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

Ayush Mishra[†], Lakshay Rastogi[‡], Raj Joshi[†], and Ben Leong[†]

[†]National University of Singapore [‡]Indian Institute of Technology, Kanpur

ABSTRACT

The Internet congestion control landscape is rapidly evolving. Since the introduction of BBR and the deployment of QUIC, it has become increasingly commonplace for companies to modify and implement their own congestion control algorithms (CCAs). To respond effectively to these developments, it is crucial to understand the state of CCA deployments in the wild. Unfortunately, existing CCA identification tools are not future-proof and do not work well with modern CCAs and encrypted protocols like QUIC. In this paper, we articulate the challenges in designing a future-proof CCA identification tool and propose a measurement methodology that directly addresses these challenges. The resulting measurement tool, called *Nebby*, can identify all the CCAs currently available in the Linux kernel and BBRv2 with an average accuracy of 96.7%. We found that among the Alexa Top 20k websites, the share of BBR has shrunk since 2019 and that only 8% of them responded to QUIC requests. Among these QUIC servers, CUBIC and BBR seem equally popular. We show that *Nebby* is extensible by extending it for Copa and an undocumented family of CCAs that is deployed by 6% of the measured websites, including major corporations like Hulu and Apple.

CCS CONCEPTS

• **Networks** → **Transport protocols**; Public Internet; • **General and reference** → *Measurement*;

KEYWORDS

congestion control; measurement study

ACM Reference Format:
Ayush Mishra, Lakshay Rastogi, Raj Joshi, and Ben Leong. 2024. Keeping an Eye on Congestion Control in the Wild with *Nebby* . In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3651890.3672223

1 INTRODUCTION

The composition of the Internet's congestion control landscape impacts how we size router buffers [16, 58], think about inter-flow fairness [37, 51, 61], and even decide on the deployability of new congestion control algorithms (CCAs) on the Internet [60]. In the past, relatively infrequent snapshots of the Internet's composition were sufficient to understand its congestion control landscape [46, 54]. However, recent developments suggest that CCAs on the Internet are evolving faster than ever before.

The deployment of BBR and its variants is a perfect example of this rapid evolution. While BBR was first introduced back in 2016, the algorithm has continued to evolve over the years. At the time of writing, Google alone is known to have deployed three different versions of BBR [13, 22, 33]. Outside of Google, operators have also been found to deploy modified versions of BBR according to their own needs [48].

The adoption of QUIC [39] on the Internet is another catalyst that has influenced the evolution of the Internet's CCA landscape in recent years. While the QUIC standard itself does not introduce any new CCAs, QUIC congestion control is implemented in the user space and thus makes it significantly easier to implement new CCAs and to deploy modified versions of existing CCAs. There is evidence that operators are already deploying their own variants of CCAs like CUBIC and BBR in their QUIC stacks [47]. These variants can behave very differently from their kernel counterparts.

Given that these developments have major consequences for the Internet's congestion control landscape, it is crucial to keep an eye on CCAs in the wild. Unfortunately, existing CCA identification tools [24, 31, 46, 50, 54, 63] do not work well with modern CCAs and encrypted protocols like QUIC.

In this paper, we revisit the problem of CCA identification from first principles and articulate the key challenges in the context of today's rapidly evolving CCA landscape (§2.1). In particular, we argue that a CCA identification technique needs to be future-proof to handle new and yet unknown CCAs. We also need a new metric that can work well with modern rate-based CCAs. To address these challenges, we propose a principled approach to CCA identification that is extensible by design (§3).

Our approach is implemented as a tool called *Nebby* that uses bytes in flight (`BiF`) instead of the `cwnd` metric used by previous approaches [24, 31, 46, 50, 54, 63]. Our key insight is that since rate-based CCAs use `cwnd` as a safeguard and not an operating point, measuring the `cwnd` is not sufficient for telling them apart. On the other hand, while `BiF` is equivalent to `cwnd` for loss-based CCAs, we show that it allows us to distinguish between different rate-based CCAs. To accurately estimate the `BiF` at the client, we introduce additional latency at a local bottleneck to gain visibility over a larger portion of the pipe between the target server and the client (§3.1). We also found a way to accurately estimate `BiF` for encrypted QUIC traffic (§3.2).

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

ACM SIGCOMM '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0614-1/24/08.
https://doi.org/10.1145/3651890.3672223

sufficient to identify all 12 CCAs available in the Linux kernel v5.18, and BBRv2 [22], with an average accuracy 96.7% (§4.1). Measurements are generally noisy. However, because there are many repeated segments in a trace, we have many opportunities to successfully detect the segments that correspond to different CCAs. To the best of our knowledge, *Nebby* is the first CCA identification tool that can identify CCAs for TCP and QUIC web servers and over a wide range of interactive applications.

We used *Nebby* to measure the Alexa Top 20,000 websites[1] during the period between Jun 2023 and Oct 2023 and made the following findings (§4):

(1) While CUBIC remains the most popular CCA on the Internet, the deployment of CCAs can differ by region for TCP. We found that while TCP BBR's adoption is still substantial, its deployment in regions like Mumbai and Sao Paulo lags behind Ohio and Paris (§4.2).
(2) Comparing our results with those from our earlier measurement study in 2019 [49], we found that BBR's share shrunk among the Alexa Top 20,000 websites since 2019. In fact, some websites have since switched from BBRv1 to CUBIC (§4.2).
(3) BRRv2 was publicly released by Google in 2019. Most new adopters of BBR deployed BBRv2, instead of BBRv1. This is a positive sign, since BBRv2 is less aggressive towards loss-based flows. However, most of the early adopters of BBR (who are still running BBR) continue to run BBRv1, and have not yet migrated to BBRv2 (§4.2).
(4) We detected the testing and deployment of BBRv3 in June 2023 before it was formally announced at the IETF in August 2023. This finding was confirmed by Google and demonstrates that *Nebby* is able to successfully detect the deployment of new and undocumented variants on the Internet by major players (§4.2).
(5) We uncovered a group of websites deploying a class of unknown variants, which we call *AkamaiCC*. These CCAs are characterized by a blocky sending behavior and behave unlike any other known CCAs. Popular websites using AkamaiCC include `apple.com`, `hulu.com`, and `tiktok.com` (§4.3).
(6) We show that QUIC adoption among the Alexa Top 20k websites is still relatively limited, with only about 8% of the websites measured responding to QUIC requests. CUBIC and BBR seem equally popular among the deployed QUIC services (§4.4).
(7) We successfully identified the CCAs used by popular websites to serve video and audio traffic over interactive sessions over a Chrome browser. In addition, we found that it was common for different CCAs to be used for different content. BBR seems to be the CCA of choice for video streaming flows, while CUBIC is often used for static content (§4.5).

2 BACKGROUND & RELATED WORK

In this section, we provide some background on identifying congestion control algorithms (CCAs) on the Internet and describe how previous measurement techniques work. We will explain how this problem has become progressively harder over the years and why previous approaches are no longer viable.

[1]While Alexa has since been shut down, we used the last updated list [12] from February 2023 to do our measurements.

There have been many previous measurement studies to identify CCAs deployed by web servers [24, 31, 46, 50, 54, 63]. In the early 2000s, this was a relatively straightforward problem to solve because the number of available CCAs was small. For example, the earliest of these studies [46, 54] only had to differentiate between Reno [55], New Reno [36], and Tahoe. They could therefore design simple but ad hoc methodologies to differentiate between them. However, as a result, their tool (TBIT) was not easily extensible to work with more sophisticated CCAs like CUBIC [34] and CTCP [59].

These shortcomings with TBIT motivated the design of CAAI [63] in 2011. CAAI was less ad hoc and measured how the `cwnd` of a CCA evolved during a connection. The measurement was made by delaying and batching ACKs sent by the client and taking advantage of the fact that most CCAs are ACK-clocked. However, this technique no longer works once rate-based CCAs, like BBR, were introduced on the Internet. Rate-based CCAs pace their packets, so delaying ACKs will no longer allow us to measure the `cwnd`.

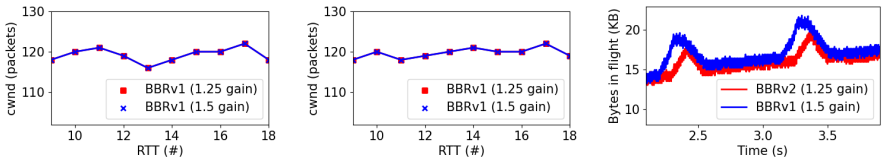
Subsequently, Gordon [50] and Inspector Gadget [31] were proposed. Gordon resorted to dropping packets to estimate the `cwnd`, while Inspector Gadget continued to use delayed ACKs to estimate the `cwnd` of a sender over a variety of network conditions. Both these tools were ultimately able to classify BBR, but they each had their own shortcomings. Gordon's strategy of dropping packets repeatedly over hundreds of connections is too aggressive and is now blocked by most websites on the Internet. We attempted re-running Gordon and were only able to successfully identify 4% of the Alexa Top 10,000 websites (details can be found in Appendix A). Inspector Gadget's machine learning-based classifier was prone to overfitting and could not detect the deployment of new and unknown CCAs. For example, even though both Gordon and Inspector Gadget performed their measurements in 2019, Gordon was able to catch the deployment of a proprietary CCA by Akamai [50] while Inspector Gadget was not able to do so.

In summary, a common critique of previous CCA identification tools is that none of them are sufficiently future-proof, and they risk quickly becoming obsolete as new and more sophisticated CCAs are deployed on the Internet. *Nebby* is designed to be future-proof and we adopt a number of strategies to ensure that it can be easily extensible in the future. In §2.1, we discuss the challenges associated with designing a future-proof CCA identification tool and briefly describe how we address each of them in developing *Nebby*.

Inferring CCA Properties. There have also been several previous works that try to infer the properties of CCAs rather than identifying them. Jaiswal et al. used `cwnd` and RTT estimates to understand the impact of a CCA on the throughput and RTT on the Internet [41]. Hagos et al. use machine learning over passively collected traces to infer the state of a CCA [35]. Ferreira et al. developed Mister 880 to reverse engineer CCAs seen in the wild [27]. While in theory Mister 880 could be extended to identifying CCAs on the Internet, it only works for relatively straightforward AIMD CCAs. All these works are promising approaches in their respective domains, but they cannot be applied directly to CCA identification.

2.1 Why CCA Identification is Hard

At a high level, CCAs can be distinguished from each other based on how they react to different network conditions. Therefore, the



(a) Gordon [50] **(b) Inspector Gadget [31]** **(c) *Nebby***

Figure 1: Comparison of `cwnd` to `BiF` for BBRv1's `ProbeBw` phase.

basic approach of identifying CCAs on the Internet is relatively straightforward. We need to emulate different *network profiles* while connecting to a web server and measure how it reacts to them using some metric. Here, we define the network profile as some combination of bandwidth and delay constraints enforced by the measurement tool on the connection.

However, the devil is in the details, and there are currently a large number of CCAs that are available and deployed on the Internet today. Moreover, there are additional challenges associated with identifying CCAs on the Internet in a future-proof way. In this section, we articulate the key challenges for CCA identification and the requirements for further extensibility.

Establishing Causality. To accurately identify a CCA, we need to be confident that the response seen is indeed caused by our network profile and not any other natural variation in the network. This was less of a problem for early tools that used network profiles that generated responses that were unlikely to naturally occur on the Internet. For example, TBIT [46, 54] drops all the packets after initiating a connection and CAAI [63] delays ACKs by 1 second. Since both these behaviors rarely happen on the Internet, the responses they elicit can be safely assumed to be caused by the measurement tool and not the network. However, as discussed earlier, this is not effective when classifying more sophisticated CCAs. To address this, more recent tools like Inspector Gadget [31] and Gordon [50] create a localized bottleneck and apply more generic network profiles to it. Since most network variations happen at the bottleneck, this limits the possibility of natural variation on the Internet impacting the connection. *Nebby* also adopts this strategy to ensure causality is maintained between the network profile and the CCA's response.

Handling Noisy Measurements. Cross-traffic and network bottlenecks between the probing server and the target server will naturally introduce noise in the measurements. There is a need to eliminate noisy measurements. The general approach is to repeat measurements to eliminate outliers [49]. Gordon also attempted to do so by repeating measurements from different vantage points [49]. *Nebby* also repeats a measurement up to 5 times if it is not able to successfully classify it, albeit from the same vantage point. This will allow us to determine if websites deploy different CCAs in different regions. Within each trace, we have many repeated segments, so we have many opportunities to pick out the characteristic shape of the CCA even when there is noise.

Handling New & Unknown CCAs. Given that we expect more new and unknown CCAs to be deployed in the wild, a modern CCA identification tool must also be able to provide insight into the behavior of CCAs that it is not able to classify. Subsequently, it should be able to determine that they are substantively different from known CCAs. This is becoming increasingly important with the deployment of new and modified CCAs in QUIC [47]. The early tools were mainly classification tools [54, 63] that attempted to classify CCAs among a known set of CCAs and were not able to detect new variants. Machine-learning-based like Inspector Gadget [31] and the work of Chen et al. [24] fare no better. Gordon was the first tool that conclusively detected an unknown and undocumented variant, by showing that Akamai deployed their own CCA variant [50], which we referred to as AkamaiCC.

Probe Traffic Cannot Seem Hostile. Padhye and Floyd had clearly articulated that a CCA identification tool must not generate traffic that would be construed to be malicious to a web server [54]. In this light, it was surprising that Gordon [50] even worked at all in 2019, given that it opens connections hundreds of times and drops a large number of packets. Modern DDoS defenses have since kicked in and so Gordon is blocked for many websites. *Nebby* adopts a mostly lightweight approach when probing websites, and does not even introduce packet drops, unlikely previous tools[50, 54, 63].

Need for a Good Metric. Any response to a given network profile must be measured as a change in some metric, like the sending rate or the `cwnd`, which is defined as the maximum number of unacknowledged packets. The sending rate of a flow is hard to measure accurately since it will be modified by every bottleneck it encounters on its path to the receiver. All previous tools therefore measure how the `cwnd` of a CCA changes during a connection. However, while `cwnd` was shown to be sufficient for identifying window-based CCAs and BBR [50], measuring the `cwnd` is not sufficient to differentiate between rate-based CCAs.

The reason why `cwnd` does not work well for rate-based CCAs is that `cwnd` is typically used by rate-based CCAs as a safeguard and not an operating point. Given that using the `cwnd` can mask a sender's true behavior and estimating their sending rate is not practical, *Nebby* elects to measure bytes in flight (`BiF`), which is defined as the instantaneous number of unacknowledged packets, to determine a target server's response to a network profile (see details in §3.1).

In Figure 1, we plot the `cwnd` and `BiF` for two versions of BBR with the same `cwnd` but different pacing gains (1.25 and 1.5). It is clear that `cwnd` measurements (from Gordon and Inspector Gadget) do not allow us to differentiate between these two different versions of BBR, while `BiF` measurements (from *Nebby*) allow us to tell them apart. Since all previous CCA identification tools [24, 31, 46, 50, 54, 63] measure `cwnd` changes, they will not be extensible to the current crop and future iterations of rate-based CCAs.

Handling encrypted packets. The latest challenge posed to identifying CCAs on the Internet is transport protocols like QUIC.

Table 1: Properties of CCA Identification tools.

Tool	Primary Design Goals				Extensibility Requirements		
	Causality	Robustness to Noise	Identify Unknown CCAs	Cannot seem Hostile	Good Metric	Works with Encryption	Client Agnostic
TBIT [46, 54]	✗	✗	✗	✓	✗	✗	✗
CAAI [63]	✗	✗	✗	✓	✗	✗	✗
IG [31]	✓	✓	✗	✓	✗	✗	✗
Gordon [50]	✓	✓	✓	✗	✗	✗	✗
Nebby	✓	✓	✓	✓	✓	✓	✓

Since all previously discussed CCA identification tools were designed for TCP, they utilize the sequence and ACK numbers that are exposed unencrypted in the TCP header. However, QUIC packets are completely encrypted and only expose the source and destination IPs and the flow ID. Therefore, previous approaches are not directly extensible to QUIC. To support non-TCP and QUIC flows, we need to be able to measure the response to a network profile without the need for sequence numbers and ACK numbers.

Need to be Client Agnostic. Earlier tools [24, 31, 46, 50, 54, 63] were able to classify traffic serving only a small set of TCP-based clients (like `wget` and `curl`). Since the choice of CCA can be expected to be influenced by the kind of application it supports, an ideal CCA identification tool needs to support a large variety of clients. In fact, for a modern CCA identification tool to be extensible and future-proof, there is a need for the tool to be client-agnostic.

Summary. In Table 1, we summarize how *Nebby* compares with existing tools in addressing the various challenges and extensibility requirements. Our key contributions are twofold: we found a more accurate metric (`BiF`) that works well even for rate-based TCP variants and also added support for identifying CCA variants deployed on modern QUIC stacks and web browsers.

3 METHODOLOGY

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

The captured trace is then processed with a classifier (§3.4) to identify the CCAs. By decoupling capturing of the trace and the classification process, we make it possible for the accuracy of the system to be improved incrementally without having to re-do our measurements. Our high-level approach is no different from previous approaches.

The key innovation in our work is in how we address the challenges laid out in §2.1 to ensure backward compatibility and future extensibility, as follows:

(1) Instead of attempting to estimate `cwnd`, we estimate `BiF`[2] by introducing additional delay at the capture point (§3.1);
(2) We developed techniques to handle QUIC packets (§3.2);
(3) We use a minimal set of two network profiles that we show is sufficient to identify all 12 TCP variants in Linux kernel v5.18 and BBRv2 (§3.3);

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

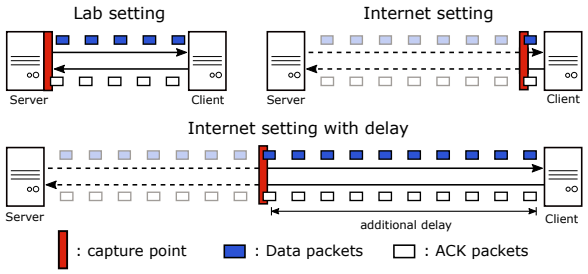


Figure 2: Using additional delay to increase the ratio of *visible* in-flight packets.

(4) We designed an extensible classifier that can identify all existing TCP variants in Linux kernel (§3.4) and show that it can be easily extended to identify new TCP variants (§4.3); and
(5) Our approach can also be extended to support new applications and multiple flows (§3.5).

3.1 Estimating Bytes in Flight (`BiF`)

Consider a server and a client as shown in Figure 2. In a controlled lab setting, we can measure the `BiF` of a flow by setting up the capture point near the server because everything between the capture point and the client is visible. This is not possible for a remote server on the Internet, since the capture point can only be set up near the client. In such a situation, it is difficult to estimate the `BiF` accurately since the majority of the packets in the pipe are not visible.

Our key insight is that if we artificially introduce additional delay between the capture point and the client, we will then have visibility over a larger portion of the pipe and hence will be able to estimate `BiF` more accurately. In particular, because current Internet latencies are small (~20 ms [57]), by introducing a latency of x ms, we can effectively have visibility over $\frac{x}{x+20}$ of the pipe.

To determine the additional delay required, we set up control servers running different CCAs on AWS and measured their `BiF` with different amounts of delay introduced. We then compared these measurements with the ground truth `BiF` values exported directly from the sockets of these control servers. We plot the results of these experiments in Figure 3 for a server running CUBIC, Reno, and BBR. From these results, we see that the accuracy is close to 100% when the total additional delay is larger than 90 ms. Beyond this, accuracy might even drop. We found that this trend was consistent for all the CCAs available in the Linux kernel.

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

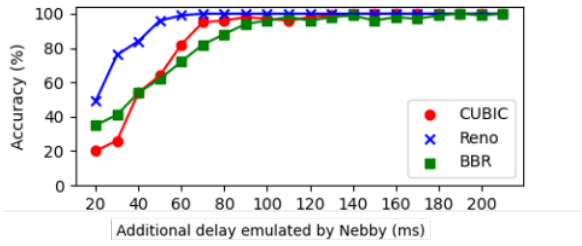


Figure 3: Impact of the additional delay on accuracy.

3.2 Handling QUIC packets

QUIC packets are encrypted and sequence numbers are not visible. In fact, there is no way to tell if a QUIC packet is a data packet or an ACK packet from the raw packet trace. Even if we were able to make this distinction, it is difficult to estimate how many bytes each ACK packet is acknowledging without access to the sequence numbers.

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

We exported the actual `BiF` logs from a `quiche` [3] sender running on an AWS instance and then compared it with what *Nebby* was measuring from a client machine. We ran this experiment for 20 trials in two different AWS regions. We found the accuracy of our `BiF` estimates for QUIC to be higher than 97%.

3.3 Minimal Set of Network Profiles

To recap, a *network profile* is a combination of some bandwidth, delay, and buffer constraints applied at the capture point, together with some actions like packet drops. Each network profile is an opportunity to differentiate between CCAs based on how they respond to that network profile. It is entirely possible that two different CCAs may have the same response to a network profile. In such cases, we need to perform measurements over additional network profiles that can help us differentiate between them. Therefore, the goal is to find the minimum number of network profiles required to differentiate between all the CCAs. Given that there are currently 12 available TCP variants in the Linux kernel v5.18 (and BBRv2 in Google's custom kernel), it was not surprising that we were not able to find one network profile that was able to allow us to distinguish between all of them.

However, we found that we were able to identify all 13 known CCAs with just 2 network profiles and without the need to introduce arbitrary packet drops, unlike previous approaches [49, 63]. By not introducing arbitrary packet drops, it not only simplifies the design of the network profile but also makes it less likely that our connection would be perceived as malicious by a remote sender. However, *Nebby* does allow *natural* packet drops to happen when there is a buffer overflow at its localized bottleneck buffer.

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

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

Set of 2 Network Profiles. *Nebby* makes all measurements with a bottleneck bandwidth of 200 Kbps and the bottleneck buffer set at 2 BDP. We found that by introducing a 50 ms one-way delay using Mahimahi [53], the network profile would result in visually distinct `BiF` graphs for all 13 known CCAs. However, since some CCAs can have similar shapes under this network profile (like New Reno, Illinois, and HSTCP (see Figures 4g, 4h, and 4d)), we also added an additional network profile with a larger delay (100 ms one-way delay). We plot the `BiF` vs time graph for all the CCAs in the Linux kernel under both these network profiles in Figure 4.

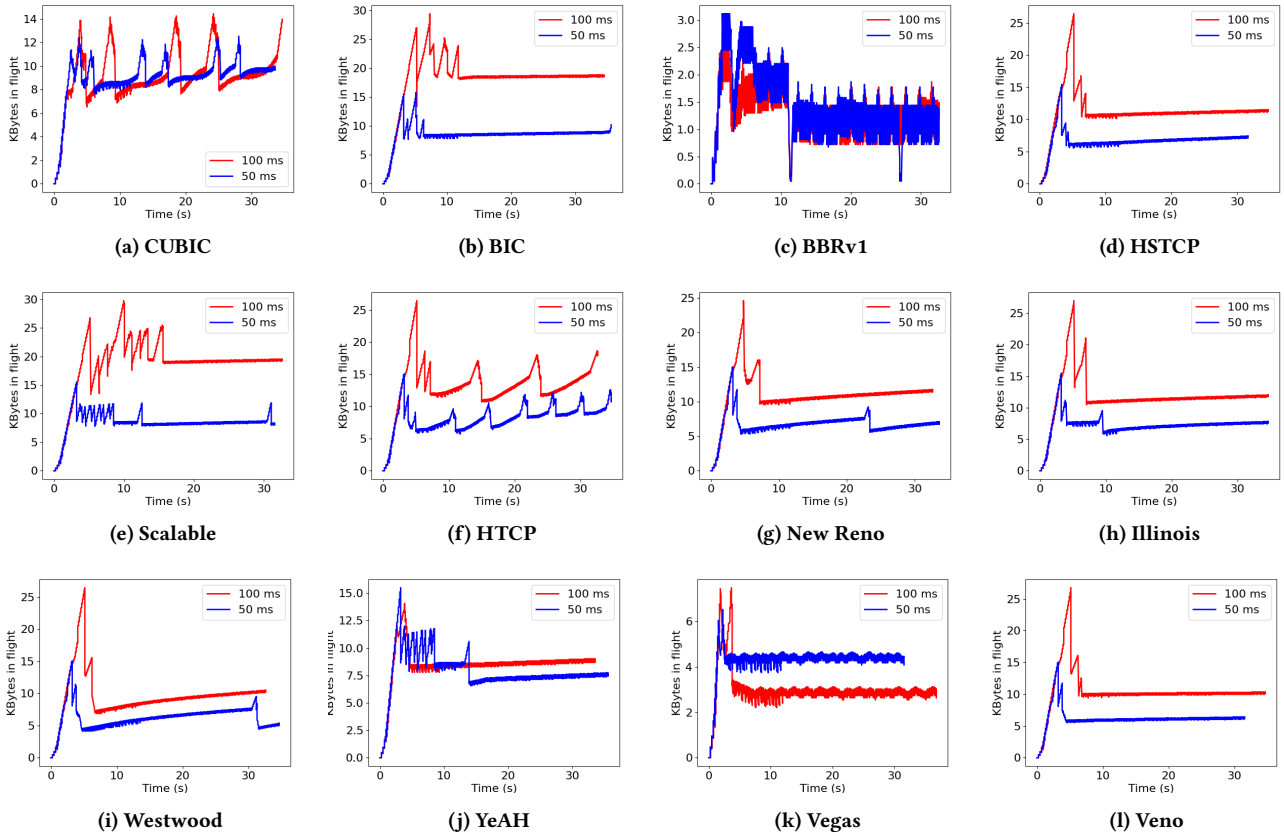
3.4 Designing an Extensible Classifier

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

To exploit this periodicity in a CCA's behavior to classify existing TCP variants currently available in the Linux kernel, including BBRv2[22], our classifier extracts these periodic behaviors from `BiF` measurement traces and classifies them as different CCAs based on their periodicity and *shape* (see Figure 5). For the set of known CCAs in the kernel, we developed 2 classifiers: one for loss-based CCAs and another for BBRv1/v2. We can easily extend *Nebby* to identify more CCA variants by adding new classifiers that can be run concurrently with our 2 current classifiers. We describe how this works in §4.3.

Before a trace can be used for classification, we first remove the noise with a low-pass filter ("smoothening") and segment the smoothened trace into several chunks. These segments are then sent to the different (currently 2) classifiers. This process is summarized in Figure 6. We describe the various components in detail below.

(1) **Smoothening**. The raw `BiF` traces captured can be noisy because of ACK compressions and cross traffic on the Internet. To



(a) CUBIC (b) BIC (c) BBRv1 (d) HSTCP

(e) Scalable (f) HTCP (g) New Reno (h) Illinois

(i) Westwood (j) YeAH (k) Vegas (l) Veno

Figure 4: Traces of TCP congestion control algorithms in the current Linux kernel.

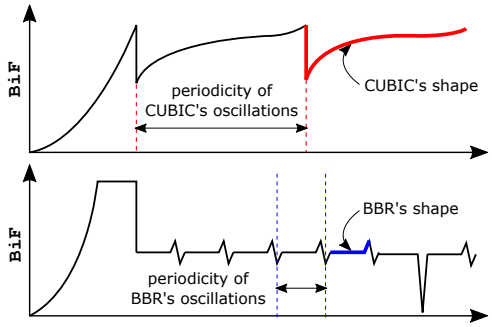


Figure 5: Characteristic features of periodic oscillations for CUBIC and BBR.

remove this noise, we remove all the variations that happen at timescales that are shorter than an RTT (since they are likely to be introduced by the network and not the CCA) by applying an FFT on the `BiF` values and removing all component frequencies that are larger than $\frac{1}{RTT}$.

(2) **Segmentation**. Since most CCAs have similar slow start phases, we ignore slow start and attempt to identify CCAs using only their behavior during congestion avoidance. As discussed earlier, CCAs typically exhibit periodicity during congestion avoidance, where they periodically probe for more bandwidth, and then eventually back off when they encounter congestion, or if the buffer overflows. We extract these periodic waveforms as individual segments that are punctuated by periods of 'back-offs,' by computing the first derivative over the `BiF` trace and identifying the back-offs by their characteristic high negative gradients. We extract the regions between these back-offs as individual segments for a given trace. A typical trace generally gets divided into multiple segments.

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

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

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

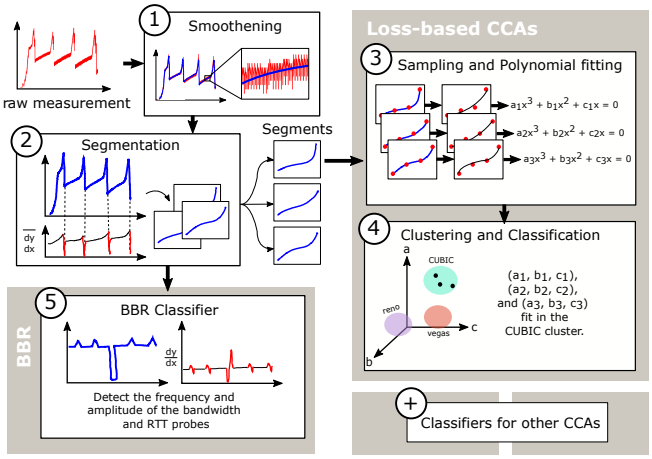


Figure 6: How *Nebby*'s Classifier works.

Table 2: Different degree clusters with their CCAs.

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

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

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

To generate the control data required to derive the required coefficients for the known CCAs, we set up control servers in AWS instances in Singapore, Mumbai, Ohio, Paris, and Sao Paulo and measured them from a host in our laboratory using *Nebby*. The measurements were made using the two network profiles described in §3.3. Each CCA was run 50 times from each vantage point, giving us a total of 250 measurements for each CCA. For each CCA, we determined the representative polynomial using `polyfit` as previously described, giving us a large number of polynomials for each CCA. Since each measurement can consist of multiple segments, we were able to derive hundreds of polynomials for each CCA from

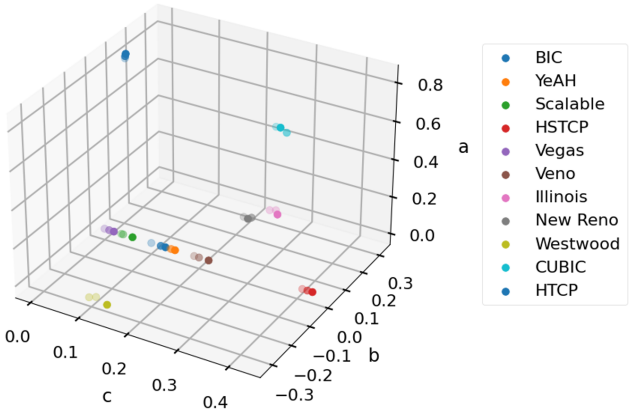


Figure 7: Coefficients for the polynomials ($ax^3+bx^2+cx+d=0$) of all the loss-based CCAs form distinct clusters.

our 250 measurements. The polynomials assigned to the segments for the known CCAs in the Linux kernel are listed in Table 2. The clusters formed by all the coefficients of these polynomials are illustrated in Figure 7. We can see that with our chosen network profiles, the loss-based CCAs formed distinct clusters that allowed us to tell the different CCAs apart. In Appendix B, we provide a detailed description of how we match a segment's coefficients with those of known CCAs. The details are omitted here due to space constraints.

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

- BBRv1 has a characteristic bandwidth probing behavior where it increases the sending rate by 25% every 8 RTTs (i.e. `ProbeBW`). This behavior is clearly visible to *Nebby* (as seen in Figure 1c) and easily detectable by looking for periodic spikes in the first derivative w.r.t. to the time of the extracted segments. BBRv1 also backs off every 10 seconds in order to estimate the minimum RTT of the path (i.e. `ProbeRTT`). Therefore, if we see a rate-based sender probe for bandwidth every 8 RTTs and backing off every 10 seconds, we conclude that the sender is BBRv1.
- The probing periods for BBRv2 are less well-defined. After slow start, BBRv2 typically enters its bandwidth cruise phase where it sends at the bottleneck bandwidth without any probing for a period that depends on the BDP. For our network settings, this probe period is about 2 seconds. Like BBRv1, BBRv2 also backs off periodically to measure the minimum RTT, albeit every 5 seconds. Therefore, if we see a rate-based sender that is stable during congestion avoidance for at least 2 seconds and backs off every 5 seconds, we conclude that the CCA must be BBRv2.

Handling New & Undocumented CCAs. Given the design of our classification framework, it can extended naturally in 3 ways: (i) existing classifiers can be modified to use the segment corresponding to Slow Start; (ii) an additional classifier can be constructed from observing the properties of a new CCA (an example is described in §4.3); or (iii) additional network profiles can be added.

Table 3: Classification accuracy (Confusion Matrix).

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

3.5 Supporting Web Browsers & Multiple Flows using Selenium

One of *Nebby*'s advantages over previous tools is that it can work with a large range of applications. In addition to TCP measurements using `wget` and QUIC measurements using a `quiche` client, *Nebby* can also classify a flow generated by a live web browser. We used a simple Selenium[15] wrapper written in about 25 lines of Python code to launch connections and stream dynamic web content like video via a Google Chrome browser.

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

4 EVALUATION

In this section, we evaluate *Nebby*'s accuracy and present our results for measurements over `wget` (TCP), `quiche` (QUIC), and a selenium web browser (TCP). All measurements were done from five viewpoints (AWS data centers) around the world, namely Ohio, Paris, Mumbai, Singapore, and Sao Paulo. All traces, unless specified, were collected between June 2023 and October 2023. *Nebby* was implemented in 100 lines of Bash and 900 lines of Python. *Nebby* is open-source and available on Github [14].

4.1 Measurement Accuracy and Usability

In order to determine the accuracy of our classifier, we set up controlled web servers on the AWS cloud in 5 regions around the world: Ohio, Paris, Mumbai, Singapore, and Sao Paulo. Each of these control servers was configured to run all the CCAs variants available in the Linux kernel v5.18. We then used *Nebby* (running locally on a machine in the lab) to measure and classify each of these web servers 10 times per congestion control algorithm. We then reversed the configuration, with the test server run locally and *Nebby* making measurements from the AWS instances - giving us a total of 100 trials per congestion control algorithm. These measurements were then classified using our classifier and the resulting classification accuracy is shown in Table 3. We achieved an average accuracy of 96.7%. The level of accuracy is comparable to Gordon [50], except that *Nebby* can achieve slightly better accuracy for CUBIC and BBR. While both Gordon and *Nebby* use shape-based classifiers, Gordon's measurements are much more coarse-grained (one data point per RTT) compared to *Nebby* (one data point per packet). As a result, *Nebby* can distinguish between all the variants in the Linux kernel, unlike Gordon, which was not able to distinguish between some pairs of TCP variants like Veno and Vegas [19]. As discussed in §2.1, this is because `BiF` is a better metric than the `cwnd` metric used by Gordon.

In terms of usability, in 2023, Gordon was able to identify CCAs for only 4% of Alexa Top 20k websites when measured from the Singapore region. This is because it creates traffic patterns deemed hostile by websites (discussed in §2.1). *Nebby* on the other hand identified ~78% of Alexa Top 20k websites when measured from the Singapore region (Table 4).

4.2 Results for Alexa Top 20k Websites

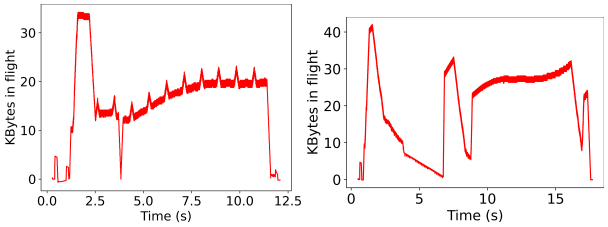
To understand how the Internet's congestion control landscape has evolved since our previous measurement study done in 2019 [49], we used *Nebby* to identify the CCA variants for the Alexa Top 20,000 websites from the 5 aforementioned viewpoints (Ohio, Paris, Mumbai, Singapore, and Sao Paulo). These measurements were made using a `wget` client over TCP. We crawled all the target websites for the largest web pages we could find to record the longest possible measurement traces. We present our results in Table 4.

By comparing our results to those from [49], we make the following observations:

(1) **Different deployments across regions.** In 2019, we observed that websites deployed the same CCAs in all 5 regions [49]. Our latest measurements using *Nebby* suggest that this is no longer the case. From Table 4, it is clear that some websites deploy variants like CUBIC, BBR, and Reno differently in different regions. In particular, we found that 13.6% of the websites were deploying different variants in different regions. About half of these websites (7% of the total) were electing to use CUBIC in Mumbai and/or Sao Paulo while running BBR in all other regions. An example of one such website is `amazon.com` which

Table 4: Distribution of CCA variants among Alexa Top 20k websites measured from different viewpoints by *Nebby*.

Variant	Ohio		Paris		Mumbai		Singapore		Sao Paulo	
	Websites	Share	Websites	Share	Websites	Share	Websites	Share	Websites	Share
BBRv1	2,594	13%	1,900	9.5%	1,635	8%	2,541	12.7%	1,280	6.4%
BBRv2	515	2.6%	373	1.9%	12	0.1%	251	1.3%	0	0%
BIC	712	3.5%	807	4%	837	4.2%	402	2%	227	1.1%
CUBIC	8,202	41%	8,406	42%	8,822	44.1	8,673	43.4%	6,982	34.9%
HSTCP	0	0%	0	0%	0	0%	0	0%	0	0%
HTCP	583	2.9%	370	1.9%	522	2.6%	421	2.1%	363	1.8%
Illinois	721	3.6%	684	3.4%	1,121	5.6%	625	3.1%	229	1.1%
New Reno	1,840	9.2%	1,509	7.5%	3,032	15.2%	2,093	10.5%	2,683	13.5%
Vegas	878	4.4%	421	2.1%	301	1.5%	526	2.6%	511	2.5%
Veno	112	0.6%	382	1.9%	52	0.3%	21	0.1%	11	0.1%
Westwood	201	1%	170	0.9%	0	0%	0	0%	0	0%
Scalable	18	0.1%	0	0%	0	0%	6	0%	0	0%
YeAh	123	0.6%	89	0.4%	64	0.3%	0	0%	112	0.6%
Unknown	3,501	17.5%	4,889	24.4%	3,602	18.1%	4,441	22.2%	7,602	38%
Unresponsive	0	0%	0	0%	0	0%	0	0%	0	0%
Total	20,000	100%	20,000	100%	20,000	100%	20,000	100%	20,000	100%



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

Figure 8: Traces for `amazon.com` in different regions.

was served in all regions using BBRv1 except Mumbai, where we found it being served using CUBIC (see Figure 8).

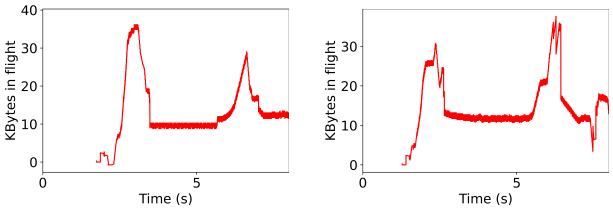
(2) **Websites deploying New Reno**. We had earlier in 2019 reported that New Reno was deployed on only 0.8% of websites. In contrast, *Nebby* found New Reno to be deployed on some 11.1% of the 20,000 surveyed websites. On further investigation, we found that among these 11% of websites, 5% were previously unclassified ('Unknown') and 4% were not present in the earlier list of Alexa top 20,000 sites. A small number (1%) were earlier classified by Gordon as Vegas.

(3) **Adoption of BBRv1.** We found a slight dip in the absolute number of websites that choose to deploy BBR in the Alexa Top 20,000 websites. Even in Ohio, which is the region with the largest deployment of BBR, the number of websites using BBR has dropped from 18% in 2019 to 15.5% in 2023. That said, it should be noted that the Alexa list [38] has evolved significantly since the last measurement study. Overall, we only share 52% of the same measured websites in 2019. Among the common websites, 12% of them (6% of the total) have migrated from using BBR in 2019 to CUBIC in 2023. A large number of these websites (9%, about 4.5% of the total) are hosted by Cloudflare. Some of these notable websites are `bbc.com` and `wikihow.com`. It remains true that most websites choosing to deploy BBR tend to serve video workloads, adult content, or large files (for example, `mega.nz`). We summarize the CCAs deployed by some of these 'heavy-hitter' websites in Table 5.

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

Table 5: CCAs deployed by most popular websites on the Internet by traffic-share.

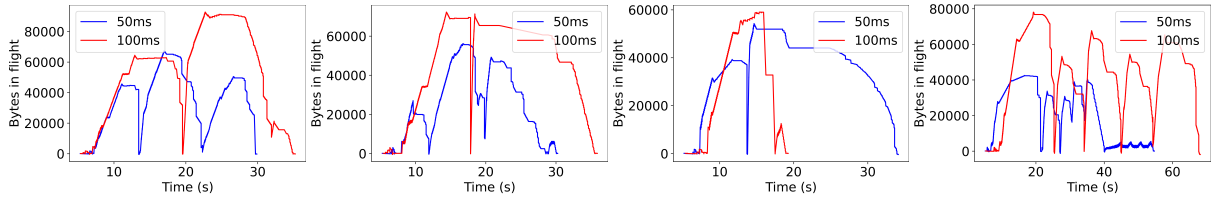
Websites	Traffic share [56]	CCA
`google domains`	13.85%	BBRv3
`netflix.com`	13.74%	Reno
`facebook.com`	6.45%	CUBIC
`apple.com`	4.59%	AkamaiCC
`disneyplus.com`	4.49%	CUBIC
`amazon.com`	4.24%	BBRv1
`tiktok.com`	3.93%	AkamaiCC
`primevideo.com`	2.67%	BBRv2
`hulu.com`	2.44%	AkamaiCC



(a) `google.com` (Jun 2023) **(b) BBRv3 (Aug 2023)**

Figure 9: Catching the deployment of BBRv3 in the wild.

Catching the deployment of BBRv3. During our measurements in June 2023 we noticed that all of the `google` domains and `youtube.com` were deploying a version of BBR that was neither BBRv1 or BBRv2. We hypothesized that what *Nebby* had actually measured in June 2023 was an early deployment of BBRv3, which was released to the community only in August 2023 (See Figure 9). We confirmed this finding with Google [32].



(a) `apple.com` **(b)** `hulu.com` **(c)** `tiktok.com` **(d)** `pornhub.com`

Figure 10: Traces for websites deploying AkamaiCC.

4.3 Extending *Nebby* to identify new CCAs

In 2019, we found that Akamai deployed an undocumented variant that we referred to as AkamaiCC [50]. In 2023, we confirmed that websites hosted by Akamai continue to deploy an undocumented variant. Two of these sites were Apple and Hulu and the corresponding traces are shown in Figures 10a and 10b. As shown in these figures, the defining characteristic of this undocumented CCA is that it would typically send data at some fixed rate for several seconds before backing off. This backoff was not triggered by dropped packets or any bandwidth limits. The fixed send rate did not seem to be determined by either the BDP or the RTT. This behavior is different from what we observed in 2019, where `cwnd` was found to be proportional to the BDP. It is therefore likely that AkamaiCC has evolved since the last measurement study in 2019, or Akamai might have deployed a CCA different from the one that was deployed in 2019.

Given these observations, we wrote a pluggable classifier for *Nebby* that detected if a flow backed off in intervals between 10 to 20 s, and maintained `BiF` at a steady level between these back-offs. Since we do not have the ground truth for AkamaiCC, the classification parameters were determined from traces obtained from 10 known Akamai-hosted websites that *Nebby* originally classified as 'Unknown'. We add this new AkamaiCC classifier to our original set of 2 classifiers (loss-based and BBR). By running this new classifier over our full data set, the CCAs for all the known Akamai-hosted websites (approximately 6%) were identified as AkamaiCC. This demonstrates that *Nebby* is easily extensible to new CCAs beyond the known and documented CCAs. In addition to these Akamai-hosted websites, we also found another 1% of websites (that were not hosted by Akamai) that deployed an AkamaiCC-like variant. Two such examples (TikTok and Pornhub[3]) are shown in Figures 10c and 10d.

4.4 CCA Implementations in QUIC Stacks

QUIC [39] is quickly gaining popularity on the Internet and is set to become the standard transport with HTTP3. Meta already supports 75% of its traffic using its `mvfst` QUIC stack [42]. We had earlier shown that the implementations of standard TCP variants in existing QUIC stacks can be quite different from that in the current Linux kernel [47].

We investigated 11 QUIC stacks with a total of 22 different implementations of CUBIC, Reno, and BBR (See Appendix C for more details). Since QUIC implementations can behave significantly differently from their kernel counterparts, we re-evaluated the accuracy of our classifier for all these QUIC CCA implementations.

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

In Table 7, we produce the confusion matrix for the classification of the CCA implementations in these QUIC stacks. In general, our classifier works very well for most of the CCA implementations in the investigated QUIC stacks.

We investigated if *Nebby* was able to identify the CCA variants implemented by the existing QUIC stacks accurately by measuring *Nebby*'s accuracy for the 22 CCA implementations of CUBIC, BBR, and Reno for 11 open-sourced QUIC stacks. We found that *Nebby* achieved an average accuracy of 92.8%. Our accuracy is not as high for `quiche` CUBIC (78%), `xquic` Reno (80%), and `mvfst` BBR (86%), but this is hardly surprising since these variants had earlier been identified to be non-conformant QUIC CCA implementations [47]. By *non-conformant*, we mean that their behavior deviates significantly from their respective kernel CCA implementations. We added the Conformance [47] for all the benchmarked QUIC CCA implementations as an extra column in Table 7 for reference.

To investigate the CCAs deployed by websites that support QUIC, we repeated measurements of the Alexa Top 20,000 website by sending requests using `quiche`'s QUIC client using *Nebby*. We found that only 8.9% of the 20,000 sites responded to QUIC requests. Most of these websites supporting QUIC were hosted by Cloudflare or were Facebook domains. All these websites also deployed the same congestion control algorithms they deployed over TCP. Our results are summarized in Table 6. We found no evidence of undocumented variants being deployed on QUIC stacks. When we studied the traces for the websites that were classified as 'Unknown,' we found that the reason why they could not be classified was because of noisy measurements.

4.5 Video Measurements with *Selenium*

In addition to measuring websites over `wget` and QUIC, we also measured websites using a standard web browser as described in §3.5. One of *Nebby*'s advantages over its predecessors is that it allows us to study streaming and interactive applications. We measured popular websites while streaming video on demand (VOD), live video, audio, and while being on a video call. Since such applications tend to open multiple concurrent connections, we use a modified version of *Nebby* that assigns separate bottleneck queues to each flow. We summarize our observations below (see Table 8).

BBR is the preferred CCA for video traffic. To investigate if certain CCAs were preferred for certain asset types, we identified which flows were serving which elements in the webpage and correlated their CCAs with them. We found that most audio and video streaming websites like Primevideo, AppleTV, Spotify, Apple Music, YouTube, Douyin, Bilibili, Twitch, HBO, and Hotstar used some version of BBR to stream audio and video. However, there were a few exceptions to this rule, namely Netflix (New Reno), TikTok and Hulu (AkamaiCC), and Disney+ and Jiocinema (CUBIC).

Table 6: Distribution of QUIC CCA variants as measured from different viewpoints on the Internet.

Variant	Ohio		Paris		Mumbai		Singapore		Sao Paulo	
	Websites	Share	Websites	Share	Websites	Share	Websites	Share	Websites	Share
CUBIC	622	3.1%	829	4.1%	927	4.6%	796	4%	403	2%
BBR	1,036	5.2%	703	3.5%	197	1%	268	1.3%	479	2.4%
BBRv2	0	0%	0	0%	0	0%	0	0%	0	0%
New Reno	25	0.1%	10	0%	11	0%	18	0.1%	10	0%
Unknown	101	0.5%	242	1.2%	847	4.2%	702	3.5%	892	4.5%
Unresponsive	18,216	91.1%	18,216	91.1%	18,018	90.1%	18,216	91.1%	18,216	91.1%
Total	20,000	100%	20,000	100%	20,000	100%	20,000	100%	20,000	100%

Table 7: Confusion Matrix for QUIC CCA variants.

		Classified as														
Organization	Variant	BBRv1	BBRv2	BIC	CUBIC	HSTCP	HTCP	Illinois	New Reno	Scalable	Vegas	Veno	Westwood	YeAH	Unknown	Conf.[47]
Alibaba	`xquic` CUBIC	0%	0%	0%	**88%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	12%	0.55
AWS	`s2n-quic` CUBIC	0%	0%	3%	**92%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	5%	0.76
Cloudflare	`quiche` CUBIC	0%	0%	12%	**78%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	10%	0.08
Go	`quicgo` CUBIC	0%	0%	12%	**84%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	4%	0.87
Google	`chromium` CUBIC	0%	0%	6%	**94%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0.6
H2O	`quicly` CUBIC	0%	0%	0%	**82%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	18%	0.68
LiteSpeed	`lsquic` CUBIC	0%	0%	4%	**92%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	4%	0.95
Meta	`mvfst` CUBIC	0%	0%	0%	**100%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0.9
Microsoft	`msquic` CUBIC	0%	0%	0%	**98%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	0.7
Mozilla	`neqo` CUBIC	0%	0%	7%	**88%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	5%	0
Rust	`quinn` CUBIC	0%	0%	0%	**90%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	10%	0.7
Alibaba	`xquic` BBR	**100%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0.15
Google	`chromium` BBR	**100%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0.7
LiteSpeed	`lsquic` BBR	**100%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0.59
Meta	`mvfst` BBR	**86%**	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	14%	0
Alibaba	`xquic` Reno	0%	0%	0%	0%	0%	0%	0%	**80%**	0%	0%	0%	0%	0%	20%	0.38
Cloudflare	`quiche` Reno	0%	0%	0%	0%	0%	0%	0%	**100%**	0%	0%	0%	0%	0%	0%	0.8
Go	`quicgo` Reno	0%	0%	0%	0%	0%	0%	0%	**98%**	0%	0%	0%	0%	0%	2%	0.92
H2O	`quicly` Reno	0%	0%	0%	0%	0%	0%	0%	**100%**	0%	0%	0%	0%	0%	0%	0.8
Meta	`mvfst` Reno	0%	0%	0%	0%	0%	0%	0%	**100%**	0%	0%	0%	0%	0%	0%	0.94
Mozilla	`neqo` Reno	0%	0%	0%	0%	0%	0%	0%	**92%**	0%	0%	0%	0%	0%	8%	0.62
Rust	`quinn` Reno	0%	0%	0%	0%	0%	0%	0%	**100%**	0%	0%	0%	0%	0%	0%	0.96

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

Inter-flow interaction for the same websites. It is well known that CUBIC and BBR flows do not mix well [51, 61]. Therefore, when we saw some websites using a combination of CUBIC and BBR flows to deliver their web pages, we were curious to see how these flows would interact with each other. For these webpages, we ran *Nebby* in its default single bottleneck setting to see how these flows would interact. Interestingly, for AppleTV, we found CUBIC and BBR flows interacting and negatively impacting each other's performance. We often observed that a CUBIC flow delivering a banner ad on `appletv.com` could cause degradation to the long-running BBR flow that was delivering video chunks for the video player. While this particular interaction might be an artifact of *Nebby*'s constraint bandwidth setting, this is enough evidence that developers need to be careful about how they deploy CCAs to avoid causing performance issues inadvertently.

5 DISCUSSION & FUTURE WORK

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

Internet Evolution. The original motivation for this study was the question: *how has the congestion control environment of the Internet changed?* Given the rapid adoption of BBR in 3 years since it was introduced in 2016, the burning question in 2019 was whether BBR would eventually replace CUBIC as the dominant congestion control on the Internet. In 2022, we modeled the interactions between

Table 8: CCAs serving popular web services running on a Selenium client.

Website	Region of popularity	Activity	Connections	Max Concurrent Connections	CCAs for Audio/Video Traffic	CCAs for Static Assets
Netflix	Global	VOD	28	5	**New Reno**+[52]	New Reno, CUBIC
Primevideo	Global	VOD	12	6	BBR	BBR
AppleTV	Global	VOD	16	6	BBR	BBR, CUBIC
Disney+	Global	VOD	20	6	CUBIC	CUBIC
HBO	Global	VOD	10	4	BBR	CUBIC
Tiktok	Global	VOD	21	4	AkamaiCC	AkamaiCC, CUBIC
YouTube	Global	VOD, live video	81	6	**BBRv3**+[32]	**BBR**+[32]
Twitch	Global	VOD, live video	118	6	BBR	CUBIC
Spotify	Global	VOD, streaming audio	8	5	**BBR**+[26]	**BBR**+[26]
Apple Music	Global	streaming audio	16	6	BBR	BBR, AkamaiCC
Zoom	Global	video call	39	6	BBR	CUBIC
Meet	Global	video call	60	5	**BBRv3**+[32]	**BBR**+[32]
Hulu	US	VOD	41	6	AkamaiCC	AkamaiCC
Douyin	China	VOD	5	6	BBR	BBR
Bilibili	China	VOD	10	3	BBR	BBR
Hotstar	India	VOD	12	5	BBR	BBR
Jiocinema	India	VOD	12	6	CUBIC	CUBIC

+ Verified through personal correspondence or public tech blog posts.

CUBIC and BBR and hypothesized that the adoption of BBR would likely slow down because as the proportion of flows switch over to BBR, the advantage of doing so will start to drop [51]. Beyond a critical mass, CUBIC will end up outperforming BBR when it is in the minority. Our latest study suggests that we might have been on to something since the proportion of BBR sites has hardly changed since 2019, despite the rapid initial adoption.

Different Strokes for Different Applications, Different Localities. One of the surprising findings of our latest measurement study is that the preferences for CCAs is not uniform, i.e. providers do not seem to have a preference of one CCA over the rest. In fact, we have found instances where a provider can deploy different CCAs under different contexts. For example, Apple deploys BBR for videos and CUBIC for ads in the same(!) session (see §4.5).

QUIC CCA variants. It has been shown that while QUIC stacks implement standard CCA variants, many of these variants behave somewhat differently from standard kernel implementations [47]. The results of our current study seem to suggest that these non-conformant variants can be classified successfully even though they do not behave exactly like standard kernel implementations. This trend might not continue to hold in the future and the characterization of the current congestion control landscape will be increasingly complex. In this measurement study, we have focused solely on identifying the congestion control variant used and we used our classifier that was trained on the kernel implementations. It is plausible for us to improve the accuracy of our classifier by using the traces from the QUIC implementations as well.

Classifying CCAs beyond those in the Linux kernel. As discussed, *Nebby* can be extended to more CCAs as they are deployed on the Internet. For example, Meta implements Copa [17] in their QUIC stack. As an extension to *Nebby*'s classifier, we wrote a simple Copa classifier that achieved an accuracy of 88% (See Appendix D for more details). When we ran this classifier for our Alexa Top 20k traces, none of the existing websites were identified to be running Copa, including Facebook domains. This is not surprising, since Copa is reported to be deployed by Meta only at the uplink [30]. We implemented a rudimentary classifier for PCC Vivace [25] as well, which had an accuracy of 58% over the Internet (See Appendix D). We did not find any websites running PCC Vivace on the Internet either.

Evolving definition of Fairness. The degree of heterogeneity in the Internet's congestion control landscape as observed by *Nebby* (and Gordon before it) does not arise in a vacuum. This heterogeneity is likely caused by the heterogeneous mix of applications that share the Internet. This is evident from different assets preferring different kinds of CCAs (See §4.5). BBR is often critiqued for not playing fair with CUBIC [37, 51, 61]. However, the very fact they are still coexisting on the Internet means that this critique (which is mostly based on throughput) is not deserved and that our definitions of fairness and deployability need to evolve [60].

6 CONCLUSION

In the early 2000s, measurement studies on the distribution of congestion control algorithms were generally conducted roughly every 10 years. This was sufficient because the evolution of congestion control algorithms was relatively slow. Today, we are in an era of rapid change where new CCAs are developed every year.

With *Nebby*, researchers finally have a reliable and extensible way to keep abreast of the evolution of the Internet congestion control landscape.

Ethics statement: This work does not raise any ethical issues.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers and our shepherd Costin Raiciu for their valuable feedback and helpful comments. This work was supported by the Singapore Ministry of Education grant MOE-T2EP20222-0006 and the NUS-AWS Cloud Credits for Research Grant.

REFERENCES

[1] 2022. Alibaba's QUIC implementation, xquic. (2022). https://github.com/alibaba/xquic

[2] 2022. Amazon Web Services's QUIC implementation, s2n-quic. (2022). https://github.com/aws/s2n-quic

[3] 2022. Cloudflare's QUIC implementation, quiche. (2022). https://github.com/cloudflare/quiche

[4] 2022. Facebook's QUIC implementation, mvfst. (2022). https://github.com/facebookincubator/mvfst

[5] 2022. Google's QUIC implementation, chromium. (2022). https://www.chromium.org/quic/playing-with-quic

[6] 2022. Go's QUIC implementation, quic-go. (2022). https://github.com/lucas-clemente/quic-go

[7] 2022. H2O's QUIC implementation, quicly. (2022). https://github.com/h2o/quicly

[8] 2022. LiteSpeed's QUIC implementation, lsquic. (2022). https://github.com/litespeedtech/lsquic

[9] 2022. Microsoft's QUIC implementation, msquic. (2022). https://github.com/microsoft/msquic

[10] 2022. Mozilla's QUIC implementation, neqo. (2022). https://github.com/mozilla/neqo

[11] 2022. Rust's QUIC implementation, quinn. (2022). https://github.com/quinn-rs/quinn

[12] 2023. Alexa Top Websites - Last Save. (2023). https://www.expireddomains.net/alexa-top-websites/

[13] 2023. BBRv3: Algorithm Bug Fixes and Public Internet Deployment. (2023). http://tinyurl.com/bbrv3ietf

[14] 2024. Nebby. (2024). www.github.com/NUS-SNL/Nebby

[15] 2024. Selenium. (2024). https://www.selenium.dev

[16] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. 2004. Sizing router buffers. *ACM SIGCOMM CCR* 34, 4 (2004).

[17] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical Delay-Based Congestion Control for the Internet. In *Proceedings of NSDI*.

[18] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. 2007. YeAH-TCP: yet another highspeed TCP. In *Proceedings of PFLDnet*.

[19] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. 1994. TCP Vegas: New techniques for congestion detection and avoidance. In *Proceedings of SIGCOMM*.

[20] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: Congestion-based Congestion Control. *CACM* 60, 2 (2017), 58–66.

[21] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR Congestion Control. IETF Draft. (2017). https://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-00

[22] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. 2019. BBR v2 - A Model-based Congestion Control. ICCRG at IETF 104. (2019). https://bit.ly/2HgGOuQ

[23] Claudio Casetti, Mario Gerla, Saverio Mascolo, Medy Y Sanadidi, and Ren Wang. 2002. TCP Westwood: end-to-end congestion control for wired/wireless networks. *Wireless Networks* 8, 5 (2002), 467–479.

[24] Xiaoyu Chen, Shugong Xu, Xudong Chen, Shan Cao, Shunqing Zhang, and Yanzan Sun. 2019. Passive TCP identification for wired and wireless networks: A long-short term memory approach. In *Proceedings of IWCMC*.

[25] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC Vivace: Online-Learning Congestion Control. In *Proceedings of NSDI*.

[26] Eirini Kakogianni Erik Carlsson. 2018. Smoother Streaming with BBR. (2018). https://engineering.atspotify.com/2018/08/smoother-streaming-with-bbr/

[27] Margarida Ferreira, Akshay Narayan, Inês Lynce, Ruben Martins, and Justine Sherry. 2021. Counterfeiting Congestion Control Algorithms. In *Proceedings of Hotnets*.

[28] Sally Floyd. 2003. HighSpeed TCP for Large Congestion Windows. RFC 3649. (2003).

[29] Cheng Peng Fu and S. C. Liew. 2006. TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks. *IEEE JSAC* 21, 2 (2006), 216–228.

[30] Nitin Garg. 2019. Engineering at Meta: Evaluating COPA congestion control for improved video performance. (2019). https://engineering.fb.com/2019/11/17/video-engineering/copa/

[31] Sishuai Gong, Usama Naseer, and Theophilus A Benson. 2020. Inspector Gadget: A Framework for Inferring TCP Congestion Control Algorithms and Protocol Configurations.. In *Network Traffic Measurement and Analysis Conference*.

[32] Google. 2023. Personal Correspondance. (2023).

[33] Google Cloud Blogs 2017. TCP BBR congestion control comes to GCP: your Internet just got faster. (2017). https://bit.ly/2Hk4WLH

[34] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Operating Systems Review* 42, 5 (2008), 64–74.

[35] Desta Haileselassie Hagos, Paal E Engelstad, Anis Yazidi, and Øivind Kure. 2018. General TCP state inference model from passive measurements using machine learning techniques. *IEEE Access* 6 (2018), 28372–28387.

[36] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. 2012. The NewReno Modification to TCP's Fast Recovery Algorithm. (2012). https://tools.ietf.org/html/rfc6582

[37] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental Evaluation of BBR Congestion Control. In *Proceedings of ICNP*.

[38] Alexa Internet Inc. 2023. The Top 500 websites on the Internet. (2023). https://www.alexa.com/topsites

[39] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-Based Multiplexed and Secure Transport, RFC 9000. (2021). https://datatracker.ietf.org/doc/html/rfc9000

[40] Van Jacobson. 1988. Congestion Avoidance and Control. *ACM SIGCOMM Computer Communication Review* 18, 4 (1988).

[41] Sharad Jaiswal, Gianluca Iannaccone, Christophe Diot, Jim Kurose, and Don Towsley. 2004. Inferring TCP connection characteristics through passive measurements. In *Proceedings of INFOCOM*.

[42] Matt Joras and Yang Chi. 2020. How Facebook is bringing QUIC to billions. (2020). https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/

[43] Tom Kelly. 2003. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *SIGCOMM CCR* 33, 2 (2003), 83–91.

[44] Douglas Leith, R Shorten, and Y Lee. 2005. H-TCP: A framework for congestion control in high-speed and long-distance networks. In *Proceedings of PFLDnet*.

[45] Shao Liu, Tamer Başar, and R. Srikant. 2006. TCP-Illinois: A Loss and Delay-based Congestion Control Algorithm for High-speed Networks. In *Proceedings of VALUETOOLS*.

[46] Alberto Medina, Mark Allman, and Sally Floyd. 2005. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM CCR* 35, 2 (2005), 37–52.

[47] Ayush Mishra and Ben Leong. 2023. Containing the Cambrian Explosion in QUIC Congestion Control. In *Proceedings of IMC*.

[48] Ayush Mishra, Sherman Lim, and Ben Leong. 2022. Understanding Speciation in QUIC Congestion Control. In *Proceedings of IMC*.

[49] Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. 2019. Gordon: Congestion Control Identification Tool. (2019). https://github.com/NUS-SNL/Gordon

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

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

[52] NetFlix. 2023. Personal Correspondance. (2023).

[53] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *Proceedings of ATC*.

[54] Jitendra Padhye and Sally Floyd. 2001. On Inferring TCP Behavior. In *Proceedings of SIGCOMM*.

[55] Vern Paxson and Mark Allman. 2009. TCP Congestion Control. RFC 5681. (2009).

[56] Canada Sandvine Inc. Waterloo, ON. 2022. The 2022 Global Internet Phenomena Report. (2022). https://www.sandvine.com/phenomena

[57] Satadal Sengupta, Hyojoon Kim, and Jennifer Rexford. 2022. Continuous in-network round-trip time monitoring. In *Proceedings of SIGCOMM*.

[58] Bruce Spang, Serhat Arslan, and Nick McKeown. 2022. Updating the theory of buffer sizing. In *Proceedings of SIGMETRICS*.

[59] Kun Tan, Jingmin Song, Qian Zhang, and Murad Sridharan. 2006. A compound TCP approach for high-speed and long distance networks. In *Proceedings of INFOCOM*.

[60] Ranysha Ware, Matthew K Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Beyond jain's fairness index: Setting the bar for the deployment of congestion control algorithms. In *Proceedings of Hotnets*.

[61] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Modeling BBR's Interactions with Loss-Based Congestion Control. In *Proceedings of IMC*.

[62] Lisong Xu, K. Harfoush, and Injong Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *Proceedings of INFOCOM*.

[63] Peng Yang, Juan Shao, Wen Luo, Lisong Xu, Jitendra Deogun, and Ying Lu. 2011. TCP Congestion Avoidance Algorithm Identification. *IEEE/ACM Transactions on Networking* 22, 4 (2011), 1311–1324.

Table 9: Distribution of TCP variants with *Gordon* [49].

TCP variant	Websites	Proportion	2019 [50]
CUBIC [34]	212	2.12%	30.7%
BBRv1 [20]	85	0.85%	17.75%
CTCP [59]/Illinois[45]	63	0.63%	5.74%
Reno [55]/HSTCP [28]	52	0.52%	0.80%
Other CCAs	0	0%	18.87%
Unknown	1,430	14.30%	12.16%
Short flows	6,282	62.82%	7.47%
Unresponsive	1,876	18.76%	6.51%
Total	10,000	100%	100%

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

A REPLICATING GORDON

Among previous measurement studies, our earlier study in 2019 was the most recent work that attempted classifying congestion control algorithms deployed by web servers on the Internet [50]. Gordon estimates a sender's congestion window by counting the number of unacknowledged packets in each RTT. To do so, Gordon makes a connection and drops packets till it sees a retransmission. Gordon repeats this process over hundreds of connections and then uses the resulting `cwnd` traces to identify the CCA. We expected the Internet to have evolved since Gordon's last measurement study in 2019, so we decided to use Gordon to classify the Alexa Top 10,000 websites. Unfortunately, we were sorely disappointed to discover that Gordon was only able to successfully identify 4% of our measured websites, and among these websites, only 4 categories of TCP variants were identified. We summarize the results in Table 9. We also reproduce the results from [50] (2019) in the last column for easy reference.

We see a sharp increase in unclassified and unknown websites. 63% of failures were due to the flows being too short, despite us crawling these websites for large web pages. We found that these websites often did not serve the requested page, but an error page instead, because Gordon's measurement was treated as a DoS attack. This is not surprising, since Gordon's methodology is extremely aggressive and was likely inferred as malicious. In summary, it is not practical to use Gordon to classify CCAs run by websites today. Moreover, Gordon only focuses on TCP connections and cannot classify flows serving web browsers and other real-world applications.

B POLYNOMIAL FITTING FOR LOSS-BASED CONGESTION CONTROL ALGORITHMS

As mentioned in §3.4, *Nebby*'s classifier classifies loss-based CCAs by matching the shape of a candidate measurement with that of known loss-based CCAs in the Linux kernel. To do so, we express each segment of a measurement trace as a polynomial and check if the coefficients of the polynomials match with that of the known loss-based CCAs.

We tested all the coefficients generated by our control tests using the D'Agostino K2 test and the Shapiro-Wilk test to verify if these coefficients were normally distributed for all of the target CCAs. We used a soft-fail hypothesis, where if a CCA's coefficients passed either of the tests, they were considered to be normally distributed. All our target CCAs passed at least one of the two tests. This allows us to model each CCA's coefficients as a normal distribution, and treat each CCA's polynomial as a multivariate random variable.

Since we can model each feature's polynomial as a multivariate random variable, it opens up the possibility of using a large variety of supervised learning algorithms. We chose to use Gaussian Naive Bayes (GNB) because it works with continuous data and does not require mapping the data to a higher dimensional feature space (as is the case for Support Vector Machines). We were reluctant to use such methods as it would mean that some new feature other than the polynomial coefficients which reflect the shape of the CCAs would be used to make the classification. As stated earlier, we only want to use the shape of the CCAs to classify measurements to avoid the risk of overfitting. GNB gives us this freedom. The prior for the GNB was set to be a discrete distribution which allows each CCA to be chosen with equal probability.

The GNB classifier gives us a probability that a given feature can belong to a certain CCA. In practice, we found the probabilities for most of our features to be skewed to only one CCA. In cases where multiple CCAs have equally high probabilities, we classify the feature as Unknown. In general, a measurement can have multiple features. In such cases, a measurement is classified as a known CCA only if all features in that measurement belong to that CCA. A measurement can also be classified as a known CCA if only some of the features belong to that known CCA while the other features are classified as unknown. We found no instances where features from the same measurements matched two different CCAs.

Table 10: List of open-source QUIC stacks studied.

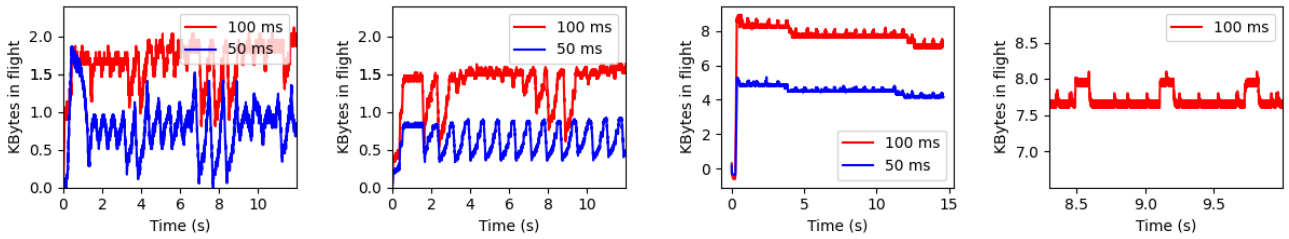
Organization	Stack	CUBIC	BBR	Reno
Alibaba	`xquic` [1]	✓	✓	✓
Amazon Web Services	`s2n-quic` [2]	✓	✗	✗
Cloudflare	`quiche` [3]	✓	✗	✓
Go	`quicgo` [6]	✓	✗	✓
Google	`chromium` [5]	✓	✓	✗
H2O	`quicly` [7]	✓	✗	✓
LiteSpeed	`lsquic` [8]	✓	✓	✗
Meta	`mvfst` [4]	✓	✓	✓
Microsoft	`msquic` [9]	✓	✗	✗
Mozilla	`neqo` [10]	✓	✗	✓
Rust	`quinn` [11]	✓	✗	✓

C QUIC STACKS CHOSEN

In Table 10, we summarize the QUIC stacks we chose to benchmark *Nebby*'s classifier with. These 11 QUIC stacks were chosen based on the criteria that they were all publicly available, open source, deployed, and implemented some congestion control algorithm.

D CLASSIFIERS FOR COPA AND PCC VIVACE

As discussed in §5, we extended *Nebby*'s classifier to include a classifier for Copa [17]. This classifier identified Copa based on its periodic oscillations around the bottleneck bandwidth that occur every five RTTs (see Figure 11). Our classifier was able to successfully classify both the original UDP implementation of Copa [17]



(a) Copa [17] **(b) `mvfst` Copa [4]** **(c) PCC Vivace [25]** **(d) Zoomed in view for a PCC trace.**

Figure 11: `BiF` traces for Copa and PCC Vivace.

Table 11: Evolution of the Internet's Congestion Control Landscape (2001–present).

Class	CCA	2001 [54]	2004 [41]	2011 [63]	2019 [50]	2023, *Nebby*
Loss-based AIMD	New Reno [55]	35% (1,571)	25% (21,266)		0.8% (160)	11.1% (11,157)
	Reno [40]	21% (945)	5% (4,115)	12.5% (623)	-	-
	Tahoe	26% (1,211)	3% (2,164)		-	-
Loss-based MIMD	CUBIC [34]	-	-	22.3% (1,115)	30.7% (6,139)	41% (41,085)
	BIC [62]			10.6% (531)	0.9% (181)	3% (2,985)
	HSTCP [28]			7.4% (369)	R	0% (0)
	Scalable [43]			1.4% (69)	0.2% (39)	0% (24)
Delay-based AIMD	Vegas [19]	-	-	1.2% (58)	2.8% (564)	2.6% (2,637)
	Westwood [23]			2% (104)	0% (0)	0.4% (371)
Delay-based MIMD	CTCP [59]	-	-	6.7% (334)	5.7% (1,148)	C
	Illinois [45]			0.6% (28)		3.4% (3,380)
	Veno [29]			0.9% (45)	V	0.6% (578)
	YeAH [18]			1.4% (72)	5.8% (1,162)	0.4% (388)
	HTCP [44]			0.4% (18)	2.8% (560)	2.3% (2,259)
Rate-based	BBRv1 [21]	-	-	-	17.8% (3,550)	10% (9,985)
	BBR G1.1 [50]				0.8% (167)	-
	BBRv2 [22]				-	1.1% (1,151)
	BBRv3				-	0.2% (204)
Unclassified		17.3% (792)	53% (44,950)	4% (198)	12.2% (2,432)	16.7% (16,733)
AkamaiCC		-	-	-	5.5% (1,103)	7.2% (7,117)
Short Flows		-	-	26% (1,300)	7.5% (1,493)	-
Unresponsive		0.7% (30)	14% (11,529)	-	6.5% (1,302)	-
Abnormal SS*		-	-	2.9% (144)	-	-
Total hosts		100% (4,550)	100% (84,394)	100% (5,000)	100% (10,000)	100% (100,000)

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

as well as `mvfst`'s implementation [4] with an accuracy of 88%. Interestingly, `mvfst` Copa's oscillations were less visible at higher RTTs, even though in theory *Nebby* should be able to view a larger portion of Copa's `BiF`.

We also wrote a classifier for PCC Vivace [25], which proved to be more challenging. Vivace is an online optimization algorithm that periodically probes above and below the receive rate to see if it can improve its utility. These probes are relatively small, but we can see in the 100 ms delay profile shown in Figure 11d that *Nebby* is able to observe the resulting oscillations. However, our classifier could only identify these *steps* in the `BiF` only about half the time, perhaps because the variations in the `BiF` are relatively small. As a result, our PCC Vivace classifier's accuracy suffered and was about 58%. We believe that with further study, it is certainly possible to develop a more accurate classifier for PCC Vivace and we leave this as future work.

E SUMMARY OF INTERNET CCA EVOLUTION (2001–PRESENT)

In Table 11, we add the results for our latest measurement study to the data from our earlier measurement study in 2019 [50]. We note that since some websites deploy different CCAs in different regions, we sum the results from all the regions to compare our latest results to previous studies which were not region-specific. Because we did not have access to the implementation of BBRv3, we were not able to tune our BBR classifier for BBRv3. Nevertheless, it is clear that while the CCA deployed by Google websites were similar to BBR, they were not BBRv1 or BBRv2. We inferred that they must be BBRv3 (0.2%).