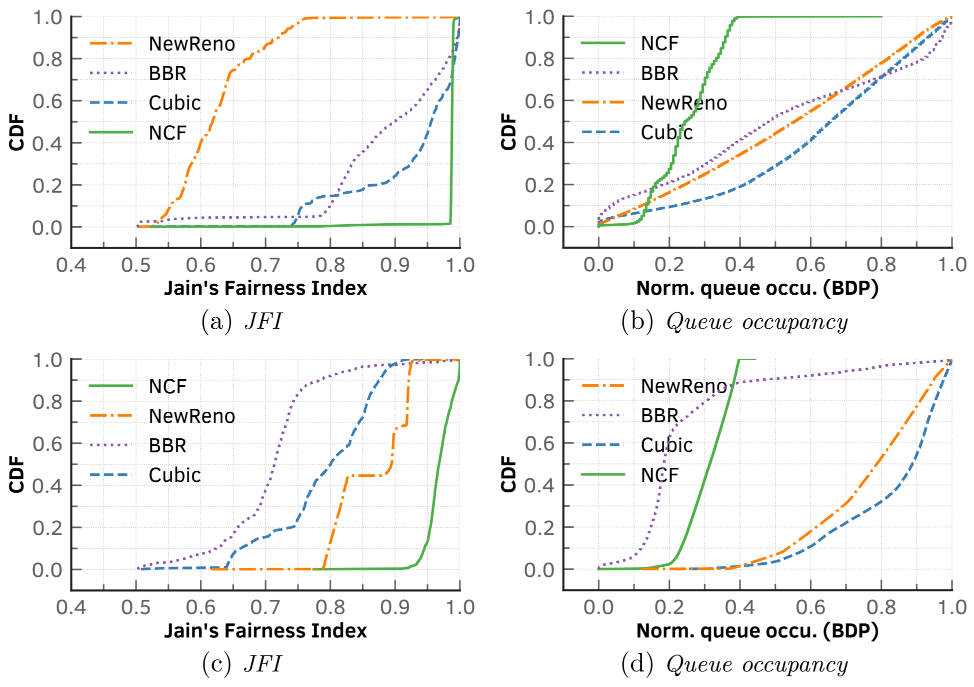
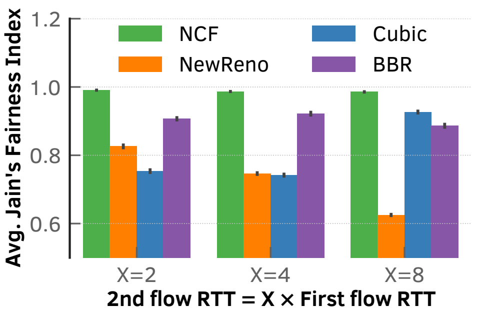


Figure 6.5: *Shallow buffer scenario:* `NCF` *offers (a) high fairness and (b) low queue occupancy compared to other CCAs for two dissimilar elephant flows—one with* 10 ms *and the other with* 80 ms *RTT. Deep buffer scenario:* `NCF` *offers (c) excellent fairness and (d) low queue occupancy compared to other CCAs for two dissimilar elephant flows—with* 10 ms *and* 80 ms *RTTs.*

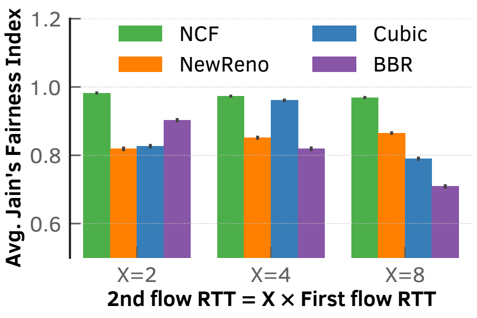
6.2.2.2 Dissimilar flows: RTT fairness

Evaluations concerning fairness of CCAs in prior work (e.g.,[26, 139]) typically consider flows with similar RTTs. In practice, outside the realm of datacenters, flows typically experience different RTTs. Such RTT differences between flows sharing a bottleneck link, even if they all use the same CCA, make it challenging for the CCA to ensure a fair share of bandwidth across the flows. The signals typically used by a sender (e.g., packet loss or delay), for determining how much it can send and/or how fast, must traverse the path from the bottleneck to the receiver, which then echoes it back to the sender. The RTT, hence, dictates a sender's response time: the larger the RTT the slower the sender reacts to changes in network conditions.

We now evaluate the behavior of various CCAs when *dissimilar* flows contend for bandwidth at a bottleneck. To this end, we stagger the starts of two elephant flows to be 15 s apart, and we measure the fairness and queue occupancy across different CCAs. The first flow has a smaller RTT than the second flow, and we set the RTT of the latter as an integral multiple of that of the former. Since the size of the bottleneck queue can be large or small depending on the location of the bottleneck (e.g., whether it is in close proximity to a CDN's edge or content provider's infrastructure, or far away), we repeat our evaluation under both *shallow* and *deep*



(a) *JFI in shallow buffer settings*



(b) *JFI in deep buffer settings*

Figure 6.6: *When we vary the RTT of the second flow as a multiple of the first (with* 10 ms *RTT),* `NCF` *offers better fairness than other CCAs in both (a) shallow and (b) deep buffer configurations in all tested scenarios.*

buffer configurations. We use 250 KB, the bandwidth delay product (BDP) for flows with 10 ms RTTs, for a shallow buffer, and we set the size of deep buffers to one of $\{500\,\text{KB}, 1000\,\text{KB}, 2000\,\text{KB}\}$ corresponding to BDPs of 20 ms, 40 ms and 80 ms, respectively. We set the bottleneck buffer to the BDP of the flow with the shorter RTT for the shallow buffer and the longer RTT for the deep buffer scenarios, and we set the bottleneck bandwidth to 200 Mbps.

In the shallow buffer configuration, even when the RTTs of the flows differ by a factor of eight, `NCF` ensures that the flows share the bandwidth equitably (Fig. 6.5a). None of the other CCAs offer comparable fairness, and NewReno performs the worst—the smaller-RTT flow enjoys a substantially larger share than the other. The fairness offered by `NCF` is rooted in its ability to keep the queue occupancy small (Fig. 6.5b), despite the dissimilar RTTs of the flows as well as the sub-RTT feedback. One aspect that benefits `NCF` is that the feedback delay for both competing flows is similar, which should be the case for most of the envisioned deployment scenarios. Next, we keep the 10 ms RTT of the first flow intact, but vary the RTT of the second one as a multiple of the first. Regardless of the differences in RTTs, per Fig. 6.6a, `NCF` ensures a fair share between the flows.

To emulate the deep buffer scenario, we set the bottleneck buffer to the BDP of the flow with larger RTT. `NCF` once again assures fairness (Fig. 6.5c) despite the vast difference in RTTs between the flows, i.e., 10 ms and 80 ms. Its queue usage is slightly larger than that of BBR, albeit its fairness is substantially better than BBR (Fig. 6.5d). As with the shallow-buffer experiment, we vary the RTT of the second flow as a multiple of the first one, and measure the fairness of CCAs in each configuration. Regardless of how dissimilar the RTTs of the flows are, Fig. 6.6b shows that `NCF` offers better fairness than the other CCAs in all configurations.

Takeaways. Regardless of how dissimilar flows are (with respect to their RTTs), `NCF` *guarantees fairness of flows under both deep and shallow buffer configurations.*

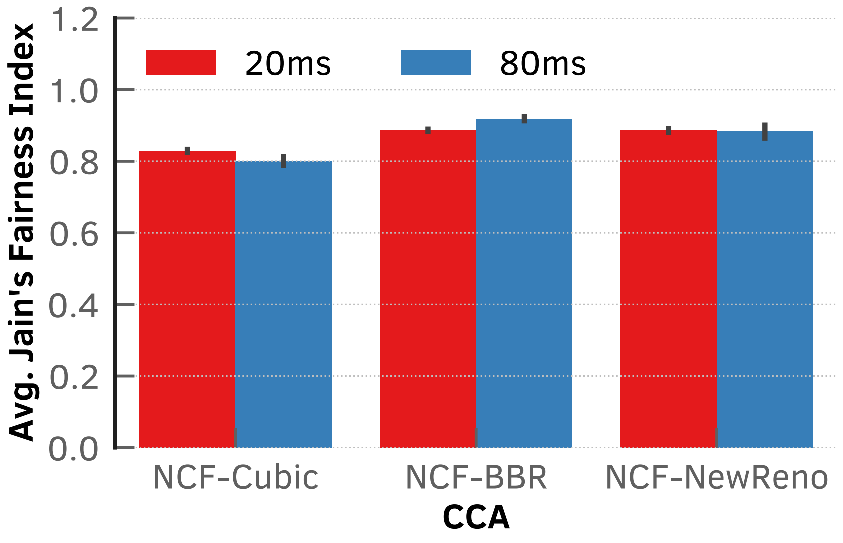
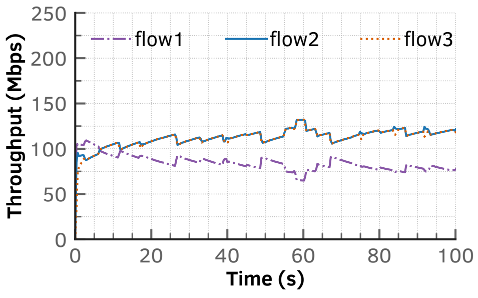


Figure 6.7: *JFI in inter-CCA scenarios. `NCF` does not harm other CCAs in the WAN, but it backs off more quickly than other CCAs, losing in the process some of its bandwidth share.*

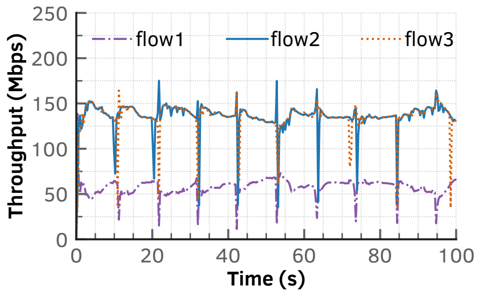
6.2.2.3 Dissimilar flows: Inter-CCA fairness

When an `NCF` flow competes with a non-`NCF` (i.e., Cubic, or BBR, or NewReno) flow for bandwidth, the former acquires less than its fair share of the bottleneck bandwidth, even when the flows experience similar RTTs (Fig. 6.7). This outcome should not be surprising, since `NCF` reacts to the buffer filling up much quicker than any of the other CCAs, owing to its short control loop. Naturally, `NCF` backs off faster than other CCAs when the buffer fill up. Although `NCF` loses some of its bandwidth share, it fares well against BBR, which relies on monitoring the RTT to pace the sender, even though the latter is known to be aggressive [26]. `NCF` becomes reasonably competitive when we increase RTTs (say, to 80 ms) to capture delivery over long-distance links. We can, hence, adopt learning-based methods to help `NCF` adapt to the network conditions [83, 92]. In the scenarios where we typically expect `NCF` to be deployed, including edge servers of CDNs and cloud providers, we can easily monitor `NCF`'s performance and safely customize or tune the protocol on the fly. As such, `NCF` does not threaten the performance or stability of competing non-`NCF` flows. We leave the optimization of `NCF` to cater to specific network conditions, deployment strategies, and evaluation of learning-based approaches for fine-tuning `NCF` to future work.

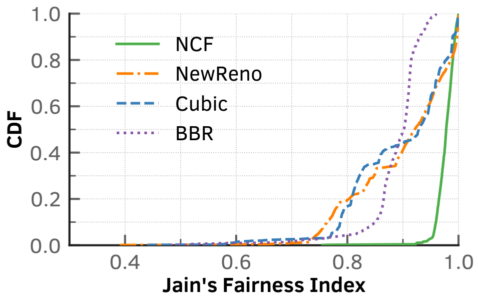
Takeaways. *`NCF` can be safely deployed in the Internet. Though it loses some of its fair share to competing CCAs, it performs competitively in WAN settings, highlighting the opportunities for research into CCA optimizations.*



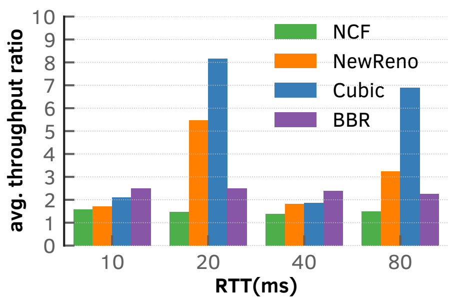
(a) *NCF; RTT: 40ms*



(b) *BBR; RTT: 40ms*



(c) *JFI*



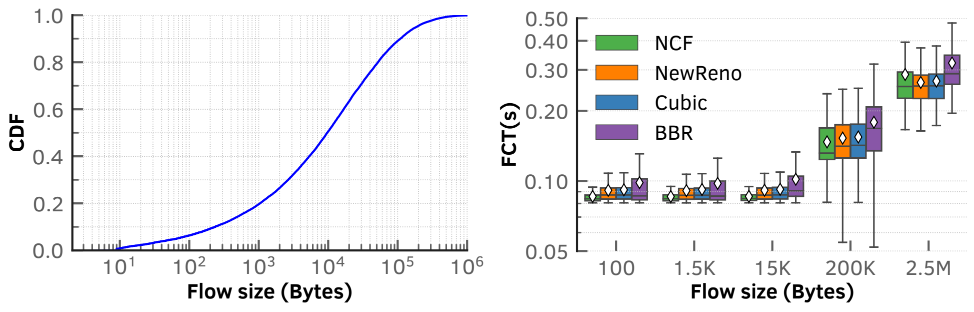
(d) *Avg. throughput ratio*

Figure 6.8: *For elephant flows in a multiple bottleneck scenario (Fig. 6.2b), (a)* `NCF` *enables flows to achieve a fair share while (b) BBR struggles to offer fairness.* `NCF` *also achieves a (c) high Jain's fairness index, and (c) near ideal average throughput ratios compared to other CCAs under various RTT settings.*

6.2.2.4 Multiple bottleneck scenarios

Prior work (and our earlier evaluations) demonstrated that a flow that traverses multiple bottlenecks (marked in purple in Fig. 6.2b) suffers substantial throughput degradation compared to single-bottleneck flows (marked in blue and tawny) [110]. Below, we evaluate how an `NCF` flow performs under such conditions by emulating the parking-lot topology with the "too many red lights" problem [110]. For this evaluation, we configured the NLEs (§6.2.1) such that all flows have the same (40 ms) RTT.

When all three flows in Fig. 6.2b are elephant flows, we observe that the multi-bottleneck (`NCF`) flow, in purple, receives less throughput than either of the single-bottleneck (`NCF`) flows (Fig. 6.8a). When the flows use BBR, per Fig. 6.8b, the multi-bottleneck flow struggles to even receive a quarter of the bottleneck bandwidth. Cubic and NewReno exhibit similar behavior. `NCF` clearly outperforms all these CCAs with respect to fairness (Fig. 6.8c). We show the ratio of the average throughput of the single-bottleneck flows to that of the multi-bottleneck flow, which evaluates the ability of a CCA to provide an equitable share to the multi-bottleneck flow [30], for the various CCAs in Fig. 6.8d. The ratios in this figure demonstrate that `NCF` consistently offers a ratio close to 1, which is the ideal value, for a wide range of RTTs.



(a) *Flows in Facebook workload* (b) *FCTs; RTT: 40ms*

Figure 6.9: *In a multiple bottleneck settings with flow sizes sampled from the Facebook workload,* `NCF` *offers the lowest FCTs for the short flows in a multiple bottleneck scenario.*

6.2.3 Real-world workloads

Next, we evaluate `NCF` using a realistic workload from Facebook. We replace each of the three flows in Fig. 6.2b with a distribution of heavy-tailed flows (shown in Fig. 6.9a), based on the Facebook workload [227]. More concretely, we use Harpoon [174] to generate flows (with each sender generating up to 400 concurrent flows) with the flow sizes and inter-arrival times sampled from the empirical distributions of corresponding metrics from the Facebook workload (refer §6.2.1). All flows have 40 ms RTT, regardless of whether they are single-bottleneck or multi-bottleneck flows. Per Fig. 6.9b `NCF` offers the lowest FCTs of all CCAs for the short flows in bins labeled "100", "1.5K" and "15K" (indicating sizes in bytes).

Takeaways. *Even in challenging multiple bottleneck scenarios,* `NCF` *ensures fairness across all flows. In our evaluations using the widely used Facebook workload,* `NCF` *offers the lowest FCTs among all other CCAs for short flows, which represent interactive or latency-sensitive applications.*

6.3 Summary

In this chapter, we introduced a rich, sub-RTT congestion signaling mechanism for use in the public Internet, combined with a CCA or sender logic ($\texttt{NCF}_{\texttt{sl}}$) at the end hosts that can exploit this feedback. Using the benchmarking recipe, we showed through rigorous evaluations how well this scheme fares against widely used CCAs in the Internet and how it equitably shares bandwidth with other flows in a range of network conditions.

Our evaluations showed that `NCF` is robust and safe for deployment in the public Internet. The need to guarantee low-latency to applications is quite evident from network operators' interests in approaches such as L4S. Emerging applications are, however, highly demanding, than ever before [228, 229]: They are not only demanding low latencies, but also of high bandwidths, and these requirements might also vary over time, i.e., over the course of a data transfer or session. `NCF`'s ability to isolate mice and elephant flow types dynamically would be crucial in meeting the stringent performance requirements of such applications. In today's Internet where CDNs, cloud, and content providers account for most of the Internet traffic, `NCF`, which only requires sender-side support, offers a new, practical, safe, and robust framework for experimenting with congestion control.

7
Summary & Future Directions

In this dissertation, we highlight the necessity of rigorous and reproducible evaluation of congestion control algorithms (CCAs), particularly those intended for deployment in the complex and dynamic environments that characterize the Internet. The continual evolution of networking infrastructure presents both ongoing challenges and novel opportunities for CCA design, necessitating careful and systematic assessment.

Given the Internet's critical role in global communication and the central function of CCAs in maintaining the Internet infrastructure's stability and fairness, it is imperative that the behavior of different CCA schemes be thoroughly understood through reproducible experimentation, even when such CCAs are designed to cater to an array of various network conditions. Such evaluations not only provides the networking community, including operators and end users, with greater confidence in a given design, but also offers developers valuable insights to inform the iterative refinement of a CCA during the design process.

We make significant strides in showcasing the challenges with current CCA evaluation approaches and how the lack of consensus in congestion control evaluations creates significant hurdles for the broader networking community, including the resulting reproducibility crisis. We address these challenges by designing a robust and easily reproducible evaluation platform, the *CCA benchmarking recipe*. By distilling past work on evaluating CCAs into three fundamental questions concerning the behavior of CCAs, we demonstrated how answering these questions offers a rigorous characterization of a CCA's behavior in a wide range of network conditions, and used the recipe to highlight several new performance insights that were missed in the original introductions of several CCAs.

7.1 Summary

Throughout this dissertation, we have sought to underscore the importance of rigorous CCA evaluations by surveying prior work and presenting targeted case studies.

Openness and reproducibility in congestion control evaluations. In Chapter 3, we strived to answer our first and second research objectives: *Examine the current state of congestion control evaluations, with a focus on identifying and addressing the challenges associated with reproducibility* and *Analyze how the lack of consensus in* *congestion control evaluations creates significant challenges for the broader networking community.* We first highlight the reproducibility crisis resulting from decades of a lack of consensus in CCA evaluation approaches by surveying state-of-the-art work on CCAs. We found that the evaluation scenarios, conditions, and even metrics used in prior work differ vastly, often with little or no justifications on these choices.

Next, we underscored the discrepancies that may arise due to such varying approaches in evaluations and presented several examples to illustrate how overlooking edge cases can obscure CCA performance insights. We initially investigated three modern CCAs, two of which have been deployed in the public Internet, that reveal unexpected performance challenges along dimensions that the designers of those respective CCAs did not explicitly evaluate, or did so in some limited capacity. We found, for instance, that Copa typically leaves nearly a third of the bandwidth unused, resulting in poor utilization of the bottleneck, while PCC Vivace flows typically fill up the bottleneck queue when competing with flows experience different RTTs. Sage, an emerging, learning-based CCAs that uses insights from prior CCAs to develop automatically better performing congestion reaction and/or avoidance policies, fails to recover and fully utilize the available bottleneck bandwidth under sudden bandwidth changes, regardless of the size of the bottleneck queue.

These findings, along with observations made from prior research, underscore the need for an open, rigorous, and reproducible evaluation framework for CCAs. Such a framework should establish minimal yet fundamental set of evaluation criteria that *any* CCA evaluation must include, along with clear rationales. Additionally, we motivate the importance of having such a benchmark for CCAs by giving an overview of how benchmarking has helped advance the field in other areas, and posit what role such a benchmark can play in the evaluation and development of CCAs.

A recipe for benchmarking congestion control algorithms. Using the insights learned in Chapter 3, we presented our design of such a CCA benchmark, which we called the *CCA benchmarking recipe*, that focuses on the objectives (or intents) of the evaluations, rather than the tests that comprise such an evaluation. The recipe distills, adapts, and expands decades of prior work on CCA evaluations into three succinct fundamental questions concerning a CCA's behavior:

1. How does a CCA cope with the performance unpredictability of the end-to-end path?
2. How does a CCA adapt to cross-traffic?
3. How does a CCA handle workload dynamics?

By distilling past work on evaluating CCAs into three fundamental questions, we demonstrated how answering these questions offers a rigorous characterization of a CCA's behavior in a wide range of network conditions. We use the recipe to highlight several new performance insights that were missed in the original introductions of seven Internet CCAs. Selected highlights of the insights we were able to obtain are as follows.

- While changes in the bottleneck buffer sizes do not substantially alter the fairness properties of most CCAs, it has a pronounced impact on their convergence times (to their fair shares) and retransmissions. For instance, the latest BBR version, BBRv3, shows the worst convergence times in networks with deep (bottleneck) buffers, and both BBRv1 and BBRv3 suffer from higher retransmissions in comparison to loss-based CCAs such as Cubic.

- We note that a majority of the CCAs are unable to cope with sudden bandwidth changes and cannot quickly discover available capacity and converge to it. For example, Sage completely fails to recover after a sudden bandwidth that require it to discover excess available capacity. Such changes may be typical in cellular or WiFi networks where one of the hosts could be a mobile end-user device.
- An important property of a CCA's deployability in the Internet is its ability to fairly share the available bottleneck bandwidth when contenting with various types of cross-traffic using different CCAs, and this is typically discussed in terms of friendliness towards loss-based CCAs (e.g., Cubic). Although the BBR variants' aggressiveness towards Cubic has been well-documented [1, 26], we were surprised to discuss that PCC is even more aggressive, using almost 100% of the bottleneck capacity when competing with Cubic, leaving almost nothing for the Cubic flow.
- To accurately emulate a CCA's behavior in real-world network environments, we leveraged several Internet traffic distributions based on MAWI traces. Our analysis revealed that a CCA's performance is significantly influenced by workload intensity, and hence, relying on a single traffic distribution (or only long-lived flows) may yield misleading insights. Consequently, it is essential that evaluations of CCA performance incorporate a range of workload intensities to capture and reflect the variability in observed behavior.

Our benchmarking recipe offers designers quick insights into the shortcomings of CCAs, and allows operators to compare and contrast, and, crucially, understand the implications of introducing a new CCA in the public Internet.

Examining the performance of modern congestion control approaches. In Chapters 5 and 6, we use the CCA benchmarking recipe to further examine the performance of the BBR variants, which have been deployed by various CDNs. Additionally, we use the recipe to reveal several new insights into the performance of a new, network-assisted CCA.

For the BBR variants, we show that although BBRv3 somewhat addressed the well-documented issues of aggressiveness and unfairness in BBRv1, it's behavior is worse than that of BBRv2. We also highlighted that BBR's bias towards long-RTT flows has not changed in BBRv3. Additionally, in our evaluations, BBRv3 struggles to achieve an equitable bandwidth sharing with competing flows. Even when competing with similar flows (i.e., flows using BBRv3) a small difference in arrival times causes BBRv3 to incur substantial delays in bandwidth convergence. Despite the revisions, optimizations, and bug fixes, our recipe was able to highlight that BBRv3's highly competitive behavior stifles Cubic on a bottleneck link, particularly in shallow buffer settings. Had the benchmarking recipe been available during the development of BBRv3, it could have provided valuable insights to the designers, potentially informing design choices and preventing the performance regressions observed between BBRv1/BBRv2 and BBRv3.

For the network-assisted CCA (NCF), we demonstrated the potential advantages of incorporating network telemetry data into congestion control approaches. Using our benchmarking recipe, we assessed NCF's performance relative to widely deployed Internet CCAs, highlighting its ability to achieve equitable bandwidth sharing across a variety of network conditions. NCF exhibited robust performance in both multi-RTT and inter-CCA environments, and consistently achieved the lowest flow completion times (FCTs) for short flows, even under complex scenarios involving multiple bottlenecks.

7.2 Future Directions

The work in this dissertation unifies evaluations into succinct, fundamental questions and serves as a starting point in identifying the minimal set of evaluation criteria that any CCA evaluation must include. Such an approach not only helps explain the *why* behind evaluations, but could also help the community move towards standardizing CCA evaluations. An open and standardized framework will help the community not only objectively evaluate their congestion control approaches but also analyze trade-offs and systematically identify opportunities to evolve designs when deployment environments or network conditions change.

We briefly discuss some future directions below.

7.2.1 Exploring the parameter space

A key challenge in proposing a comprehensive CCA evaluation is the intractability of exhaustively enumerating the (high dimensional) parameter space. Many factors influence the behavior of a CCA ranging from end-host system settings (e.g., kernel configuration and NIC optimizations) to network configurations (e.g., buffer size, delays, and queuing discipline) to traffic conditions (e.g., mixing of different flow distributions). Exploring each such parameter and considering all possible values that it can take quickly results in a combinatorial explosion of the parameter space. Given the recent proliferation of learning-based approaches in networking [92, 94, 230], a natural question that follows is whether we can use modern learning methods to sample the high-dimensional parameter space for CCA evaluations.

7.2.2 Explaining CCA behavior

One of the advantages of a standardized evaluation is that a designer can easily infer where, i.e., along what dimensions, their CCA fares poorer compared to others. Advances in interpretable machine learning models can perhaps aid in automatically constructing counter examples of situations where a CCA fares poorly such as those described in [126, 127]. CCA designers can quickly investigate the test scenarios and rectify the problem in the design. Future research can also focus on specific instances where a prior work diverges and demonstrate how their designs fare better than the prior work. Engineers and practitioners can also use such tools to determine where (and why) a concerned CCA might benefit the applications in their network.

7.2.3 Publishing data sets

Congestion control research would also benefit from large content providers and network operators publishing their traffic data sets or insights into their traffic matrices. On this topic, Gupta et. al [231] call for using invaluable campus network resources to overcome the lack of open datasets for networking researchers. There are also several ongoing efforts on generating high-fidelity synthetic data sets that could be invaluable for evaluating CCAs [232, 233].

We stress that any call for standardizing CCA evaluations should also consider proposals for curating diverse public (network traffic) data sets. To address the legitimate security and privacy concerns associated with releasing network traffic patterns, operators may consider sharing aggregated or anonymized packet traces (without payload), e.g., similar to the datasets made available by CAIDA [234] or those from the SWIM project [235].

There are multiple compelling incentives for content providers and network operators to contribute traffic matrices derived from their operational networks. First, the availability of realistic traffic traces enables the research community to design, evaluate, and compare congestion control mechanisms under realistic conditions. This leads to a better understanding of performance trade-offs and fosters the development of algorithms that improve network efficiency, benefiting the broader Internet ecosystem. Better congestion control ultimately reduces packet loss, increases throughput, and improves user experience, even for the network operators' own services. Second, operators who contribute such datasets have the opportunity to shape the development of reproducible evaluation benchmarks that are widely adopted in both academia and industry. This leadership role can directly influence broader industry standards, ensuring that they are informed by and aligned with real-world traffic conditions. Finally, by providing representative traffic matrices, operators can actively guide the research agenda toward protocols and deployment scenarios that are relevant to their own infrastructure, e.g., by emphasizing the performance needs of delay-sensitive applications.

7.2.4 CCA benchmarking in wireless, cellular mobile scenarios

Extending the benchmarking recipe evaluations presented in this thesis to cover network conditions uniquely present in wireless and cellular network scenarios. For instance, evaluations that consider outages and sudden fluctuations in mmWave communications [236], interactions between offloading and pacing, power imbalances and the high variability in media access [237, 238], present an opportunity to gain valuable insights into how CCAs should be assessed in such environments, including the identification of appropriate performance metrics. Given the increasing prevalence of wireless and cellular access technologies as primary means of Internet connectivity, it is important to investigate the implications of these inherently variable and less predictable network conditions. Such environments differ significantly from the more stable, wired networks for which many existing CCAs were originally designed, and therefore warrant dedicated analysis to understand their impact on congestion control behavior and fairness.

Throughout this dissertation, we conducted studies to highlight the challenges stemming from the lack of consensus and reproducibility in CCA evaluations. We believe that our approach of unifying goals concerning CCA evaluations to identify the minimal, fundamental set of criteria that any CCA evaluation must include helps in the drive towards standardizing CCA evaluations for the networking community.

Bibliography

[1] Danesh Zeynali, Emilia N. Weyulu, Seifeddine Fathalli, Balakrishnan Chandrasekaran, and Anja Feldmann. “Promises and Potential of BBRv3”. In: *Passive and Active Measurement (PAM).* 2024 (cit. on pp. ix, 2, 7, 16, 17, 29, 40, 42, 51, 52, 54, 65, 71, 75, 76, 97).

[2] Danesh Zeynali, Emilia N. Weyulu, Seifeddine Fathalli, Balakrishnan Chandrasekaran, and Anja Feldmann. “BBRv3 in the public Internet: a boon or a bane?” In: *Applied Networking Research Workshop (ANRW).* 2024 (cit. on pp. ix, 7, 16).

[3] Anja Feldmann, Balakrishnan Chandrasekaran, Seifeddine Fathalli, and Emilia N. Weyulu. “P4-enabled Network-assisted Congestion Feedback: A Case for NACKs”. In: *Stanford Workshop on Buffer Sizing (BS)* (2019) (cit. on pp. ix, 2, 7, 84).

[4] Emilia N. Weyulu, Danesh Zeynali, Seifeddine Fathalli, Adrian Zapletal, Balakrishnan Chandrasekaran, Anja Feldmann, and Fernando Kuipers. *A Recipe for Benchmarking Congestion Control Algorithms: Towards unifying and standardizing CCA evaluations.* Under Submission. 2025 (cit. on pp. ix, 7).

[5] Emilia N. Weyulu, Seifeddine Fathalli, Danesh Zeynali, Adrian Zapletal, Balakrishnan Chandrasekaran, Anja Feldmann, and Fernando Kuipers. *A call for an open, reproducible, standardized approach for evaluating congestion control algorithms.* Under Submission. 2025 (cit. on pp. x, 41).

[6] Seifeddine Fathalli, Emilia N. Weyulu, Danesh Zeynali, Balakrishnan Chandrasekaran, and Anja Feldmann. *Network-assisted Congestion Feedback.* Under Submission. 2025 (cit. on pp. x, 7).

[7] Emilia Ndilokelwa Weyulu and Dirk Trossen. “Exploring The Benefits of In-Band Service Routing”. In: *IFIP Networking.* 2024 (cit. on p. x).

[8] Cisco. *Cisco Annual Internet Report (2018–2023) White Paper.* Tech. rep. 2024 (cit. on pp. 1, 9).

[9] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. “A year in lockdown: how the waves of COVID-19 impact internet traffic”. In: *Communications of the ACM* (2021) (cit. on p. 1).

[10] Larry L. Peterson, Lawrence Brakmo, and Bruce S Davie. *TCP Congestion Control: A Systems Approach.* Systems Approach LLC, 2022 (cit. on pp. 1, 9).

[11] Sandvine. *The Global Internet Phenomena Report March 2024.* Tech. rep. Sandvine Corporation, 2024. URL: https://www.sandvine.com/global-internet-phenomena-report-2024?utm_referrer=https%3A%2F%2Fwww.sandvine.com%2Fphenomena (cit. on pp. 1, 9).

[12] Kevin R. Fall and William Richard Stevens. *TCP/IP Illustrated: The Protocols, 2e.* Addison-Wesley, 2012 (cit. on pp. 1, 10, 11, 14–16).

[13] Akshay Narayan, Frank Cangialosi, Deepti Raghavan, Prateesh Goyal, Srinivas Narayana, Radhika Mittal, Mohammad Alizadeh, and Hari Balakrishnan. *Restructuring endpoint congestion control [Paper presentation].* https://people.cs.rutgers.edu/~sn624/talks/ccp-iitm.pdf. [Last accessed: August 21, 2024]. 2018 (cit. on p. 2).

[14] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. "Bolt:{Sub-RTT} Congestion Control for {Ultra-Low} Latency". In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 2023 (cit. on pp. 2, 19, 81, 83–85).

[15] Venkat Arun and Hari Balakrishnan. "Copa: Practical delay-based congestion control for the internet". In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 2018 (cit. on pp. 2, 5, 17, 31, 32, 37, 42, 45, 53, 65).

[16] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. "Achieving Consistent Low Latency for Wireless Real-Time Communications with the Shortest Control Loop". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2022 (cit. on pp. 2, 19, 81, 85).

[17] Prateesh Goyal, Anup Agarwal, Ravi Netravali, Mohammad Alizadeh, and Hari Balakrishnan. "ABC: A Simple Explicit Congestion Controller for Wireless Networks". In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 2020 (cit. on pp. 2, 81).

[18] Chen-Yu Yen, Soheil Abbasloo, and H. Jonathan Chao. "Computers Can Learn from the Heuristic Designs and Master Internet Congestion Control". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2023 (cit. on pp. 2, 20, 31, 34, 45, 48).

[19] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. "The Great Internet TCP Congestion Control Census". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* (2019) (cit. on pp. 2, 5, 16, 27, 39, 45, 51, 65, 66, 75, 84).

[20] Ayush Mishra, Lakshay Rastogi, Raj Joshi, and Ben Leong. "Keeping an Eye on Congestion Control in the Wild with Nebby". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2024 (cit. on pp. 2, 5, 27, 39, 65, 66, 84).

[21] Ranysha Ware, Adithya Abraham Philip, Nicholas Hungria, Yash Kothari, Justine Sherry, and Srinivasan Seshan. "CCAnalyzer: An Efficient and Nearly-Passive Congestion Control Classifier". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2024 (cit. on pp. 2, 5, 16, 27, 39, 51, 65, 66, 84).

[22] Ayush Mishra, Sherman Lim, and Ben Leong. "Understanding Speciation in QUIC Congestion Control". In: *ACM Internet Measurement Conference (IMC).* 2022 (cit. on pp. 2, 18, 27, 41).

[23] Martin Duke and Gorry Fairhurst. *Specifying New Congestion Control Algorithms.* Internet-Draft draft-ietf-ccwg-rfc5033bis-08. Internet Engineering Task Force, Aug. 2024 (cit. on pp. 2, 20, 40, 41).

[24] Venkat Arun, Mina Tahmasbi Arashloo, Ahmed Saeed, Mohammad Alizadeh, and Hari Balakrishnan. "Toward formally verifying congestion control behavior". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2021 (cit. on pp. 2, 27, 32, 49).

[25] Goeff Huston. *BBR TCP.* https://labs.ripe.net/author/gih/bbr-tcp/. [Last accessed: Mai 10, 2024]. 2017 (cit. on pp. 2, 29, 42).

[26] Ranysha Ware, Matthew Mukerjee, Srini Seshan, and Justine Sherry. "Modeling BBR's Interactions with Loss-Based Congestion Control". In: *ACM Internet Measurement Conference (IMC).* 2019 (cit. on pp. 2, 16, 33, 43, 51, 54, 65, 67, 71, 75, 76, 89, 91, 97).

[27] 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: *SIGCOMM Workshop on Hot Topics in Networking (HotNets).* 2019 (cit. on pp. 2, 22, 25, 30).

[28] Adrian Zapletal and Fernando Kuipers. "Slowdown as a Metric for Congestion Control Fairness". In: *SIGCOMM Workshop on Hot Topics in Networking (HotNets).* 2023 (cit. on pp. 2, 22, 25, 61, 77).

[29] Sally Floyd. *Metrics for the Evaluation of Congestion Control Mechanisms.* RFC 5166. Mar. 2008 (cit. on pp. 2, 3, 20, 22, 25, 27, 30, 32, 40, 41, 46).

[30] David Hayes, David Ros, Lachlan L.H. Andrew, and Sally Floyd. *Common TCP Evaluation Suite.* Internet-Draft draft-irtf-iccrg-tcpeval-01. Internet Engineering Task Force, July 2014 (cit. on pp. 2, 21, 26, 27, 32, 40, 45, 46, 57, 92).

[31] Sally Floyd and Mark Allman. *Specifying New Congestion Control Algorithms.* RFC 5033. Aug. 2007 (cit. on p. 3).

[32] Nitin Garg. *Evaluating COPA congestion control for improved video performance.* https://engineering.fb.com/2019/11/17/video-engineering/copa/. [Last accessed: September 03, 2024]. 2019 (cit. on pp. 5, 17, 31).

[33] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach, 4e.* Morgan Kaufmann Publishers, 2007 (cit. on pp. 9, 11, 15, 17).

[34] Leonard Kleinrock. “Power and deterministic rules of thumb for probabilistic problems in computer communications”. In: *IEEE International Conference on Communications.* 1979 (cit. on pp. 10, 16, 65).

[35] Cardwell, Neal and Cheng, Yuchung and Gunn, C. Stephen and Yeganeh, Soheil Hassas and Jacobson, Van. “BBR: Congestion-Based Congestion Control”. In: *ACM Queue* (2016) (cit. on pp. 10, 16, 29, 34, 37, 42, 48, 52, 81, 84, 87).

[36] Luca Schumann, Trinh Viet Doan, Tanya Shreedhar, Ricky Mok, and Vaibhav Bajpai. “Impact of evolving protocols and COVID-19 on internet traffic shares”. In: *arXiv preprint arXiv:2201.00142* (2022) (cit. on pp. 11, 17).

[37] Mark Allman, Vern Paxson, and Richard W. Stevens. *TCP Congestion Control.* RFC 2581. Apr. 1999 (cit. on pp. 11, 12).

[38] James F Kurose and Keith Ross. *Computer Networking: A top-down approach featuring the Internet, 7/E.* Pearson Education, 2016 (cit. on p. 12).

[39] V. Jacobson. “Congestion Avoidance and Control”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 1988 (cit. on pp. 12, 13).

[40] Ethan Blanton, Dr. Vern Paxson, and Mark Allman. *TCP Congestion Control.* RFC 5681. Sept. 2009 (cit. on pp. 12, 14).

[41] Jerry Chu, Nandita Dukkipati, Yuchung Cheng, and Matt Mathis. *Increasing TCP’s Initial Window.* RFC 6928. Apr. 2013 (cit. on p. 12).

[42] Jan Rüth, Ike Kunze, and Oliver Hohlfeld. “TCP’s Initial Window—Deployment in the Wild and Its Impact on Performance”. In: (2019) (cit. on p. 12).

[43] Saverio Mascolo, Claudio Casetti, Mario Gerla, M. Y. Sanadidi, and Ren Wang. “TCP westwood: Bandwidth estimation for enhanced transport over wireless links”. In: *International Conference on Mobile Computing and Networking.* 2001 (cit. on p. 13).

[44] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. “Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks”. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 2013 (cit. on pp. 13, 17).

[45] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. “Adaptive Congestion Control for Unpredictable Cellular Networks”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2015 (cit. on pp. 13, 17).

[46] Andrei Gurtov, Tom Henderson, and Sally Floyd. *The NewReno Modification to TCP’s Fast Recovery Algorithm.* RFC 3782. Apr. 2004 (cit. on p. 15).

[47] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: a new TCP-friendly high-speed TCP variant". In: *ACM SIGOPS Operating Systems Review* (2008) (cit. on pp. 15, 37, 41, 42, 73, 84, 87).

[48] Jim Gettys and Kathleen Nichols. "Bufferbloat: dark buffers in the internet". In: *Communications of the ACM* (2012) (cit. on pp. 15, 33).

[49] Brakmo, L.S. and Peterson, L.L. "TCP Vegas: end to end congestion avoidance on a global Internet". In: *IEEE Journal on Selected Areas in Communications* (1995) (cit. on pp. 15, 16, 37).

[50] Richard J La, Jean Walrand, and Venkatachalam Anantharam. *Issues in TCP vegas*. Citeseer, 1999 (cit. on p. 16).

[51] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. "A compound TCP approach for high-speed and long distance networks". In: *IEEE Int. Conference on Computer Communications (INFOCOM)*. 2006 (cit. on pp. 16, 37).

[52] Stanislav Shalunov, Greg Hazel, Jana Iyengar, and Mirja Kühlewind. *Low Extra Delay Background Transport (LEDBAT)*. RFC 6817. Dec. 2012 (cit. on pp. 16, 47).

[53] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. "Developing a predictive model of quality of experience for internet video". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2013) (cit. on p. 16).

[54] Ioannis Arapakis, Xiao Bai, and B. Barla Cambazoglu. "Impact of response latency on user behavior in web search". In: *International ACM Conference on Research and Development in Information Retrieval*. 2014 (cit. on p. 16).

[55] Neal Cardwell and Cheng Yuchung. *TCP BBR congestion control comes to GCP - your Internet just got faster*. https://cloud.google.com/blog/products/networking/tcp-bbr-congestion-control-comes-to-gcp-your-internet-just-got-faster/. [Last accessed: February 04, 2024]. 2017 (cit. on p. 16).

[56] Mario Hock, Roland Bless, and Martina Zitterbart. "Experimental evaluation of BBR congestion control". In: *IEEE Int. Conference on Network Protocols (ICNP)*. 2017 (cit. on pp. 16, 37, 52, 54, 65, 67, 69, 73, 75).

[57] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, Victor Vasiliev, Matt Mathis Bin Wu, Priyaranjan Jha, Yousuk Seung, and Van Jacobson. *BBR v2: A Model-based Congestion Control IETF 105 Update*. Tech. rep. Internet Engineering Task Force (IETF), 2019. URL: https://datatracker.ietf.org/meeting/105/materials/slides-105-iccrg-bbr-v2-a-model-based-congestion-control-00 (cit. on pp. 17, 29).

[58] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Ian Swett, Jana Iyangar, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, and Van Jacobson. *BBR Congestion Control Work at Google IETF 101 Update*. Tech. rep. Internet Engineering Task Force (IETF), 2018. URL: https://datatracker.ietf.org/meeting/101/materials/slides-101-iccrg-an-update-on-bbr-work-at-google-00 (cit. on pp. 17, 29, 45).

[59] Aarti Nandagiri, Mohit P. Tahiliani, Vishal Misra, and K. K. Ramakrishnan. "BBRvl vs BBRv2: Examining Performance Differences through Experimental Evaluation". In: *IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN* (2020) (cit. on pp. 17, 67).

[60] Alexey Ivanov. "Evaluating BBRv2 on the Dropbox Edge Network". In: *Netdev, The Technical Conference on Linux Networking*. 2020 (cit. on pp. 17, 67).

[61] Yeong-Jun Song, Geon-Hwan Kim, Imtiaz Mahmud, Won-Kyeong Seo, and You-Ze Cho. "Understanding of BBRv2: Evaluation and Comparison With BBRv1 Congestion Control Algorithm". In: *IEEE Access* (2021) (cit. on pp. 17, 67).

[62] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Steven Yang, David Morley, Soheil Hassas Yeganeh, Ian Swett, Bin Wu, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, and Van Jacobson. *BBRv3: Algorithm Bug Fixes and Public Internet Deployment*. Tech. rep. Internet Engineering Task Force (IETF), 2023. URL: https://datatracker.ietf.org/meeting/117/materials/slides-117-ccwg-bbrv3-algorithm-bug-fixes-and-public-internet-deployment-00 (cit. on pp. 17, 29, 45, 47, 66, 67, 70, 75, 76).

[63] Neal Cardwell, Ian Swett, and Joseph Beshay. *BBR Congestion Control*. Internet-Draft draft-cardwell-ccwg-bbr-00. Internet Engineering Task Force, July 2024 (cit. on p. 17).

[64] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. "The QUIC transport protocol: Design and internet-scale deployment". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2017 (cit. on p. 17).

[65] Robin Marx. *Head-of-Line Blocking in QUIC and HTTP/3: The Details*. https://calendar.perfplanet.com/2020/head-of-line-blocking-in-quic-and-http-3-the-details/. [Last accessed: May 22, 2025]. 2020 (cit. on p. 17).

[66] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018 (cit. on p. 17).

[67] C Kenjiro, M Koushirou, and K Akira. "Traffic data repository at the WIDE project". In: *USENIX Annual Technical Conference (ATC)*. 2000 (cit. on pp. 17, 22, 43, 46, 60, 77).

[68] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021 (cit. on p. 17).

[69] Mike Bishop. *HTTP/3*. RFC 9114. June 2022 (cit. on pp. 18, 39, 41).

[70] Web Almanac. *HTTP Archive's annual state of the web report*. Tech. rep. HTTP Archive, 2024. URL: https://almanac.httparchive.org/en/2024/http (cit. on pp. 18, 39, 41).

[71] Ayush Mishra and Ben Leong. “Containing the Cambrian Explosion in QUIC Congestion Control”. In: *ACM Internet Measurement Conference (IMC)*. 2023 (cit. on pp. 18, 27, 41).

[72] K. K. Ramakrishnan and Raj Jain. “A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 1988 (cit. on pp. 18, 84).

[73] Sally Floyd and Van Jacobson. “Random Early Detection Gateways for Congestion Avoidance”. In: *IEEE/ACM Transactions on Networking (ToN)* (1993) (cit. on pp. 18, 84, 85).

[74] K. Ramakrishnan, S. Floyd, and D. Black. “The Addition of Explicit Congestion Notification (ECN) to IP”. In: *RFC 3168, Internet Request for Comments* 3168 (Sept. 2001) (cit. on pp. 18, 84, 85).

[75] Rong Pan, Balaji Prabhakar, and Ashvin Laxmikantha. “QCN: Quantized congestion notification”. In: *IEEE 802* (2007) (cit. on pp. 18, 85).

[76] Dina Katabi, Mark Handley, and Charlie Rohrs. “Congestion Control for High Bandwidth-delay Product Networks”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2002 (cit. on pp. 18, 83–85).

[77] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown. “Processor Sharing Flows in the Internet”. In: *International Workshop on Quality of Service (IWQoS)*. 2005 (cit. on pp. 18, 84, 85).

[78] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. “Data Center TCP (DCTCP)”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2010 (cit. on pp. 18, 81, 85).

[79] Radhika Mittal, Terry Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. “TIMELY: RTT-based Congestion Control for the Datacenter”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2015 (cit. on pp. 18, 85).

[80] David Zats, Anand Padmanabha Iyer, Ganesh Ananthanarayanan, Rachit Agarwal, Randy Katz, Ion Stoica, and Amin Vahdat. “FastLane: Making Short Flows Shorter with Agile Drop Notification”. In: *ACM Symposium on Cloud Computing*. 2015 (cit. on pp. 18, 85).

[81] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. “HPCC: High Precision Congestion Control”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2019 (cit. on pp. 18, 19, 83, 85, 87).

[82] Serhat Arslan and Nick McKeown. “Switches Know the Exact Amount of Congestion”. In: *Workshop on Buffer Sizing.* 2019 (cit. on p. 18).

[83] Parvin Taheri, Danushka Menikkumbura, Erico Vanini, Sonia Fahmy, Patrick Eugster, and Tom Edsall. “RoCC: Robust Congestion Control for RDMA”. In: *ACM CoNEXT.* 2020 (cit. on pp. 19, 85, 91).

[84] Prateesh Goyal, Preey Shah, Kevin Zhao, Georgios Nikolaidis, Mohammad Alizadeh, and Thomas E. Anderson. “Backpressure Flow Control”. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 2022 (cit. on pp. 19, 85).

[85] Bob Briscoe, Koen De Schepper, Marcelo Bagnulo, and Greg White. *Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture.* RFC 9330. Jan. 2023 (cit. on pp. 19, 85).

[86] Koen De Schepper, Olivier Tilmans, Bob Briscoe, and Vidhi Goel. *Prague Congestion Control.* Internet-Draft. July 2024 (cit. on pp. 19, 85).

[87] Bob Briscoe, Mirja Kühlewind, and Richard Scheffenegger. *More Accurate Explicit Congestion Notification (ECN) Feedback in TCP.* Internet-Draft. July 2024 (cit. on pp. 19, 85).

[88] Koen De Schepper, Bob Briscoe, and Greg White. *Dual-Queue Coupled Active Queue Management (AQM) for Low Latency, Low Loss, and Scalable Throughput (L4S).* RFC 9332. Jan. 2023 (cit. on pp. 19, 85).

[89] Keith Winstein and Hari Balakrishnan. “Tcp ex machina: Computer-generated congestion control”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2013 (cit. on pp. 19, 37, 42).

[90] Francis Y Yan, Jestin Ma, Greg D Hill, Deepti Raghavan, Riad S Wahby, Philip Levis, and Keith Winstein. “Pantheon: the training ground for Internet congestion-control research”. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 2018 (cit. on pp. 19, 20, 26, 27, 35, 41, 43).

[91] Mo Dong, Qingxi Li, Doron Zarchy, P Brighten Godfrey, and Michael Schapira. “{PCC}: Re-architecting Congestion Control for Consistent High Performance”. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI).* 2015 (cit. on pp. 19, 42).

[92] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. “PCC Vivace: Online-Learning Congestion Control”. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2018 (cit. on pp. 19, 31, 33, 37, 45, 65, 81, 91, 98).

[93] Nathan Jay, Noga H. Rotman, Brighten Godfrey, and Michael Schapira and Aviv Tamar. “A Deep Reinforcement Learning Perspective on Internet Congestion Control”. In: *International Conference on Machine Learning*. 2019 (cit. on p. 19).

[94] Soheil Abbasloo, Chen-Yu Yen, and H Jonathan Chao. “Classic meets modern: A pragmatic learning-based congestion control for the internet”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2020 (cit. on pp. 20, 37, 42, 98).

[95] NS-3. *NS-3 Network Simulator*. https://www.nsnam.org/. [Last accessed: Mai 10, 2024]. 2023 (cit. on pp. 20, 27, 41).

[96] Bob Lantz, Brandon Heller, and Nick McKeown. “A network in a laptop: rapid prototyping for software-defined networks”. In: *SIGCOMM Workshop on Hot Topics in Networking (HotNets)*. 2010 (cit. on p. 20).

[97] Sebastian Zander and Grenville Armitage. *TEACUP v1.0 – A System for Automated TCP Testbed Experiments*. Tech. rep. Centre for Advanced Internet Architectures, Swinburne University of Technology, 2015 (cit. on pp. 20, 26, 27, 41).

[98] Toke Høiland-Jørgensen, Carlo Augusto Grazia, Per Hurtig, and Anna Brunstrom. “Flent: The FLExible Network Tester”. In: *EAI International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*. 2017 (cit. on pp. 20, 26, 27, 41).

[99] Google. *Transperf*. https://github.com/google/transperf. [Last accessed: Nov 16, 2024]. 2025 (cit. on p. 20).

[100] Linux. *namespaces(7) — Linux manual page*. https://man7.org/linux/man-pages/man7/namespaces.7.html. [Last accessed: Mar 11, 2025]. 2024 (cit. on p. 20).

[101] Charles E Leiserson. “Fat-trees: Universal networks for hardware-efficient supercomputing”. In: *IEEE transactions on Computers* (1985) (cit. on p. 21).

[102] Michael Schapira and Keith Winstein. “Congestion-control throwdown”. In: *SIGCOMM Workshop on Hot Topics in Networking (HotNets)*. 2017 (cit. on pp. 21, 43).

[103] Anirudh Sivaraman, Keith Winstein, Pratiksha Thaker, and Hari Balakrishnan. “An experimental study of the learnability of congestion control”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)* (2014) (cit. on pp. 21, 43).

[104] Won-Seok Park, Minsu Shin, Hyang-Won Lee, and Song Chong. "A joint design of congestion control and burst contention resolution for optical burst switching networks". In: *IEEE Journal of Lightwave Technology* (2009) (cit. on p. 21).

[105] Kun Tan, Feng Jiang, Qian Zhang, and Xuemin Shen. "Congestion control in multihop wireless networks". In: *IEEE Transactions on Vehicular Technology* (2007) (cit. on p. 21).

[106] Xiaolong Li and Homayoun Yousefi'zadeh. "Robust EKF-based wireless congestion control". In: *IEEE Transactions on Communications (TCOM)* (2013) (cit. on p. 21).

[107] Saurabh Jain, Yueping Zhang, and Dmitri Loguinov. "Towards experimental evaluation of explicit congestion control". In: *International Workshop on Quality of Service (IWQoS)*. 2008 (cit. on pp. 21, 37).

[108] Rayadurgam Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012 (cit. on p. 21).

[109] Larry L. Peterson, Lawrence Brakmo, and Bruce S. Davie. *TCP Congestion Control: A Systems Approach*. Systems Approach, 2022 (cit. on pp. 21, 43).

[110] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. "Distributed Network Monitoring and Debugging with Switchpointer". In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2018 (cit. on pp. 21, 92).

[111] Christian L. Vielhaus, Caspar von Lengerke, Vincent Latzko, Justus Rischke, Martin Reisslein, and Frank H. P. Fitzek. "Evaluating Transport Layer Congestion Control Algorithms: A Comprehensive Survey". In: *IEEE Comm. Surveys and Tutorials* (2025) (cit. on p. 22).

[112] Ike Kunze, Jan Rüth, and Oliver Hohlfeld. "Congestion Control in the Wild - Investigating Content Provider Fairness". In: *IEEE Trans. on Network and Service Management (TNSM)*. 2019 (cit. on pp. 22, 30, 37).

[113] Youngmi Joo, Vinay Ribeiro, Anja Feldmann, Anna C. Gilbert, and Walter Willinger. "TCP/IP Traffic Dynamics and Network Performance: A Lesson in Workload Modeling, Flow Control, and Trace-driven Simulations". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2001) (cit. on pp. 22, 43, 85).

[114] Geoff Huston. *APRICOT 2018: Geoff Huston on BBR*. https://www.youtube.com/watch?v=l5jGxrX3w9M. [Last accessed: Mai 10, 2024]. 2018 (cit. on pp. 22, 29, 43).

[115] Safiqul Islam, Kristian Hiorth, Carsten Griwodz, and Michael Welzl. "Is it really necessary to go beyond a fairness metric for next-generation congestion control?" In: *Applied Networking Research Workshop (ANRW)*. 2022 (cit. on p. 25).

[116] Martin Duke and Gorry Fairhurst. *Specifying New Congestion Control Algorithms*. Active Internet-Draft draft-ietf-ccwg-rfc5033bis-02. Internet Engineering Task Force, Oct. 2023 (cit. on pp. 26, 30).

[117] Soheil Abbasloo. "Internet Congestion Control Benchmarking". In: *arXiv e-prints* (July 2023) (cit. on pp. 26, 30).

[118] George A. Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (Nov. 1995) (cit. on p. 26).

[119] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009 (cit. on p. 26).

[120] Transaction Processing Performance Council. *TPC Benchmarks Overview*. https://www.tpc.org/default5.asp. [Last accessed: Nov 16, 2024]. 2024 (cit. on p. 26).

[121] Standard Performance Evaluation Corporation. *Standard Performance Evaluation Corporation*. https://www.spec.org/. [Last accessed: Nov 16, 2024]. 2024 (cit. on p. 26).

[122] OMNeT++. *OMNeT++*. https://omnetpp.org/. [Last accessed: Mai 10, 2024]. 2023 (cit. on pp. 27, 41).

[123] X. Zeng, R. Bagrodia, and M. Gerla. "GloMoSim: a library for parallel simulation of large-scale wireless networks". In: *Workshop on Parallel and Distributed Simulation (PADS)*. 1998 (cit. on pp. 27, 41).

[124] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. "Mahimahi: accurate {Record-and-Replay} for {HTTP}". In: *USENIX Annual Technical Conference (ATC)*. 2015 (cit. on pp. 27, 41, 67).

[125] Margarida Ferreira, Akshay Narayan, Inês Lynce, Ruben Martins, and Justine Sherry. "Counterfeiting Congestion Control Algorithms". In: *ACM Internet Measurement Conference (IMC)*. 2021 (cit. on p. 27).

[126] Devdeep Ray and Srinivasan Seshan. "CC-Fuzz: Genetic Algorithm-Based Fuzzing for Stress Testing Congestion Control Algorithms". In: *SIGCOMM Workshop on Hot Topics in Networking (HotNets)*. 2022 (cit. on pp. 27, 41, 99).

[127] Margarida Ferreira, Ranysha Ware, Yash Kothari, Inês Lynce, Ruben Martins, Akshay Narayan, and Justine Sherry. "Reverse-Engineering Congestion Control Algorithm Behavior". In: *ACM Internet Measurement Conference (IMC)*. 2024 (cit. on pp. 27, 41, 99).

[128] Belma Turkovic, Fernando A. Kuipers, and Steve Uhlig. "Interactions between Congestion Control Algorithms". In: *Network Traffic Measurement and Analysis Conference (TMA)*. 2019 (cit. on pp. 27, 37, 41, 43, 65, 67, 73, 75).

[129] Matthias Flittner, Mohamed Naoufal Mahfoudi, Damien Saucez, Matthias Wählisch, Luigi Iannone, Vaibhav Bajpai, and Alex Afanasyev. "A Survey on Artifacts from CoNEXT, ICN, IMC, and SIGCOMM Conferences in 2017". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2018) (cit. on p. 29).

[130] Monya Baker. *1,500 scientists lift the lid on reproducibility.* https://www.nature.com/articles/533452a. [Last accessed: Nov 16, 2024]. 2016 (cit. on p. 29).

[131] Neal Cardwell, Yuchung Cheng, Kevin Yang, David Morley, Soheil Hassas Yeganeh, Priyaranjan Jha, Yousuk Seung, Van Jacobson, Ian Swett, Bin Wu, and Victor Vasiliev. *BBR Congestion Control: Fundamentals and Updates.* https://research.cec.sc.edu/files/cyberinfra/files/BBR%20-%20Fundamentals%20and%20Updates%202023-08-29.pdf. [Last accessed: Mai 10, 2024]. 2023 (cit. on pp. 29, 51, 66).

[132] Saahil Claypool. *Sharing but not Caring - Performance of TCP BBR and TCP CUBIC at the Network Bottleneck.* 2019 (cit. on p. 29).

[133] Mario Hock, Felix Neumeister, Martina Zitterbart, and Roland Bless. "TCP LoLa: Congestion control for low latencies and high throughput". In: *IEEE Conference on Local Computer Networks (LCN).* 2017 (cit. on p. 37).

[134] Luigi A Grieco and Saverio Mascolo. "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2004) (cit. on p. 37).

[135] Douglas J Leith, Robert N Shorten, and Gavin McCullagh. "Experimental evaluation of cubic-TCP". In: *Protocols for FAST Long-Distance Networks* (2008) (cit. on p. 37).

[136] Yee-Ting Li, Douglas Leith, and Robert N Shorten. "Experimental evaluation of TCP protocols for high-speed networks". In: *IEEE/ACM Transactions on Networking (ToN)* (2007) (cit. on pp. 37, 65).

[137] Sangtae Ha, Yusung Kim, Long Le, Injong Rhee, and Lisong Xu. "A step toward realistic performance evaluation of high-speed TCP variants". In: *Protocols for FAST Long-Distance Networks.* 2006 (cit. on pp. 37, 65).

[138] Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle. "Towards a Deeper Understanding of TCP BBR Congestion Control". In: *IFIP Networking.* 2018 (cit. on pp. 37, 65, 67, 69).

[139] Yi Cao, Arpit Jain, Kriti Sharma, Aruna Balasubramanian, and Anshul Gandhi. "When to Use and When Not to Use BBR: An Empirical Analysis and Evaluation Study". In: *ACM Internet Measurement Conference (IMC).* 2019 (cit. on pp. 33, 37, 51, 65, 67, 69, 71, 75, 89).

[140] Ferenc Fejes, Gergő Gombos, Sándor Laki, and Szilveszter Nádas. "Who Will Save the Internet from the Congestion Control Revolution?" In: *Workshop on Buffer Sizing.* 2019 (cit. on pp. 37, 67).

[141] Adithya Abraham Philip, Ranysha Ware, Rukshani Athapathu, Justine Sherry, and Vyas Sekar. "Revisiting TCP Congestion Control Throughput Models & Fairness Properties at Scale". In: *ACM Internet Measurement Conference (IMC)*. 2021 (cit. on pp. 37, 76, 81).

[142] Daniela M Casas-Velasco, Fabrizio Granelli, and Nelson LS da Fonseca. "Impact of Background Traffic on the BBR and CUBIC Variants of the TCP Protocol". In: *IEEE Networking Letters* (2022) (cit. on p. 37).

[143] Simon Scherrer. *Model-Based Insights on the Future Internet*. Ph.D Thesis, ETH Zurich. 2024 (cit. on p. 31).

[144] Compira Labs. *Compira Labs Boosts Quality of Experience for Leading US Service Provider*. https://www.compiralabs.com/_files/ugd/5d7963_cb52534516fd4c1d84458720279cd963.pdf. [Last accessed: Nov 16, 2024]. 2022 (cit. on p. 31).

[145] Stephen Hemminger and Fabio Ludovici and Hagen Paul Pfeifer. *tc-netem(8) — Linux manual page*. https://man7.org/linux/man-pages/man8/tc-netem.8.html. [Last accessed: Mai 10, 2024]. 2023 (cit. on pp. 31, 45).

[146] Alexey N. Kuznetsov and Bert hubert. *tc-tbf(8) - Linux man page*. https://linux.die.net/man/8/tc-tbf. [Last accessed: Mai 10, 2024]. 2023 (cit. on pp. 31, 45).

[147] Feixue Han, Qing Li, Peng Zhang, Gareth Tyson, Yong Jiang, Mingwei Xu, Yulong Lan, and ZhiCheng Li. "ETC: An Elastic Transmission Control Using End-to-End Available Bandwidth Perception". In: *USENIX Annual Technical Conference (ATC)*. 2024 (cit. on p. 32).

[148] Zeqi Lai, Zonglun Li, Qian Wu, Hewu Li, Weisen Liu, Yijie Liu, Xin Xie, Yuanjie Li, and Jun Liu. "Mind the Misleading Effects of LEO Mobility on End-to-End Congestion Control". In: *SIGCOMM Workshop on Hot Topics in Networking (HotNets)*. 2024 (cit. on p. 32).

[149] Randell Jesup and Zaheduzzaman Sarker. *Congestion Control Requirements for Interactive Real-Time Media*. RFC 8836. Jan. 2021 (cit. on p. 33).

[150] Kathleen Nichols and Van Jacobson. "Controlling Queue Delay". In: *Communications of the ACM* (2012) (cit. on pp. 33, 85).

[151] Jonathan Corbet. *TSO sizing and the FQ scheduler*. https://lwn.net/Articles/564978/. [Last accessed: Mai 10, 2024]. 2023 (cit. on p. 35).

[152] Yuchung Cheng and Neal Cardwell. "Making linux TCP fast". In: *Netdev, The Technical Conference on Linux Networking*. 2016 (cit. on p. 35).

[153] Michael Kerrisk. *overview of time and timers*. https://man7.org/linux/man-pages/man7/time.7.html. [Last accessed: Mai 10, 2024]. 2023 (cit. on p. 35).

[154] Benjamin Peters, Pinhan Zhao, Jae Won Chung, and Mark Claypool. "Tcp hystart performance over a satellite network". In: *Netdev, The Technical Conference on Linux Networking*. 2021 (cit. on p. 35).

[155] Jan Rüth, Christian Bormann, and Oliver Hohlfeld. “Large-Scale Scanning of TCP’s Initial Window”. In: *ACM Internet Measurement Conference (IMC).* 2017 (cit. on p. 35).

[156] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. “An argument for increasing TCP’s initial congestion window”. In: (2010) (cit. on p. 35).

[157] ACM Internet Measurement Conference. *Call For Papers, ACM IMC 2025.* https://conferences.sigcomm.org/imc/2025/cfp/. [Last accessed: Nov 16, 2024]. 2024 (cit. on p. 35).

[158] PAM. *Call for Papers.* https://udesa.edu.ar/call-papers. [Last accessed: Nov 16, 2024]. 2024 (cit. on p. 35).

[159] Rukshani Athapathu, Ranysha Ware, Srinivasan Seshan Aditya Abraham Philip, and Justine Sherry. “Prudentia: Measuring Congestion Control Harm on the Internet”. In: *Networking Networking Women Workshop (N2Women).* 2020 (cit. on p. 41).

[160] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. “Sizing Router Buffers”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2004 (cit. on pp. 42, 47, 51, 66).

[161] Neda Beheshti, Yashar Ganjali, Monia Ghobadi, Nick McKeown, and Geoff Salmon. “Experimental Study of Router Buffer Sizing”. In: *ACM Internet Measurement Conference (IMC).* 2008 (cit. on pp. 42, 47, 51, 66).

[162] Balakrishnan Chandrasekaran, Georgios Smaragdakis, Arthur Berger, Matthew Luckie, and Keung-Chi Ng. “A Server-to-Server View of the Internet”. In: *ACM CoNEXT.* 2015 (cit. on pp. 42, 43, 47, 51, 66, 69, 82).

[163] Vasileios Giotsas, Christoph Dietzel, Georgios Smaragdakis, Anja Feldmann, Arthur Berger, and Emile Aben. “Detecting Peering Infrastructure Outages in the Wild”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM).* 2017 (cit. on p. 43).

[164] S. Ebrahimi-Taghizadeh, A. Helmy, and S. Gupta. “TCP vs. TCP: a systematic study of adverse impact of short-lived TCP flows on long-lived TCP flows”. In: *IEEE Joint Conf. of the IEEE Computer and Communications Societies.* 2005 (cit. on p. 43).

[165] Simon Bauer, Benedikt Jaeger, Fabian Helfert, Philippe Barias, and Georg Carle. “On the evolution of internet flow characteristics”. In: *Applied Networking Research Workshop (ANRW).* 2021 (cit. on p. 43).

[166] Jonathan Corbet. *TCP small queues.* https://lwn.net/Articles/507065/. [Last accessed: Mai 10, 2024]. 2012 (cit. on p. 45).

[167] Neal Cardwell. *BBR evaluation with netem.* https://groups.google.com/g/bbr-dev/c/8LYkNt17V_8. [Last accessed: Mai 10, 2024]. 2017 (cit. on p. 45).

[168] Praveen Balasubramanian, Osman Ertugay, and Daniel Havey. *LEDBAT++: Congestion Control for Background Traffic.* Internet-Draft draft-irtf-iccrg-ledbat-plus-plus-01. Internet Engineering Task Force, Aug. 2020 (cit. on pp. 45, 55).

[169] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Ian Swett, Jana Iyangar, Victor Vasiliev, and Van Jacobson. *BBR Congestion Control: IETF 100 Update: BBR in shallow buffers.* Tech. rep. Internet Engineering Task Force (IETF), 2017. URL: https://www.ietf.org/proceedings/100/slides/slides-100-iccrg-a-quick-bbr-update-bbr-in-shallow-buffers-00.pdf (cit. on pp. 45, 70).

[170] Google. *TCP BBR v3 Release.* https://github.com/google/bbr/tree/v3. [Last accessed: Mai 10, 2024]. 2023 (cit. on p. 45).

[171] Li, Q. *LEDBAT-Plus-Plus.* https://github.com/google/bbr/tree/v3. [Last accessed: Jan 10, 2025]. 2022 (cit. on p. 46).

[172] iPerf2. *iPerf2.* https://iperf.fr/iperf-doc.php. [Last accessed: Mai 10, 2024]. 2023 (cit. on p. 46).

[173] International Computer Science Institute (ICSI). *Zeek: An Open Source Network Security Monitoring Tool.* http://www.zeek.org. [Last accessed: Mai 10, 2024]. 2022 (cit. on p. 46).

[174] Joel Sommers, Hyungsuk Kim, and Paul Barford. "Harpoon: A Flow-Level Traffic Generator for Router and Network Tests". In: *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science* (2004) (cit. on pp. 46, 61, 77, 87, 93).

[175] Oliver Hohlfeld, Enric Pujol, Florin Ciucu, Anja Feldmann, and Paul Barford. "A QoE Perspective on Sizing Network Buffers". In: *ACM Internet Measurement Conference (IMC).* 2014 (cit. on p. 47).

[176] Arun Vishwanath, Vijay Sivaraman, and Marina Thottan. "Perspectives on Router Buffer Sizing: Recent Results and Open Problems". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2009) (cit. on p. 47).

[177] Joel Sommers, Paul Barford, Albert Greenberg, and Walter Willinger. "An SLA Perspective on the Router Buffer Sizing Problem". In: *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science* (2008) (cit. on p. 47).

[178] Ravi S. Prasad, Constantine Dovrolis, and Marina Thottan. "Router Buffer Sizing Revisited: The Role of the Output/Input Capacity Ratio". In: *ACM CoNEXT.* 2007 (cit. on p. 47).

[179] Yashar Ganjali and Nick McKeown. "Update on Buffer Sizing in Internet Routers". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2006) (cit. on p. 47).

[180] Gaurav Raina, Don Towsley, and Damon Wischik. "Part II: Control Theory for Buffer Sizing". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2005) (cit. on p. 47).

[181] Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker. “Recursively Cautious Congestion Control”. In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2014 (cit. on p. 61).

[182] RIPE Atlas. *RTT Measurements to Fixed Destinations*. https://atlas.ripe.net/results/maps/rtt-fixed//. [Last accessed: Mai 10, 2024]. 2024 (cit. on p. 62).

[183] Dah-Ming Chiu and Raj Jain. “Analysis of the increase and decrease algorithms for congestion avoidance in computer networks”. In: *Computer Networks and ISDN Systems* (1989) (cit. on p. 65).

[184] Ranysha Ware. “Battle for Bandwidth: On The Deployability of New Congestion Control Algorithms”. PhD thesis. School of Computer Science, Carnegie Mellon University, 2024 (cit. on p. 65).

[185] Saahil Claypool, Jae Chung, and Mark Claypool. “Comparison of TCP Congestion Control Performance over a Satellite Network”. In: *Passive and Active Measurement (PAM)*. 2021 (cit. on p. 67).

[186] Per Hurtig, Habtegebreil Kassaye Haile, Karl-Johan Grinnemo, Anna Brunström, Eneko Atxutegi, Fidel Liberal, and Åke Arvidsson. “Impact of TCP BBR on CUBIC Traffic: A Mixed Workload Evaluation”. In: *International Teletraffic Congress*. 2018 (cit. on p. 67).

[187] Fred Baker and Gorry Fairhurst. *IETF Recommendations Regarding Active Queue Management*. RFC 7567. July 2015 (cit. on p. 67).

[188] Carlo Augusto Grazia, Natale Patriciello, Martin Klapez, and Maurizio Casoni. “A cross-comparison between TCP and AQM algorithms: Which is the best couple for congestion control?” In: *International Conference on Telecommunications*. 2017 (cit. on p. 67).

[189] Hao Zhang, Haiting Zhu, Yu Xia, Lu Zhang, Yuan Zhang, and Yingying Deng. “Performance Analysis of BBR Congestion Control Protocol Based on NS3”. In: *International Conference on Advanced Cloud and Big Data (CBD)*. 2019 (cit. on p. 67).

[190] Elie F. Kfoury, Jose Gomez, Jorge Crichigno, and Elias Bou-Harb. “An emulation-based evaluation of TCP BBRv2 Alpha for wired broadband”. In: *Computer Communications* (2020) (cit. on p. 67).

[191] Mohit P. Tahiliani and Vishal Misra. “A Principled Look at the Utility of Feedback in Congestion Control”. In: *Workshop on Buffer Sizing*. 2019 (cit. on p. 67).

[192] Furong Yang, Qinghua Wu, Zhenyu Li, Yanmei Liu, Giovanni Pau, and Gaogang Xie. “BBRv2+: Towards balancing aggressiveness and fairness with delay-based bandwidth probing”. In: *Computer Networks* (2022) (cit. on p. 67).

[193] Pengqiang Bi, Mengbai Xiao, Dongxiao Yu, and Guanghui Zhang. “oBBR: Optimize Retransmissions of BBR Flows on the Internet”. In: *USENIX Annual Technical Conference (ATC)*. 2023 (cit. on p. 67).

[194] Intel Corporation. *Intel® Tofino 6.4Tbps, 4 pipelines*. https://www.intel.com/content/www/us/en/products/sku/218643/intel-tofino-6-4-tbps-4-pipelines/specifications.html. [Last accessed: Mai 10, 2024]. 2017 (cit. on pp. 69, 86).

[195] Bruce Spang, Brady Walsh, Te-Yuan Huang, Tom Rusnock, Joe Lawrence, and Nick McKeown. "Buffer sizing and Video QoE Measurements at Netflix". In: *Workshop on Buffer Sizing*. 2019 (cit. on p. 69).

[196] P. Brown. "Resource sharing of TCP connections with different round trip times". In: *IEEE Int. Conference on Computer Communications (INFOCOM)*. 2000 (cit. on p. 73).

[197] Jeonghoon Mo and Jean Walrand. "Fair end-to-end window-based congestion control". In: *IEEE/ACM Transactions on Networking (ToN)* (2000) (cit. on p. 81).

[198] Liangcheng Yu, John Sonchack, and Vincent Liu. "Cebinae: scalable in-network fairness augmentation". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2022 (cit. on p. 81).

[199] Yanpei Chen, Rean Griffith, Junda Liu, Randy H. Katz, and Anthony D. Joseph. "Understanding TCP Incast Throughput Collapse in Datacenter Networks". In: *ACM Workshop on Research on Enterprise Networking (WREN)*. 2009 (cit. on p. 81).

[200] Venkat Arun, Mohammad Alizadeh, and Hari Balakrishnan. "Starvation in end-to-end congestion control". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2022 (cit. on p. 81).

[201] Renjie Zhou, Dezun Dong, Shan Huang, and Yang Bai. "FastTune: Timely and Precise Congestion Control in Data Center Network". In: *IEEE Intl. Conf. on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 2021 (cit. on pp. 81, 83, 84).

[202] Inho Cho, Keon Jang, and Dongsu Han. "Credit-Scheduled Delay-Bounded Congestion Control for Datacenters". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2017 (cit. on pp. 81, 84).

[203] Changhyun Lee, Chunjong Park, Keon Jang, Sue Moon, and Dongsu Han. "Accurate Latency-based Congestion Feedback for Datacenters". In: *USENIX Annual Technical Conference (ATC)*. 2015 (cit. on p. 81).

[204] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2014) (cit. on p. 81).

[205] Enric Pujol, Ingmar Poese, Johannes Zerwas, Georgios Smaragdakis, and Anja Feldmann. "Steering hyper-giants' traffic at scale". In: *ACM CoNEXT*. 2019 (cit. on pp. 82, 83).

[206] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, et al. "Annulus: A dual congestion control loop for datacenter and wan traffic aggregates". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2020 (cit. on p. 82).

[207] Wohlfart, Florian and Chatzis, Nikolaos and Dabanoglu, Caglar and Carle, Georg and Willinger, Walter. "Leveraging interconnections for performance: the serving infrastructure of a large CDN". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2018 (cit. on p. 83).

[208] Gupta, Arpit and Vanbever, Laurent and Shahbaz, Muhammad and Donovan, Sean P and Schlinker, Brandon and Feamster, Nick and Rexford, Jennifer and Shenker, Scott and Clark, Russ and Katz-Bassett, Ethan. "SDX: a software defined Internet exchange". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2014) (cit. on p. 83).

[209] Poese, Ingmar and Frank, Benjamin and Smaragdakis, Georgios and Uhlig, Steve and Feldmann, Anja and Maggs, Bruce. "Enabling content-aware traffic engineering". In: *ACM SIGCOMM Computer Communications Review (CCR)* (2012) (cit. on p. 83).

[210] Dhamdhere, Amogh and Clark, David D and Gamero-Garrido, Alexander and Luckie, Matthew and Mok, Ricky KP and Akiwate, Gautam and Gogia, Kabir and Bajpai, Vaibhav and Snoeren, Alex C and Claffy, Kc. "Inferring persistent interdomain congestion". In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2018 (cit. on p. 83).

[211] Alexandros Milolidakis, Romain Fontugne, and Xenofontas Dimitropoulos. "Detecting network disruptions at colocation facilities". In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE. 2019, pp. 2161–2169 (cit. on p. 83).

[212] Luckie, Matthew and Dhamdhere, Amogh and Clark, David and Huffaker, Bradley and Claffy, KC. "Challenges in inferring internet interdomain congestion". In: *ACM Internet Measurement Conference (IMC)*. 2014 (cit. on p. 83).

[213] Ling, Fei-Yang and Tang, Shou-Lian and Wu, Miao and Li, Ya-Xian and Du, Hui-Ying. “Research on the net neutrality: The case of Comcast blocking”. In: 2010 (cit. on p. 83).

[214] John Postel. “Internet Control Message Protocol; RFC792”. In: *DARPA Network Working Group* (1981) (cit. on p. 83).

[215] R Pan and B Prabhakar. “CHOKe: A stateless mechanism for providing Quality of Service in the Internet”. In: *Allerton Conference on Communication, Control and Computing*. 1999 (cit. on p. 83).

[216] Fred Baker. *Requirements for IP Version 4 Routers*. RFC 1812. June 1995 (cit. on p. 83).

[217] Fernando Gont. *ICMP Attacks against TCP*. RFC 5927. July 2010 (cit. on p. 83).

[218] Fernando Gont. *Deprecation of ICMP Source Quench Messages*. RFC 6633. May 2012 (cit. on p. 83).

[219] Hiroyuki Ohsaki, Masayuki Murata, Hiroshi Suzuki, Chinatsu Ikeda, and Hideo Miyahara. “Rate-based Congestion Control for ATM Networks”. In: *ACM SIGCOMM Computer Communications Review (CCR)* (1995) (cit. on p. 84).

[220] Shan Huang, Dezun Dong, and Wei Bai. “Congestion control in high-speed lossless data center networks: A survey”. In: *Future Generation Computer Systems* (2018) (cit. on p. 84).

[221] Stefan Alfredsson, Giacomo Del Giudice, Johan Garcia, Anna Brunstrom, Luca De Cicco, and Saverio Mascolo. “Impact of TCP congestion control on bufferbloat in cellular networks”. In: *International Symposium on “A World of Wireless, Mobile and Multimedia Networks” (WoWMoM)*. 2013 (cit. on p. 84).

[222] Janey C. Hoe. “Improving the Start-up Behavior of a Congestion Control Scheme for TCP”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 1996 (cit. on p. 84).

[223] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. “Queues Don’t Matter When You Can JUMP Them!” In: *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 2015 (cit. on p. 85).

[224] Kanon Sasaki, Takahiro Hirofuchi, Saneyasu Yamaguchi, and Ryousei Takano. “An Accurate Packet Loss Emulation on a DPDK-Based Network Emulator”. In: *Asian Internet Engineering Conference (AINTEC)*. 2019 (cit. on p. 86).

[225] F-Stack. *High Performance Network Framework Based on DPDK*. http://www.f-stack.org. [Last accessed: Mai 10, 2024]. 2017 (cit. on p. 86).

[226] Christopher W.F. Parsonson, Joshua L. Benjamin, and Georgios Zervas. “Traffic generation for benchmarking data centre networks”. In: *Optical Switching and Networking Journal* (2022) (cit. on p. 86).

[227] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. “Inside the Social Network’s (Datacenter) Network”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2015 (cit. on pp. 86, 93).

[228] Luke Hsiao, Brooke Krajancich, Philip Levis, Gordon Wetzstein, and Keith Winstein. “Towards Retina-Quality VR Video Streaming: 15ms Could Save You 80% of Your Bandwidth”. In: *ACM SIGCOMM Computer Communications Review (CCR)* 52.1 (Mar. 2022) (cit. on p. 93).

[229] S. Shunmuga Krishnan and Ramesh K. Sitaraman. “Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-experimental Designs”. In: *ACM Internet Measurement Conference (IMC)*. 2012 (cit. on p. 93).

[230] Wei Li, Fan Zhou, Kaushik Roy Chowdhury, and Waleed M Meleis. “QTCP: Adaptive congestion control with reinforcement learning”. In: *IEEE Trans. Network Science and Engineering* (2018) (cit. on p. 98).

[231] Arpit Gupta, Chris Mac-Stoker, and Walter Willinger. “An Effort to Democratize Networking Research in the Era of AI/ML”. In: *SIGCOMM Workshop on Hot Topics in Networking (HotNets)*. 2019 (cit. on p. 99).

[232] Franck Le, Mudhakar Srivatsa, Raghu Ganti, and Vyas Sekar. “Rethinking data-driven networking with foundation models: challenges and opportunities”. In: *SIGCOMM Workshop on Hot Topics in Networking (HotNets)*. 2022 (cit. on p. 99).

[233] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. “Practical GAN-based synthetic IP header trace generation using NetShare”. In: *Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*. 2004 (cit. on p. 99).

[234] *The CAIDA Anonymized Internet Traces Data Access*. https://www.caida.org/catalog/datasets/passive_dataset_download/. [Last accessed: September 03, 2024]. 2019 (cit. on p. 99).

[235] SWIM Project, UC Berkeley. *SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol*. [Last accessed: June 10, 2025]. 2025. URL: https://github.com/SWIMProjectUCB/SWIM/wiki (cit. on p. 99).

[236] Michael Welzl, Peyman Teymoori, Safiqul Islam, David Hutchison, and Stein Gjessing. “Future Internet Congestion Control: The Diminishing Feedback Problem”. In: *IEEE Communications Magazine* (2022) (cit. on p. 99).

[237] Robert McMahon. *Panel discussion: Wi-Fi considerations for congestion control*. https://datatracker.ietf.org/meeting/121/materials/slides-121-iccrg-panel-discussion-wi-fi-considerations-for-congestion-control-02.pdf. Presented at the Internet Congestion Control Research Group (ICCRG) session, IETF 121, Dublin, November 2024. Nov. 2024 (cit. on p. 100).

[238] Ingemar Johansson. *Panel discussion: 5G and congestion control considerations*. https://datatracker.ietf.org/meeting/121/materials/slides-121-iccrg-panel-discussion-5g-congestion-control-considerations-00. Presented at the Internet Congestion Control Research Group (ICCRG) session, IETF 121, Dublin, November 2024. Nov. 2024 (cit. on p. 100).

List of Figures

List of Tables