# The Great Internet TCP Congestion Control Census

AYUSH MISHRA, National University of Singapore, Singapore
XIANGPENG SUN, National University of Singapore, Singapore
ATISHYA JAIN, Indian Institute of Technology, Delhi, India
SAMEER PANDE, Indian Institute of Technology, Delhi, India
RAJ JOSHI, National University of Singapore, Singapore
BEN LEONG, National University of Singapore, Singapore

In 2016, Google proposed and deployed a new TCP variant called BBR. BBR represents a major departure from traditional congestion-window-based congestion control. Instead of using loss as a congestion signal, BBR uses estimates of the bandwidth and round-trip delays to regulate its sending rate. The last major study on the distribution of TCP variants on the Internet was done in 2011, so it is timely to conduct a new census given the recent developments around BBR. To this end, we designed and implemented *Gordon*, a tool that allows us to measure the exact congestion window (`cwnd`) corresponding to each successive RTT in the TCP connection response of a congestion control algorithm. To compare a measured flow to the known variants, we created a localized bottleneck where we can introduce a variety of network changes like loss events, bandwidth change, and increased delay, and normalize all measurements by RTT. An offline classifier is used to identify the TCP variant based on the `cwnd` trace over time.

Our results suggest that CUBIC is currently the dominant TCP variant on the Internet, and it is deployed on about 36% of the websites in the Alexa Top 20,000 list. While BBR and its variant BBR G1.1 are currently in second place with a 22% share by website count, their present share of total Internet traffic volume is estimated to be larger than 40%. We also found that Akamai has deployed a unique loss-agnostic rate-based TCP variant on some 6% of the Alexa Top 20,000 websites and there are likely other undocumented variants. The traditional assumption that TCP variants "in the wild" will come from a small known set is not likely to be true anymore. We predict that some variant of BBR seems poised to replace CUBIC as the next dominant TCP variant on the Internet.

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

Keywords: congestion control; measurement study

**ACM Reference Format:**
Ayush Mishra, Xiangpeng Sun, Atishya Jain, Sameer Pande, Raj Joshi, and Ben Leong. 2019. The Great Internet TCP Congestion Control Census. In *PREPRINT: Proc. ACM Meas. Anal. Comput. Syst.*, Vol. 3, 3, Article 45 (December 2019). ACM, New York, NY. 24 pages. https://doi.org/10.1145/3366693

Authors' addresses: Ayush Mishra, ayush@comp.nus.edu.sg, National University of Singapore, Singapore; Xiangpeng Sun, sun.xiangpeng@comp.nus.edu.sg, National University of Singapore, Singapore; Atishya Jain, atishya.jain.cs516@cse.iitd.ac.in, Indian Institute of Technology, Delhi, India; Sameer Pande, sameer.vivek.pande.cs117@cse.iitd.ac.in, Indian Institute of Technology, Delhi, India; Raj Joshi, rajjoshi@comp.nus.edu.sg, National University of Singapore, Singapore; Ben Leong, benleong@comp.nus.edu.sg, National University of Singapore, Singapore.


2476-1249/2019/12-ART45 $15.00
https://doi.org/10.1145/3366693

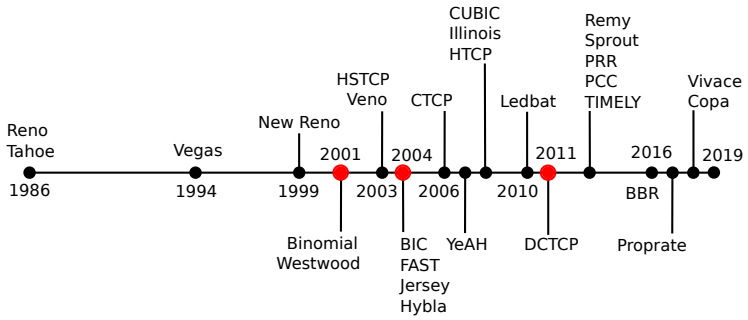



Fig. 1. The evolution of TCP congestion control.

## 1 INTRODUCTION

Over the past 30 years, TCP congestion control has evolved to adapt to the changing needs of the users and to exploit improvements in the underlying network. Most recently, in 2016, Google proposed and deployed a new TCP variant called BBR [4] (**B**ottleneck **B**andwidth and **R**ound-trip propagation time). BBR represents a major departure from traditional congestion-window-based congestion control. Instead of using packet loss as a congestion signal, BBR uses estimates of the bandwidth and round-trip delays to regulate its sending rate. BBR has since been introduced in the Linux kernel and deployed by Google across its data centers. We summarize the evolution of TCP congestion control in Fig. 1 (with previous studies of TCP distributions indicated in red [24, 27, 38]). As the TCP ecosystem has changed significantly since the last study [39] done in 2011, it is timely to conduct a new census to understand the latest distribution of TCP variants on the Internet.

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

To this end, we designed *Gordon*, a tool that allows us to measure the exact congestion window (cwnd) corresponding to each successive RTT in the TCP connection response of a congestion control algorithm "in the wild." While rate-based TCP variants do not maintain a congestion window, they typically maintain a maximum allowable number of packets in flight [4], which we can measure as the *effective* congestion window for each RTT. To compare this response to that of known variants, we created a localized bottleneck where we introduced a variety of network changes: loss events, bandwidth change, and increased delay. We also normalize all measurements by RTT. An offline classifier is then used to identify the TCP variant based on the cwnd trace over time. By decoupling measurement from classification unlike prior studies [24, 27, 38], our approach allows us to not only identify known TCP variants but also detect new undocumented variants. Our approach also makes it possible to improve the accuracy of the classifier without repeating the relatively expensive measurements, if new network profiles are not required for the improved classifier.

We used Gordon to measure the 20,000 most popular websites according to the Alexa rankings [18]. The following are our key findings:

(1) Our results suggest that, as expected, CUBIC is currently the dominant TCP variant on the Internet and is deployed at 36% of all the classified websites, which is an increase from what was reported in the last study in 2011 (§4.5).
(2) The rate of BBR adoption over the past 3 years since its release has been phenomenal. BBR (together with its Google variant) is currently the second most popular TCP variant deployed at 22% of the classified websites (§4.5).
(3) While BBR has a share of only 22% by website count, we estimate that its present share of total Internet traffic volume already exceeds 40%. This proportion will almost certainly exceed 50% if Netflix and Akamai also decide to adopt BBR (§4.3).
(4) The assumption that TCP variants "in the wild" will come from a known set is not true anymore. In particular, we found that Akamai has deployed a unique loss-agnostic rate-based TCP variant on some 6% of the Alexa Top 20,000 websites (§4.4).

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

The rest of the paper is organized as follows: in §2, we provide an overview of previous attempts to characterize congestion control variants deployed in the wild. In §3, we describe the design and implementation of Gordon's measurement and classification techniques. In §4, we first evaluate the measurement accuracy of Gordon and then present detailed results of using Gordon to identify TCP variants for the Alexa Top 20,000 websites [18] on the Internet. In §5, we describe the practical difficulties we faced, the current limitations of Gordon, and future directions to improve Gordon for understanding the long-term evolution of Internet congestion control. Finally, we conclude in §6.

## 2 RELATED WORK

To the best of our knowledge, there have been four prior studies attempting to characterize TCP congestion control variants deployed in the wild. In 2001, Padhye et al. [27] used a tool called TBIT that performed a specialized 25-packet drop and accept pattern which allowed it to detect if a web server was running one of the four target congestion control variants: Reno, New Reno, Reno Plus and Tahoe. At the time of publication, the consensus was that Reno was the most widely deployed variant. However, their results showed that most of the Internet was already using New Reno.

In 2004, Medina et al. [24] followed up on the work by Padhye et al. by using TBIT to conduct active and passive measurements of over 84,000 hosts on the Internet. While they were only able to classify 33% of their target hosts, the categorized hosts showed a continued trend of moving from Reno to New Reno, as observed earlier by Padhye et al. [27].

A study by Yang et al. [38] in 2011 provides the most recent update on the distribution of congestion control variants on the Internet. In this work, they classify TCP variants on the Internet using cwnd traces collected via two distinct network profiles. Their tool, CAAI, extracts feature vectors from these cwnd measurements and identifies them via a classifier trained on cwnd traces from controlled servers in a local testbed. While both CAAI and Gordon make cwnd measurements to identify congestion control variants on the Internet, they do so in very different ways. CAAI uses delayed ACKs to bloat the RTT in an attempt to 'space out' the individual cwnds in a connection.

This approach would not work while measuring rate-based variants, which is one of the main motivations for our work. Rate-based variants like BBR will continue to send packets that fill the entire network pipeline and render CAAI's delayed ACK measurement technique untenable. Their measurements showed that BIC, CUBIC, and Compound TCP (CTCP) together had become more popular than New Reno. Separately, Yang et al. also identified delay-based variants like YeAH [2], Vegas [3], Veno [13] and Illinois [22] [39]. They found that about 4% of the Internet hosts tested were using these delay-based congestion control variants. While we too aim to measure the general distribution of congestion control protocols, we are more focused on studying the adoption of more recent rate-based variants, like BBR. We summarize the key findings of our work together with these previous studies in Table 9 (§4.5).

In terms of our measurement methodology, unlike prior tools [27, 38] that attempt to directly classify the variants, Gordon decouples measurement and classification by design. In other words, the classifier can essentially be swapped with other classifiers that work with our cwnd traces. Instead of attempting to classify a TCP variant among a set of known TCP variants, we capture its response to a fixed trace of varying network conditions to determine the entire evolution of a TCP sender's cwnd over time and normalize the result by RTT. Our approach allows us to identify and make useful observations about undocumented variants (see §4.4). Our approach also makes Gordon easily extensible as we leverage these observations to design new measurement and classification methods to account for the new variants discovered in the wild. Like CAAI [38], Gordon also emulates a controlled network environment between a measurement server and the web servers on the Internet. However, CAAI emulates only changes in RTT and packet loss, while Gordon extends the emulation to changes in bandwidth. Gordon differs from CAAI in the way that classification is done. Gordon applies a decision tree to collected cwnd trace for a website, while CAAI collects a set of reference traces under a range of controlled network conditions and compares the trace of a probed website to these reference traces to find the closest match.

Chen et al. used deep neural networks to analyze passive measurements taken from TCP receivers and identify the congestion control variant used by a TCP sender [8]. They used traces of long continuous flows to train a Long Short Term Memory (LSTM) neural network that classifies the trace behaviors into the congestion control variants by using features such as RTT, packets in flight and throughput. Their evaluation was done only in a controlled testbed and so it is not surprising that neural networks can classify relatively well-behaved traces. Because evaluation was not done on actual Internet hosts, no attempts were made at addressing the noise from packet losses on the Internet. We have reason to believe that such noise would introduce significant errors. A key insight that makes Gordon work is our simple but effective technique to eliminate noisy traces from random packet losses (see §3.1). Also, while supervised learning approaches can identify known TCP variants, they will not be able to uncover new undocumented variants that are surprisingly common (see §4.4).

There have also been some works on TCP-related measurements that focus on evaluating congestion control algorithms and their implementations. Hagos et al. used machine learning to infer the state of a TCP sender [16]. Comer et al. used active probing techniques to reveal implementation flaws, protocol violations and design decisions of the 5 commercial black box congestion control implementations [10]. Sun et al. [33] and Lubben et al. [23] also evaluated the correctness of TCP implementations in controlled testing environments. None of these are directly applicable for identifying TCP variants on the Internet.

## 3 METHODOLOGY

Gordon emulates a local bottleneck and tracks the evolution of the effective congestion window (cwnd) (see §3.1) of the probed TCP variant while changing the available bandwidth, increasing

the delay and introducing packet losses in a controlled manner (see §3.2). In the case of rate-based protocols that do not use a cwnd for rate regulation, we track the unacknowledged packets in flight as the cwnd of the protocol. The key insight behind our design is that any congestion control protocol must ultimately react to changing networking conditions. We then try to identify the TCP variant from the observed cwnd response via offline processing (see §3.3).

Gordon targets identifying congestion control variants that have been deployed in the Windows and Linux kernels. However, since it operates as an interceptor, it is not limited to measuring only TCP behavior, and can be used to measure UDP traffic as well. In this work, we concentrate on making measurements on TCP web traffic since TCP supports an overwhelmingly large proportion of Internet traffic [32].

### 3.1 Measuring cwnd over time

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

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

**MTU sizing and crawling for large web-pages.** Since we measure cwnd in packets, a straightforward way to obtain more packets from an HTTP/HTTPS page download is to reduce the MTU size of the connection. IPv4 [11] specifies a minimum MTU size of 68 bytes. However, we found that setting an MTU size of 68 bytes often resulted in some connections failing without reason. Through repeated trials for all the websites in the Alexa Top 20,000 list, we found that while an MTU of 68 bytes works for most websites, some accept only connections with larger MTU sizes. To address this issue, Gordon uses binary search to determine the minimum MTU size for a website and performs the measurement using this MTU size. This acceptable MTU size search is done before every measurement since the minimum acceptable MTU size could vary depending on the underlying Internet path, which could change over time.

However, reducing the MTU size was often not enough to yield a sufficiently long trace to identify the TCP variant. Thus, we first used a crawler to determine the available pages for each website (to the best of our ability) and used the largest of these pages to perform our measurements. Using our final network profile (see §3.2), Gordon needs about 80 packets for 30 RTTs to be able to accurately plot cwnd evolution graphs for more complicated algorithms like CUBIC. With most

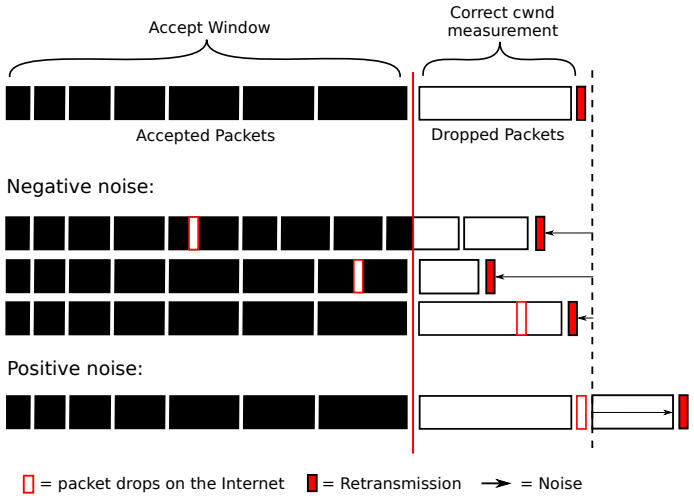



Fig. 2. Possible scenarios for random losses.

websites accepting 68 -byte MTUs, this would mean an ideal web-page size for Gordon would be at least 165 KB.

**Handling Random Packet Losses.** In Fig. 2, we present the various scenarios when we observe packet loss in our measurements. We note that most packet losses result in a lower estimate (*negative* noise). It is only when the first re-transmitted packet is lost that we end up counting the entire re-transmitted window twice and have *positive* noise. The latter is easily eliminated if we stop counting packets when we see the re-transmission of any packet in the current cwnd measurement window.

We eliminate negative noise caused by random losses by repeating the measurement for each congestion window several times and taking the maximum window measurement as the cwnd. In Fig. 3, we plot the measurement noise from random losses while measuring various web-servers on the Internet (both real hosts on the Internet and controlled servers set up on AWS) for different number of trials. We see that 15 trials per cwnd measurement are sufficient to eliminate negative noise. Here, by 'noise' we mean the cumulative sum of the difference between the measured and ground truth cwnd values. In this experiment, the ground truth was taken to be the measurements made over 50 trials. In addition to this, all our experiments were done over wired links to minimize the possibility of random packet losses.

In Fig. 4, we plot the window measurements for `reddit.com` using 15 trials per cwnd measurement. The red points are the individual window measurements. We see that taking the maximum over 15 trials per window measurement are sufficient to provide us with a relatively smooth cwnd evolution curve. The small cwnd during the first 5 RTTs is the result of the SSL certificate exchange protocol.

### 3.2 Designing a Network Profile

Our goal is to identify TCP variants from the evolution of their cwnd over time. Conceptually, we described a way to do this measurement in §3.1. However, we need a way to normalize the measurements so that they can be compared to base measurements of known TCP variants. Since we have full control over the network bottleneck, we can impose a common network profile on all the websites. In particular, we introduce a packet loss event and a bandwidth change event at the network bottleneck and observe the response of the probed TCP algorithm.

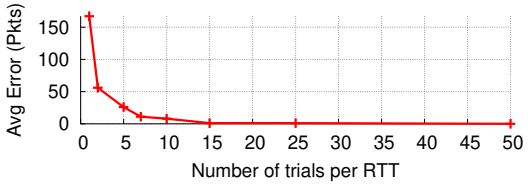



Fig. 3. Sensitivity analysis for repeated measurements.

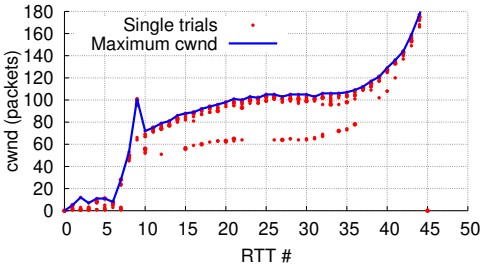



Fig. 4. cwnd measurement for `reddit.com`.

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

Through measurements, we found that most connections have a starting window size of 10 packets, as suggested by Chu et al. [19, 31]. This means that for a typical Slow Start, we can expect the first few congestion windows to be 10, 20, 40, 80, etc. In Fig. 5, we plot the evolution of cwnd for a controlled web server running CUBIC while Gordon emulates a drop at different stages of a connection - namely when the measured cwnd first reaches more than 40, 80 and 160 packets. We evaluate CUBIC since it has relatively complex cwnd evolution in the Congestion Avoidance phase. Fig. 5 shows that if the packet drop occurs too early, the subsequent cwnd is relatively small and it might be hard to discern between the curve shapes after the packet drop. On the other hand, if the packet drop is too late, the window size becomes very large and we need very large flows (large web pages) to make a measurement that captures the entire CUBIC curve. We found that inflicting a packet loss after the cwnd reaches 80 packets achieves a good trade-off between these two concerns. We call this value the *Packet Drop Threshold*. Except for this inflicted packet drop meant to "force" cwnd-based TCP variants into *Congestion Avoidance* phase, no other packets are explicitly dropped by Gordon during the measurement. Our buffer is big enough to avoid buffer overflows.

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

Since our methodology requires us to limit the sender's cwnd to about 100 packets to make the flows last long enough, we emulate a BDP of 50 packets. We achieve this BDP by maintaining an RTT of 100 ms between the sender and the receiver and limiting the initial bottleneck bandwidth to 500 packets/s for the first 1,500 packets received. This rate is reduced to 334 packets/s for the next 1,500 packets before the bandwidth is restored to 500 packets/s. This behavior can be seen in Fig. 6, where we show the available bandwidth in terms of the BDP for the flow (since the delay is a constant). We can see that the cwnd for a controlled web server running BBR tracks the available bandwidth at twice the BDP emulated by Gordon after a measurement delay of 10 RTTs. We decided on changing the BDP every 1,500 packets because it would result in a period of 15 to 20 RTTs and works for the general file sizes in our sampled websites. This change in bandwidth also allows us

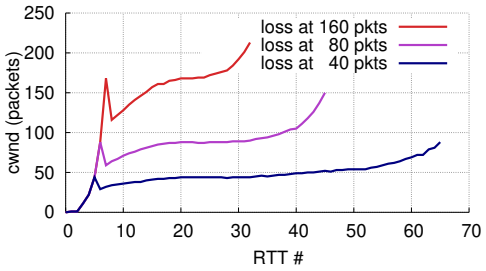



Fig. 5. Evolution of CUBIC cwnd for different packet drops.

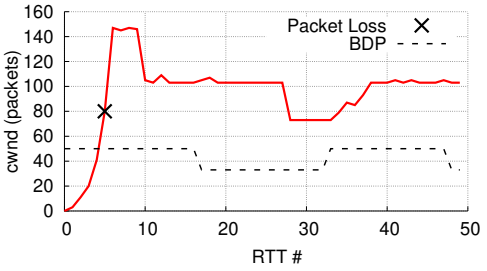



Fig. 6. How BBR reacts to the bandwidth changes.

to identify other rate-based variants that may react to a change in bottleneck bandwidth but track the emulated BDP differently.

**Final Network Profile.** In summary, we inflict a packet drop for the first window where the number of packets received is strictly larger than 80. The available bandwidth of the bottleneck alternates between 500 packets/s and 334 packets/s after every 1,500 packets received. In Fig. 7, we plot the responses for some common congestion control algorithms as measured by Gordon while applying the final network profile. We note that except three pairs of congestion control algorithms (Veno/Vegas, New Reno/HSTCP and CTCP/Illinois) we are generally able to identify the TCP variant from the shape of the curve within the first 30 RTTs. These shapes are deterministic and Gordon is consistently able to record traces like the ones in Fig. 7 over multiple runs. These shapes show slight deviations when measured over the Internet, and their impact on our classification accuracy is discussed in § 4.1.

In the future, if there are deployments of other congestion control variants, additional network profiles can easily be added to Gordon to identify them. In this work, we limit ourselves to using a single network profile because of the cost associated with measuring each website.

### 3.3 Classification

The output from Gordon is a plot of estimated cwnd versus time (RTT #) of the target host in response to our final network profile. It remains for us to determine the TCP variant from the shape of the graphs. For measurements that are sufficiently long and yield enough data, we expect the shapes to be similar to those shown in Fig. 7.

We use a simple decision-tree-based approach to identifying variants over the Internet (see §4.1). One of the benefits of our approach of decoupling measurement and classification is that other researchers are free to swap our classifier with a different classifier. We have made the source code for Gordon and our measurement traces publicly available (§8).

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

**Case 1: Back-off After Packet Loss**. We divide the resulting Congestion Avoidance phase into 3 regions (as shown in Fig. 8).

(1) Catch-Up: This region corresponds to the region right after the algorithm backs off to a lower cwnd after encountering a packet loss.
(2) Steady: This is the region where cwnd demonstrates linear or no growth.

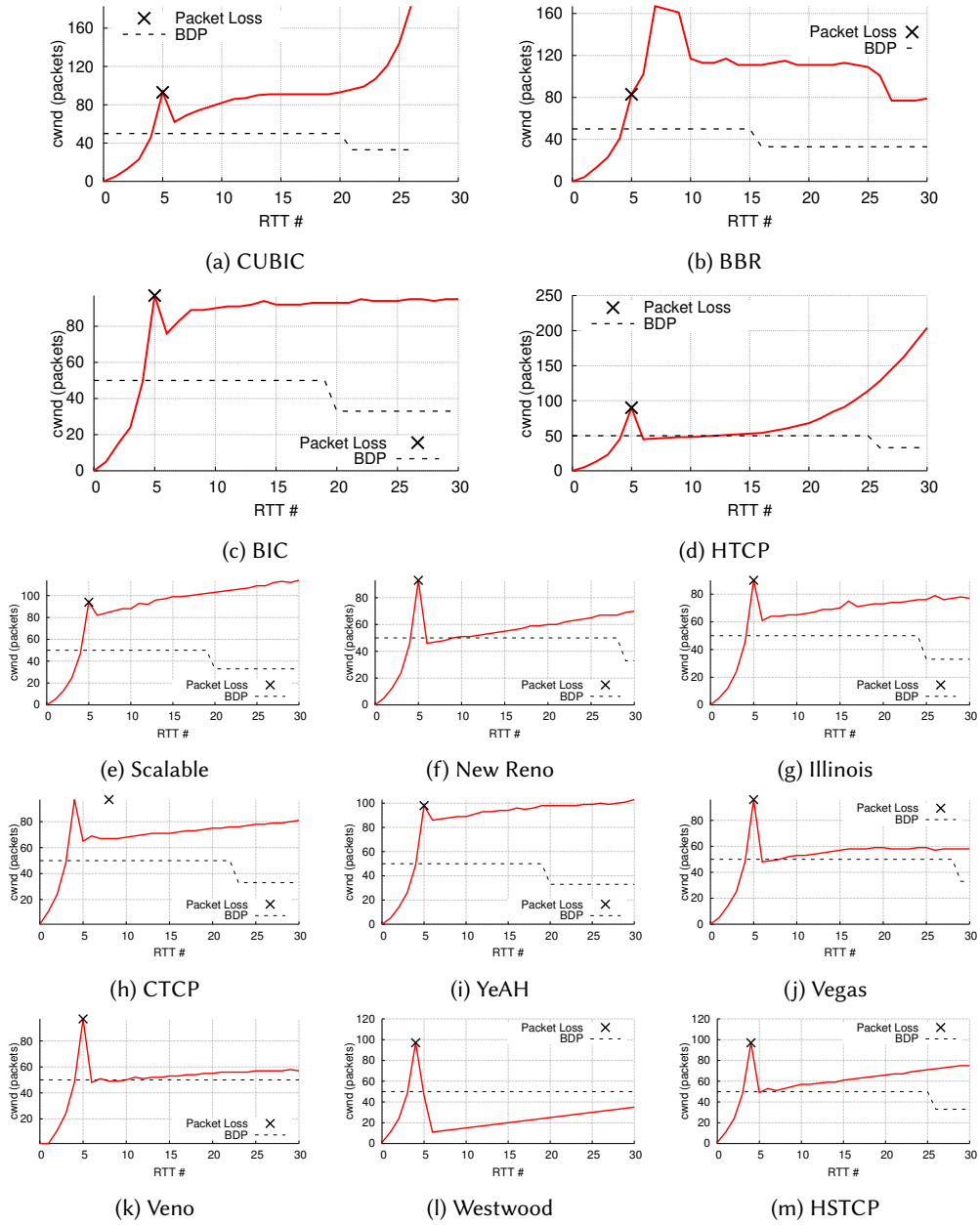



Fig. 7. Curves for TCP congestion control algorithms in response to our final network profile.

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

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

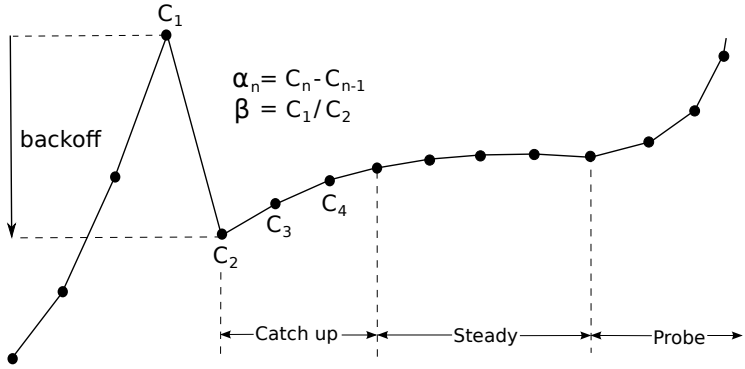
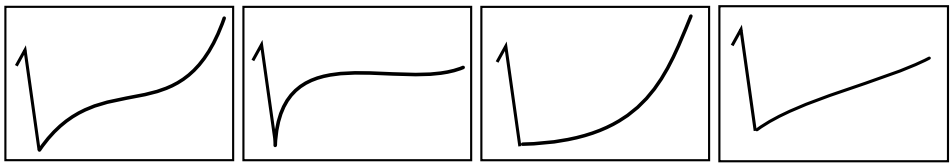



Fig. 8. Calculating $\alpha$ and $\beta$ from the 3 regions.



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

Fig. 9. Shapes identified by the classifier.

Table 1. Shape Classification.

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

(1) $\alpha_n = C_n - C_{n-1}, n \geq 3$ is the increase in cwnd between 2 successive measurements by Gordon for all RTTs after back off (see Fig. 8).
(2) $\beta = \frac{C_2}{C_1}$, is the proportion of back-off after packet loss.

Based on the division of the Congestion Avoidance phase into 3 regions, we found that the curves for the known cwnd-based TCP variants would take one of the 4 shapes shown in Fig. 9. We can computationally classify a curve into one of the 4 shapes based on the change in gradient ($\frac{d\alpha}{dt}$) for each region and by determining whether the steady region exists, as shown in Table 1.

Once we have the shape and the values of $\alpha_i$ and $\beta$, we can determine the variant from Table 2 by computing $\bar{\alpha}$, the mean of $\alpha_i$. Many of the values in Table 2 were obtained from the papers [3, 4, 15, 20–22, 28, 34, 37] describing the various algorithms. However, we found some difference when we measured the references traces obtained in our network testbed. Some adjustments were then made to ensure that the values of $\beta$ and $\bar{\alpha}$ reflected what we observed in our traces. Algorithms that react to loss, but cannot be classified into one of these shapes are classified as *Unknown*. We note that

Table 2. Known TCP Variant Classification.

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

CUBIC [15], BIC [37] and HTCP [21] can be identified by shape alone. Scalable [20], Illinois [22], CTCP [34], YeAH [2], New Reno [28], Veno [13], Westwood [7] and Vegas [3] all increase their cwnd linearly during Congestion Avoidance and are very similar in shape.

While most variants can be differentiated by their values of $\beta$ and $\bar{\alpha}$ (slope of the cwnd graph in the Congestion Avoidance phase), Gordon is not able to differentiate between three pairs of algorithms - CTCP and Illinois, New Reno and HSTCP and between Vegas and Veno. In the Congestion Avoidance phase, both Vegas and Veno initially increase their congestion window by 1 every RTT ($\alpha = 1$) before having more or less constant cwnd and are therefore indistinguishable when they interact with our network profile. Similarly, both CTCP and Illinois evolve their cwnd values using similar functions after seeing a packet loss. HSTCP and New Reno both back-off to half their cwnd on seeing a packet loss and increment their cwnd by 1 every RTT in Congestion Avoidance mode. Therefore, Gordon classifies them together as 'Vegas/Veno', 'CTCP/Illinois', and 'New Reno/HSTCP,' respectively. It remains as future work to introduce a second stage to disambiguate between these pairs (§5).

**Case 2: No Back-off**. For variants that do not back-off after a packet loss, we try to either classify them as BBR or an unknown variant. Even though BBR is a rate-based algorithm, it maintains a cwnd that is equal to twice the BDP. Also, since BBR uses the maximum receive rate in the past 10 RTTs for calculating it's BDP [5], we expect to see a drop in cwnd corresponding to our network profile's drop in bandwidth delayed by 10 RTTs.

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

### 3.4 Implementation

In Fig. 11, we present an overview of Gordon's system design. To inflate the RTT between our measurement server and the remote host (as discussed in §3.2), Gordon is run inside a Mahimahi delay shell [25]. We use `wget` [30] to emulate a browser making an HTTP/HTTPS GET request to

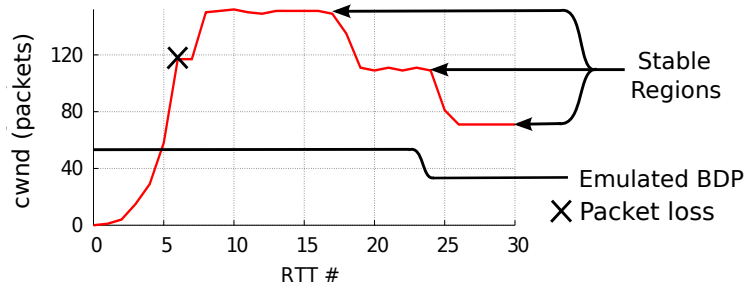



Fig. 10. Identifying stable regions for loss-agnostic flows.

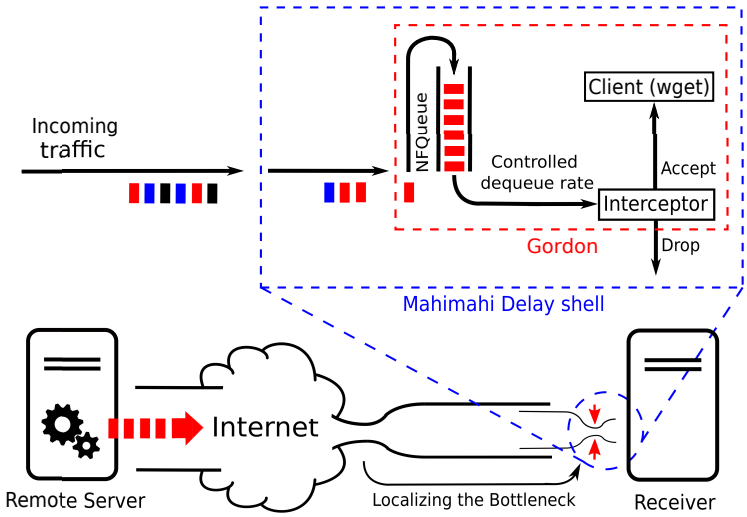



Fig. 11. Gordon Design.

the target web server. The incoming HTTP response packets are redirected to an NFQueue [26] using a Linux Netfilter redirect rule.

The interceptor module of Gordon dequeues packets from the NFQueue and selectively delivers them to `wget` or drops them. Gordon controls the rate at which packets are dequeued from the NFQueue to localize the bottleneck of the connection and to regulate the bottleneck bandwidth (as described in §3.2). The interceptor module is implemented in about 350 lines of C code. The final output consists of a trace of the maximum `cwnd` size observed for each RTT period, which is processed offline by a classifier written in 440 lines of Python code. For each website in the Alexa Top 20,000 list [18], we used a web crawler written in about 300 lines of Python code to obtain URLs to the largest web pages/objects that it could find on the website.

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

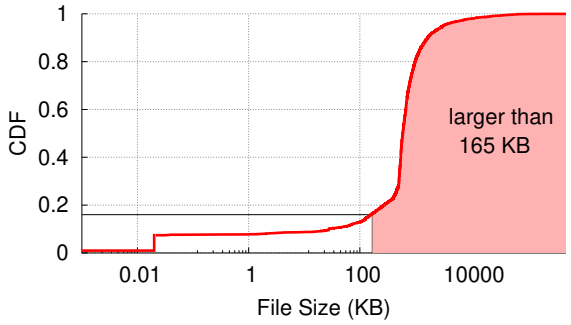



Fig. 12. CDF of file sizes used in measurements.

these five individual measurements separately. This web service was implemented in about 2,050 lines of TypeScript code.

At the moment, we have made Gordon and our measurements available on GitHub (see §8). We are still working to make the web service available as a live dashboard of the TCP variant distribution on the Internet.

## 4 RESULTS

We measured and classified the top 20,000 websites on the Internet based on their Alexa ranking [18]. These measurements were made between 11 July 2019 and 17 October 2019 (unless specified). The distribution of the file sizes obtained using a crawler (see §3.1) for the measurements is shown in Fig. 12. We can see that about 18% of the websites return pages smaller than the ideal page size of 165 KB (§ 3.1). We were able to classify the variants for some of these websites with page sizes smaller than 165 KB. We refer to the remaining websites that cannot be classified as 'Short flows'.

We also found that about 1,302 websites in the Alexa Top 20,000 list did not respond to `wget` requests. These websites had DDoS protection from Cloudflare or Google's ReCaptcha, and therefore did not respond to repeated `wget` requests. A small fraction of the websites also had invalid URLs that did not even open on a web browser. Upon further investigation, we found that these URLs were links to phishing websites that had been visited so often that they had made it to the Alexa Top 20,000 list. Collectively, we consider these websites to be 'Unresponsive'.

### 4.1 Verification of Measurement Accuracy

First, we validate the accuracy of our approach by setting up a physical test web server in Singapore and performing measurements from AWS EC2 instances in 9 locations (viewpoints): Paris, London, Ireland, Sydney, Seoul, Mumbai, Virginia, Oregon, and Ohio. The RTTs for the measurements ranged from 59 ms to 255 ms. To provide the ground truth, the test server runs one of the known TCP variants, which was then measured 5 times from each viewpoint, to give a total 45 measurements for each variant. Later, the configuration was reversed, with the AWS instances running a known variant and acting as web servers while a local server made measurements. In Table 3, we present the confusion matrix for these 90 measurements (per algorithm). The key takeaway is that for known variants, the accuracy is high and false positives are relatively rare. Note that the figures in Table 3 reflect the accuracy of single measurements. If we take the majority result across the five measurements from an individual viewpoint, we can achieve 100% classification success. The errors are caused by noisy measurements arising from Internet traffic.

Table 3. Classification accuracy.

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

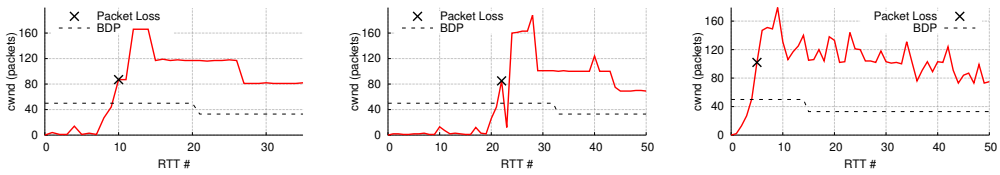
Table 4. Distribution of variants as measured from different viewpoints on the Internet.

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

### 4.2 TCP variants on the Internet

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

Table 4 shows the distribution of TCP variants on the Internet as measured from these viewpoints. As expected, we found that for certain websites, some viewpoints gave less noisy measurements compared to others. This is the only reason for the slight discrepancies between numbers reported from different viewpoints. Out of the 20,000 target websites, a total of 13,739 websites were classified similarly from all viewpoints. Out of the remaining 6,261 websites, 1,424 websites were successfully classified from some viewpoint and 3,535 websites could not be classified from any viewpoints.




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

Fig. 13. The evolution of BBR.

Table 5. Distribution of variants.

| Variant | Websites | Proportion |
|---|---|---|
| CUBIC [15] | 6,139 | 30.70% |
| BBR [4] | 3,550 | 17.75% |
| BBR G1.1 | 167 | 0.84% |
| YeAH [2] | 1,162 | 5.81% |
| CTCP [34]/Illinois[22] | 1,148 | 5.74% |
| Vegas [3]/Veno [13] | 564 | 2.82% |
| HTCP [21] | 560 | 2.80% |
| BIC [37] | 181 | 0.90% |
| New Reno [28]/HSTCP [12] | 160 | 0.80% |
| Scalable [20] | 39 | 0.20% |
| Westwood [7] | 0 | 0.00% |
| Unknown | 3,535 | 17.67% |
| Short flows | 1,493 | 7.46% |
| Unresponsive websites | 1,302 | 6.51% |
| Total | 20,000 | 100% |

The distribution of variants as measured from these viewpoints shows the same general trend of CUBIC [15] being the dominant congestion control variant in terms of website count, with BBR [17] coming in second. In Table 5, we show the consolidated numbers for all websites following the rule that if a website has been identified to be using some known congestion control variant in any of the regions, it is considered to be running that congestion control variant. There were no classification conflicts between different viewpoints for these 1,424 successfully classified websites. In other words, we found no evidence for websites deploying different congestion control algorithms in different regions.

**Google's custom version of BBR.** Gordon discovered that some Google-owned domains (167, including YouTube) were using a modified version of BBR that reacted differently to packet loss compared to vanilla BBR (see Fig. 13b). This difference was first observed in February 2019. Before that, we had observed traces resembling vanilla BBR (Fig. 13a). While we initially suspected that this new variant was BBRv2, we checked the cwnd evolution of BBRv2 that was recently released in July 2019 (see Fig. 13c) and found that it was not. We thus refer to this variant as BBR G1.1 in Tables 4 and 5. It should be noted here that this anomalous behavior was only observed for Google websites. None of the other websites identified to be using BBR showed this anomalous behavior even after repeated measurements. They all deployed vanilla BBR.

Table 6. Excerpt of website traffic share (source: Sandvine [32]).

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

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

We have confirmed our findings about BBR with Google. In particular, Google is frequently running experiments and testing refinements to BBR. Google currently runs a slightly modified version of BBRv1 that has a gentler reaction to packet loss than the open-source BBRv1. This experimental variant (BBR G1.1) was meant as an incremental step toward BBRv2. However, BBR G1.1 was deployed in late 2017, which does not explain our observation of a trace resembling vanilla BBR from Google websites in December 2018. We have thus been measuring Google sites repeatedly and found that we still see traces with the shape shown in Fig. 13a occasionally. Hence, it is possible that Gordon occasionally fails to detect the drop in cwnd for BBR G1.1 immediately after a packet loss event from time to time. At some level, this is not surprising since BBR does not actively maintain a cwnd like traditional cwnd-based TCP variants.

### 4.3 Traffic Volume & Popularity

We believe that the distribution of TCP variants by pure website count in Table 5 does not present the full picture.

**Understanding Traffic by Volume**. In Table 6, we present Internet traffic volume data by Sandvine [32]. Based on the reported Internet traffic volume, we expect BBR variants to already contribute at least 40% of the global Internet traffic. During our measurements, we found that Netflix had switched from CUBIC to BBR in early March 2019, only to switch back to CUBIC in April 2019. We note that Google recently announced that Netflix is currently experimenting with BBR [6]. We also contacted Netflix and were told that the Netflix website was hosted on AWS. Netflix however uses different protocols depending on the context, and that most of their video streaming traffic is delivered via their Open Connect Appliances running FreeBSD's New Reno with RACK [9] extensions. The reason for choosing New Reno over CUBIC was that the Netflix team felt that the New Reno stack was more mature and that improving loss-detection/loss-recovery heuristics from RACK would be more helpful for their chunked-delivery use case. We were informed by the Akamai team that Akamai would be deploying BBR G1.1 on more of their hosted sites in the near future. If Netflix and Akamai does do the switch to BBR, BBR and its variants' traffic share on the Internet would increase to well above 50%.

**Understanding Traffic by Popularity**. Similar trends can also be observed if we consider the popularity of the websites. In Fig. 14, we plot the distribution of the identified variants for the top-k sites. We see that BBR is the most widely deployed variant among the top 250 websites, accounting for 25.2% of all hosts. Another interesting observation was that BBR was the most common TCP variant for adult entertainment websites. All in all, our results suggest that BBR is rapidly catching

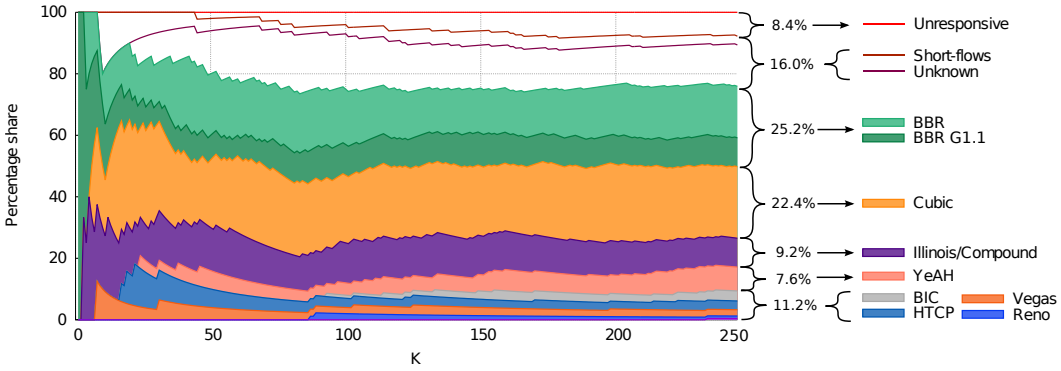



Fig. 14. Distribution of variants among the Alexa Top-$k$ sites.

Table 7. Custom network profiles to investigate uncategorized hosts.

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

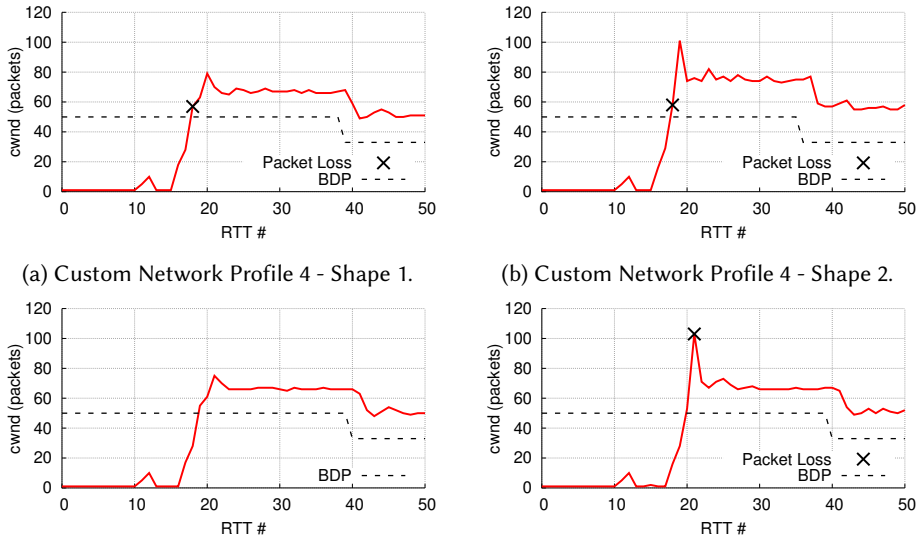
up with CUBIC in popularity and some variant of BBR is poised to overtake CUBIC as the dominant TCP variant.

### 4.4 Whithering the Unknown Variants

One of the benefits of our methodology is that Gordon can provide us with insights on a congestion control variant even if we are not able to identify it. Given that a larger number of websites (5,028 in total) were classified as 'Unknown' or 'Short flows' (together referred to as 'Uncategorized' hosts henceforth), we ran a variety of *new* network profiles to investigate their behavior under different conditions. These network profiles were designed with different combinations of emulated BDPs, delays and Packet Drop Thresholds. We hypothesize that the same TCP variant would exhibit the same behaviors for all network profiles, while different TCP variants may exhibit the same behavior for some profiles, but different behavior for others, to allow us to tell them apart. Our goal is to identify large clusters of traces that could suggest the presence of a new major, but hitherto unknown, variant.

Given that Gordon can modify these three network parameters, we came up with eight custom network profiles (shown in Table 7) that are distributed over the range of these network parameters. Each of these network profiles emulates a fixed RTT and BDP for an experiment run and introduces a packet drop when the cwnd size goes above the Packet Drop Threshold for the first time.

**Reaction to Loss.** We found that among the 5,028 (25.14%) websites with unknown variants, only 3,275 (16.38%) of them reacted to packet loss. Out of these 3,275 websites, 1,493 (7.47%) are




(a) Custom Network Profile 4 - Shape 1.

(b) Custom Network Profile 4 - Shape 2.

(c) Custom Network Profile 1 - Shape 1.

(d) Custom Network Profile 1 - Shape 2.

Fig. 15. Sample traces for websites hosted by Akamai.

short flows and the remaining 1,782 (8.91%) websites gave inconsistent measurements after reacting to a packet loss. In other words, repeating our measurements yielded different traces each time. We hypothesize that these are likely cwnd-based TCP variants that we are not able to classify because of noise. It was surprising that this noisiness was observed for all 5 viewpoints.

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

**Akamai Congestion Control Variant.** We found that all 1,103 (5.52%) websites that reacted to changes in the BDP but not to packet loss were all hosted by the Akamai CDN. These websites typically maintained the cwnd at a fixed multiple of the BDP, ranging from 1.2 to 1.5. Typical shapes for these websites are shown in Fig. 15. We found that the traces for the AkamaiCC websites in response to our custom network profiles tend to take one of 2 shapes shown in Figs. 15a and 15b. As shown in Figs. 15c and 15d, these shapes remain consistent across different network profiles. While this behavior matches no known TCP implementation in the Windows or the Linux kernel, we hypothesized that it was the result of TCP optimizations developed at Akamai [1] or the deployment of FAST TCP [36]. We refer to this variant as *AkamaiCC* in the rest of the paper. Some notable websites identified to use AkamaiCC include `microsoft.com`, `apple.com`, and `hulu.com`.

We found that a total of 1,260 (6.30%) websites among the Alexa Top 20,000 websites were hosted by Akamai, but not all of them show the behavior illustrated in Fig. 15. All the remaining 157 (0.79%) Akamai-hosted websites did not react to loss and yielded noisy measurements and so are categorized as 'Unknown.' It is plausible that these websites are also running AkamaiCC, but we are not able to see the AkamaiCC shape in their traces because of noise.

We contacted Akamai to verify these results and they have confirmed that AkamaiCC was likely FAST TCP. That said, the Akamai team also highlighted that Akamai does not typically deploy a

Table 8. Summary of websites not classified as known congestion control variants.

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

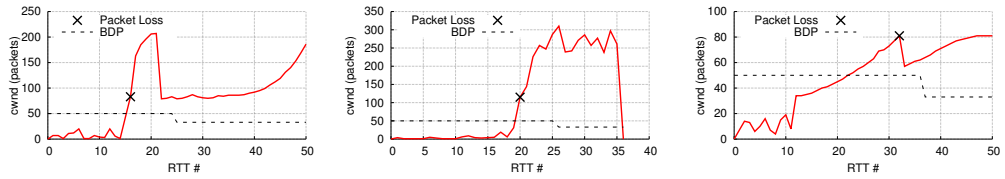
specific TCP variant for a specific website (though there were cases where they might). It is plausible that our findings were an artifact of our experimental setup and the pages that we had chosen to download from the Alexa Top 20,000 websites. Akamai currently deploys a variety of Congestion Control variants including FAST TCP, a modified version of Reno, vanilla BBR, a modified version of BBR, QDK (a proprietary Congestion Control algorithm), and CUBIC. In addition, under some network conditions, Akamai servers could switch between these algorithms in the middle of a connection. This would be a plausible explanation for the noisy and unrecognizable traces observed for some of the 157 Akamai-hosted websites that we could not classify.

In summary, as shown in Table 8, a large number of the websites that Gordon was not able to identify as known variants can be attributed to the 1,103 (5.51%) websites running AkamaiCC. In other words, Gordon can classify 14,773 (73.87%) of the Alexa Top 20,000 websites as some variant. Among the remaining 5,227 (26.14 %) websites, 1,302 (6.51%) were found to be unresponsive, 1,493 (7.47%) had web pages that were too small to yield a long enough trace for classification, and 2,432 (12.16%) could not be classified because most of them yielded noisy and inconsistent traces.

**The best of the rest.** We know from our results in §4.1 that some of the websites classified as one of the 2,432 "Unknown" websites would be known variants that Gordon is not able to identify because of noise. However, it is likely that there also new and undocumented variants because of the diverse behaviors that we observed. We reproduce 3 of the more interesting traces in Fig. 16:

(1) `amazon.com`: In Fig. 16a, we see that Amazon has deployed a TCP variant which resembles HTCP in its Congestion Avoidance phase. However, the variant either does not back-off on seeing a Gordon-induced packet loss or has a significant multi-RTT delay in its response to loss. The reduction at `cwnd` = 200 is not due to buffer overflow. The buffer for Gordon can hold significantly more packets than that without suffering overflow.
(2) `zhihu.com`: The behavior observed for the trace in Fig. 16b showed an unrestricted growth in the sender's `cwnd`. It seems like the deployed TCP variant is oblivious to packet losses and bandwidth changes, and simply maintains a very high and constant `cwnd`.
(3) `yahoo.co.jp`: The behavior in Fig. 16c suggests that whatever congestion control variant Yahoo has deployed is exiting Slow Start prematurely, and conservatively increases its `cwnd` until it sees a packet loss.

We believe that these examples serve to illustrate the diversity in the unknown variants and would convince the reader that it was not embarrassing that we have not been able to identify these variants in the first instance. It would likely take another full paper to comprehensively analyze and classify these variants. At some level, these results are also a wake-up call. In academia, we often assumed that we would be the ones to invent new TCP variants and the industry would




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

Fig. 16. The weird and wonderful universe of TCP "in the wild."

Table 9. Evolution of TCP variants on the Internet over the past 2 decades.

| | 2001 [27] | | 2004 [24] | | 2011 [38] | | 2019 | |
|---|---|---|---|---|---|---|---|---|
| Loss-based AIMD | New Reno | 35% (1,571) | New Reno | 25% (21,266) | | | New Reno[h] | 0.80% (160) |
| | Reno | 21% (945) | Reno | 5% (4,115) | AIMD | 12.46% (623) | Reno[d] | - |
| | Tahoe | 26% (1,211) | Tahoe | 3% (2,164) | | | Tahoe[d] | - |
| Loss-based MIMD | - | - | - | - | CUBIC | 22.30% (1,115) | CUBIC | 30.70% (6,139) |
| | | | | | BIC | 10.62% (531) | BIC | 0.90% (181) |
| | | | | | HSTCP | 7.38% (369) | HSTCP[h] | |
| | | | | | Scalable | 1.38% (69) | Scalable | 0.20% (39) |
| Delay-based AIMD | - | - | - | - | Vegas | 1.16% (58) | Vegas[v] | 2.82% (564) |
| | | | | | Westwood | 2.08% (104) | Westwood | 0% (0) |
| Delay-based MIMD | - | - | - | - | CTCP | 6.68% (334) | CTCP | 5.74% (1,148) |
| | | | | | Illinois | 0.56% (28) | Illinois | |
| | | | | | Veno | 0.90% (45) | Veno[v] | |
| | | | | | YeAH | 1.44% (72) | YeAH | 5.81% (1,162) |
| | | | | | HTCP | 0.36% (18) | HTCP | 2.80% (560) |
| Rate-based | - | - | - | - | - | - | BBR | 17.75% (3,550) |
| | - | - | - | - | - | - | BBR G1.1 | 0.84% (167) |
| | - | - | - | - | - | - | AkamaiCC | 5.51% (1,103) |
| Unknown | 17.30% (792) | | 53% (44,950) | | Unknown | 3.96% (198) | | 12.16% (2,432) |
| | | | | | Abnormal SS[*] | 2.88% (144) | | |
| Short flows | - | | - | | | 26% (1,300) | | 7.47% (1,493) |
| Unresponsive | 0.7% (30) | | 14% (11,529) | | - | | | 6.51% (1,302) |
| Total hosts | 100% (4,550) | | 100% (84,394) | | | 100% (5,000) | | 100% (20,000) |

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

subsequently, pick the winner. The development of BBR has been led by Google, and it seems that companies such as Akamai, Amazon, and Netflix, are not too far behind.

### 4.5 TCP Evolution over the past Two Decades

To have an overview of TCP evolution on the Internet over the years, we compare our results to previous studies conducted by Padhye et al. [27], Medina et al. [24] and Yang et al. [38] in Table 9. Since the total hosts measured over the various studies vary widely and there is also a large variance in terms of success rates, we normalize the results over the total reported successful classifications for each study in Table 10 to obtain estimated proportions of the various variants. New Reno had

Table 10. Share of TCP variants normalized over all successful classifications.

| | 2001 [27] | | 2004 [24] | | 2011 [38] | | 2019 | |
|---|---|---|---|---|---|---|---|---|
| Loss-based AIMD | New Reno | 35% | New Reno | 29% | AIMD | 17% | New Reno[h] | <1% |
| | Reno | 21% | Reno | 6% | | | Reno[d] | - |
| | Tahoe | 26% | Tahoe | 3% | | | Tahoe[d] | - |
| Loss-based MIMD | - | - | - | - | CUBIC | 30% | CUBIC | 36% |
| | | | | | BIC | 14% | BIC | 1% |
| | | | | | HSTCP | 10% | HSTCP[h] | |
| | | | | | Scalable | 2% | Scalable | <1% |
| Delay-based AIMD | - | - | - | - | Vegas | 2% | Vegas[v] | 3% |
| | | | | | Westwood | 3% | Westwood | 0% |
| Delay-based MIMD | - | - | - | - | CTCP | 9% | CTCP | 7% |
| | | | | | Illinois | <1% | Illinois | |
| | | | | | Veno | 1% | Veno[v] | |
| | | | | | YeAH | 2% | YeAH | 7% |
| | | | | | HTCP | <1% | HTCP | 3% |
| Rate-based | - | - | - | - | - | - | BBR | 21% |
| | - | - | - | - | - | - | BBR G1.1 | 1% |
| | - | - | - | - | - | - | AkamaiCC | 6% |
| Unknown | | 18% | | 62% | Unknown | 5% | | 14% |
| | | | | | Abnormal SS[*] | 4% | | |
| Total Measurable hosts | | 100% | | 100% | | 100% | | 100% |

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

rapidly surpassed Reno as the dominant TCP variant in early 2000's. The next 10 years saw the rise of loss-based MIMD protocols such as CUBIC and BIC which dominated the overall adoption decreasing the share of loss-based AIMD protocols. By 2011, HSTCP and Microsoft's CTCP also held significant shares of 10% and 9% respectively.

Five years later in 2019, traditional loss-based AIMD schemes have become near-extinct (at least in terms of pure website count). On the other hand, the adoption of delay-based Vegas has almost doubled. CUBIC has remained the most popular TCP variant and has increased its share among the top 20,000 hosts to some 36% compared to 30% in 2011, while the share of BIC and HSTCP has reduced significantly.

While delay-based variants seem to have become slightly more popular over the past decade (increasing in share from 15% in 2011 to 20% in 2019), CTCP seems to have slightly decreased in popularity in recent years. We suspect that this is likely due to Microsoft's addition of CUBIC as an option in Windows Server 2016 and making CUBIC the default congestion control algorithm in Windows 10 (2019 builds) and Windows Server 2019 [29].

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

## 5 DISCUSSION AND FUTURE WORK

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

The Gordon project started as a measurement study to understand the latest distribution of TCP variants on the Internet since the last study was done 8 years ago. We decided to start with a simple approach of measuring cwnd one RTT at a time. While the modifications we adopted (§3.1) might seem straightforward, in hindsight, it took us a while to get the details right and collect all the data. It turns out that our chosen approach made the mitigation of random losses much easier and works well for the vast majority of websites surveyed.

**Bringing Uncooperative Sites On-board**. We suspect that our approach in making a large number of abrupt connections can be improved. In particular, we observed that many of the websites for which we were not able to identify the TCP variant were in the banking and government sectors (§4.2). We are not entirely surprised that we were often throttled or blocked in the midst of a measurement run since our connections would seem to be misbehaving to the TCP sender. We have classified these hosts as 'Unresponsive' in Table 5. Currently, we are studying how we can optimize the probing of target websites without restarting new connections so that we can reduce the proportion of unresponsive websites.

**Easy Extensions**. While we had hoped to design "one tool to measure them all," we have subsequently realized that there is a limitation to our approach. Because we normalize a host's sending behavior by RTT, behavioral differences that exist at a granularity smaller than one RTT cannot be observed. This would also explain why we are often unable to observe a drop in the cwnd after a packet loss event for the currently deployed Google G1.1 BBR variant (see §4.2). That said, the variants that Gordon is not able to differentiate are relatively small and insignificant in terms of popularity, so we did not invest more time to work on them. In principle, we can classify them by adding a second stage to the classification process after Gordon is done with a high-level classification. In the future, we plan to add such extensions to Gordon for differentiating between such variants.

**Further Exploration**. There is still room to investigate the sites that we have not been able to classify successfully. For example, the hosts with loss-agnostic behavior might react to multiple packet losses (instead of one) and a new network profile (§3.2) that drops several packets or explicitly sends triple duplicate ACKs could potentially be able to detect such behavior. Since our key design principle is to look for generic characteristics such as reaction to bandwidth changes, delay and different types of loss, Gordon can easily adapt to and discover new future variants that are not known today.

**Understanding Long-Term TCP Evolution**. Our results suggest an active push by large Internet companies towards rate-based TCP variants. However, as common cloud service providers like Google Cloud are now enabling BBR [14] and the Akamai CDN is running AkamaiCC, small entities using these services might be making the switch to rate-based variants without knowing it. This suggests that the landscape of TCP congestion control is undergoing rapid and significant change, possibly led by the CDNs. Therefore, we do not think that taking a snapshot every 10 years is good enough. Moving forward, we plan to enhance and automate Gordon to record a continuous view of the Internet's ongoing transition to a new era of rate-based congestion control.

## 6 CONCLUSION

In this paper, we used Gordon to identify the TCP variants for the top 20,000 websites based on their Alexa rankings [18]. Our results suggest that CUBIC is currently still the dominant TCP variant on the Internet and is deployed at 36% of the Alexa Top 20,000 websites that we successfully classified. Rate-based TCP variants like BBR have the next largest share. While BBR and its variant BBR G1.1 have a share of only 22% in terms of website count, their present share of total Internet traffic volume is likely to be larger than 40% [32]. This proportion will almost certainly exceed 50% if Netflix also decides to adopt BBR.

Since it is natural for the Internet to evolve, this is not the first time that we are seeing a dominant TCP variant in the process of being replaced by an alternative. However, we believe that the current change represents a fundamental shift in the underlying Internet. In previous transitions, all the TCP variants were cwnd-based and the interactions between AIMD/MIMD protocols have been well-understood. BBR represents a fundamental departure in our approach to congestion control. While BBR has been studied and issues have been highlighted [17, 35], to the best of our knowledge, the interactions between BBR and CUBIC at scale are not fully understood. While nothing seems to have broken thus far even as BBR has gained traction, we do not yet know for sure that nothing will necessarily go wrong. We believe that our results suggest the need for more in-depth study in the interactions between BBR and CUBIC to ensure the future stability and success of the Internet.

## 7 ACKNOWLEDGMENTS

We thank our shepherd, Michael Sirivianos and the anonymous SIGMETRICS reviewers for their valuable feedback and comments. We thank Neal Cardwell, Yuchung Chen, Nimantha Baranasuriya, Venkat Padmanabhan, Daniel Havey, Praveen Balasubramaniam, Igor Lubashev, and Mike Afergan, for their suggestions and for their help in corroborating some of the findings presented in this paper.

## 8 RESOURCES

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

## REFERENCES

[1] Akamai Developer Blogs 2019. TCP Optimizations - Akamai Developer. (2019). https://bit.ly/35LYJUx.
[2] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. 2007. YeAH-TCP: yet another highspeed TCP. In *Proceedings of PFLDnet*.
[3] 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*.
[4] 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.
[5] 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
[6] 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
[7] 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.
[8] 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. *arXiv preprint:1904.04430* (2019).
[9] Yuchung Cheng, Neal Cardwell, Nandita Dukkipati, and Priyaranjan Jha. 2019. RACK: a time-based fast loss detection algorithm for TCP. IETF Draft. (2019). https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-rack-05
[10] Douglas E. Comer and John C. Lin. 1994. Probing TCP Implementations. In *Proceedings of USTC*.
[11] DARPA. 1981. Internet Protocol. RFC 791. (1981).

[12] Sally Floyd. 2003. HighSpeed TCP for Large Congestion Windows. RFC 3649. (2003).
[13] 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.
[14] Google Cloud Blogs 2017. TCP BBR congestion control comes to GCP: your Internet just got faster. (2017). https://bit.ly/2Hk4WLH
[15] 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.
[16] Desta H. 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.
[17] Mario Hock, Roland Bless, and Martina Zitterbart. 2017. Experimental Evaluation of BBR Congestion Control. In *Proceedings of ICNP*.
[18] Alexa Internet Inc. 2018. The Top 500 websites on the Internet. (2018). https://www.alexa.com/topsites
[19] Yuchung Cheng Jerry Chu, Nandita Dukkipati and Matt Mathis. 2013. Increasing TCP's Initial Window. RFC 6928. (2013).
[20] Tom Kelly. 2003. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *SIGCOMM CCR* 33, 2 (2003), 83–91.
[21] 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*.
[22] 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*.
[23] Ralf LÃijbben and Markus Fidler. 2016. On characteristic features of the application level delay distribution of TCP congestion avoidance. In *Proceedings of ICC*.
[24] Alberto Medina, Mark Allman, and Sally Floyd. 2005. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM CCR* 35, 2 (2005), 37–52.
[25] 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*.
[26] Netfilter Organization. 2019. libnetfilter_queue. (2019). https://bit.ly/2HimY17
[27] Jitendra Padhye and Sally Floyd. 2001. On Inferring TCP Behavior. In *Proceedings of SIGCOMM*.
[28] Vern Paxson and Mark Allman. 2009. TCP Congestion Control. RFC 5681. (2009).
[29] Brien Posey. 2019. Explore the Cubic congestion control provider for Windows. (2019). https://bit.ly/2VfhxoA
[30] GNU Project. 2019. wget. (2019). https://www.gnu.org/software/wget
[31] Jan Rüth, Christian Bormann, and Oliver Hohlfeld. 2017. Large-scale scanning of TCP's initial window. In *Proceedings of IMC*.
[32] Canada Sandvine Inc. Waterloo, ON. 2018. The 2018 Global Internet Phenomena Report. (2018). https://www.sandvine.com/phenomena
[33] W. Sun, L. Xu, and S. Elbaum. 2018. Scalably Testing Congestion Control Algorithms of Real-World TCP Implementations. In *Proceedings of ICC*.
[34] 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*.
[35] Ranysha Ware, Matthew K. Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Modeling BBR's Interactions with Loss-Based Congestion Control. In *Proceedings of IMC*.
[36] David X Wei, Cheng Jin, Steven H Low, and Sanjay Hegde. 2007. FAST TCP. *IEEE/ACM Transactions on Networking*.
[37] Lisong Xu, K. Harfoush, and Injong Rhee. 2004. Binary increase congestion control (BIC) for fast long-distance networks. In *Proceedings of INFOCOM*.
[38] 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.
[39] Peng Yang and Lisong Xu. 2011. A survey of deployment information of delay-based TCP congestion avoidance algorithm for transmitting multimedia data. In *Proceedings of GLOBECOM Workshops*.

Received August 2019; revised September 2019; accepted October 2019