TCP's Initial Window – Deployment in the Wild and its Impact on Performance

Jan Rüth, Ike Kunze, Oliver Hohlfeld

***Abstract*—TCP congestion control and particularly its initial congestion window (IW) size is one long-debated topic that can influence web performance. Its size is, however, assumed to be static by IETF recommendations—despite being network- and application-dependent—and only infrequently changed in its history. To understand if the standardization and research perspective still meets Internet reality, we study the IW configurations in IPv4 and of major content delivery networks (CDNs). We have been regularly inspecting IPv4 for HTTP and TLS servers to investigate their IW configuration and found a steady increase in IETF-recommended configurations. We additionally study how CDNs configure their IWs given their relevance for content distribution. To shed light on network-dependent CDN configurations, we use a globally distributed infrastructure of VPNs giving access to residential access links. We observe that most CDNs are well aware of the IW's impact and find a high amount of customization that is *beyond* current Internet standards. We find various initial window configurations, most below 50 segments, yet, with exceptions of up to 100 segments—the tenfold of current standards. Our study highlights that Internet reality has drifted away from recommended and standardized practices. Driven by these findings, we investigate the effects of this new reality on the slow start of Cubic and BBR congestion controlled TCP flows. We find that TCP pacing is a key to enable increased IWs when competing against other traffic.**

***Index Terms*—TCP congestion control, Initial Congestion Window, Pacing, Measurements, Cubic, BBR**

I. Introduction

The Internet and specifically the web have transformed the way we gather information, interact or do business. This increasing dependence on the web has fueled a pursuit of researchers and operators to develop and implement web performance optimizations. For example, Google has pushed several improvements to web technology, including new protocols such as HTTP/2 (through SPDY) or QUIC, both of which have found swift adoption [1]–[3] by others. Apart from technological advances, content distribution networks (CDNs) have changed the Internet on an architectural scale. Their ongoing quest to serve web content from nearby servers has flattened the hierarchical structure of the Internet [4] thereby promising lower latencies. In pursuit of performance, CDNs are known to be early adaptors of new technology in an attempt to optimize the web experience for their customers.

While adopting new technologies offers promising gains, their correct configuration is often challenging—e.g., HTTP/2 server push is regarded as a key feature but known to be notoriously hard to use [3], [5], [6]. Some of these configuration challenges stem from the fact that they are dependent on network and application characteristics.

One long-debated performance configuration parameter is TCP's (and QUIC's) initial congestion window (IW) size. The IW characterizes the performance at the beginning of a new connection and is a key component of all congestion control algorithms. The IW size controls the amount of unacknowledged data sent at connection start and thereby heavily influences the start-up behavior of new and especially short-lived connections (e.g., typical web transfers) or those that are revived from idle. A small IW can prolong transmissions and cause unnecessary latency as the transport protocol needs to await feedback (ACKs) to increase the congestion window. Contrary, too large IWs can lead to loss and retransmissions when the network simply cannot handle large bursts of data. Thus choosing the optimal value for each network is critical for good performance—and thus interesting for CDNs.

Despite its relevance, the IW size is typically regarded as a *static* parameter whose IETF-recommended size should fit all networks and applications. Since its first definition to 1 segment in 1988 [7], its recommended size has only changed twice, to 2-4 segments in 1998 [8], [9] and—motivated by increasing access speeds and its promise to shorten page loading times [10]—to 10 segments in 2013 [11]. It was very recently shown that IW customization can help in reducing CDN latency [12]. In this regard, a small-scale study by CDNPlanet showed that half of the probed CDNs use IW10 as the IETF-recommended size while others already use larger IW sizes [13]. Others [14] even argue to abandon static IETF-standardized values for the IW to enable customization already in the standards. Yet, little is known about IW configurations especially in case of CDNs.

In this paper, we broadly probe IPv4 hosts and specifically CDNs to gather an empirical understanding on how IW customization already takes place in today's Internet. We investigate the prevalence of IETF-recommended values in samples of IPv4 over a period of 1.5 years extending our previous work [15], [16]. Further, we investigate CDN IW configurations from globally distributed vantage points and from our University's network, thereby shedding a light on the degree of customization that CDNs show today. Our results show that the Internet at large converges towards the current IETF recommendation of 10 segments. On the other hand, we observe that IW customization beyond standardized practices is already common practice and there exists a gap between standardization, research, and Internet reality.

Driven by this gap, we further extend our work [16] and investigate the effects of this new Internet reality on the performance of slow start when competing against other flows. To this end, we seed a testbed study by our real-world observations and investigate the start-up performance benefits and disadvantages when the traditional CUBIC and the recent BBR congestion control have to compete for traffic against an elephant flow. Our findings indicate that the way CDNs utilize IW customization can indeed yield drastic performance increases. Specifically, this work contributes the following:

- The first long term analysis of the evolution of IW configurations in IPv4. Our results show a convergence towards the IETF-recommended values at Internet-scale.
- We provide a comprehensive analysis of current IW configuration practices of CDNs. We show that IWs are configured up to ten times larger than recommended by IETF's current experimental standard.
- Further, we find multiple CDNs which use customized IW configurations, i.e., deliver data using different IWs for different customers or service types. We observe that larger IWs are for example preferred for streamed video instances, yet, content types do not necessarily enforce certain IW settings.
- By analyzing IWs through different geographically distributed networks, we find instances of network-dependent IW configurations of CDNs.
- We investigate the burstiness of IWs and find that some CDNs utilize pacing to space out packets over time during slow-start to potentially reduce the chance of losses.
- Our measurements show different configurations of packet pacing, challenging the traditional notion of IWs, we find that a data rate better captures the demand on a network than a fixed number of segments.
- We build a testbed to evaluate our real-world observations in a controlled environment; we find that increasing IWs *can* increase or hurt performance, specifically our investigations show that increasing the IW should go hand in hand with TCP pacing to actually benefit.

Structure. Section II discusses related work and shows how IWs are defined, standardized, and how they impact performance. Our IW scanning methodology, IP-based IPv4 scans and the resulting changes for our scanner architecture to investigate CDNs is introduced in Section III. Following, Section IV–Section VI paint the global CDN IW configuration space by discussing CDN specific IW configuration. In Section VII, we project our real-world findings to a testbed and investigate its impact on flow start-up behavior. Finally, Section VIII concludes our findings.

II. TCP's Initial Congestion Window

We start by exploring TCP's Initial Congestion Window (IW): *i)* how it is defined and sized, *ii)* how its size influences flow completion time, and *iii)* how related works have gathered an understanding on IWs through Internet measurements.

IW Definition. TCP's IW governs the number of unacknowledged bytes in flight until the first acknowledgment is received. That is typically data sent in the first roundtrip of a connection after the three-way handshake is completed. Thus, the IW at the sender-side and the receive window at receiver side define the application's data rate at the start of the connection and bootstraps the window doubling during slow start. Furthermore, depending on the congestion control algorithm, the IW is also used after long idle periods for restart (e.g., in web browsers when using HTTP/2).

IW Size Definition. The IW is typically defined in bytes and often operating systems allow configuration of the IW in multiples of the maximum segment size (MSS). To this end, many RFCs (here at the example of [11]) define a dualism for the IW, either in terms of the multiple of the MSS, e.g., IW 10 for ten segments worth of data, or by an upper limit of bytes, e.g., 14600 byte typically corresponding to a classic full ethernet frame minus TCP and IP headers times the multiple.

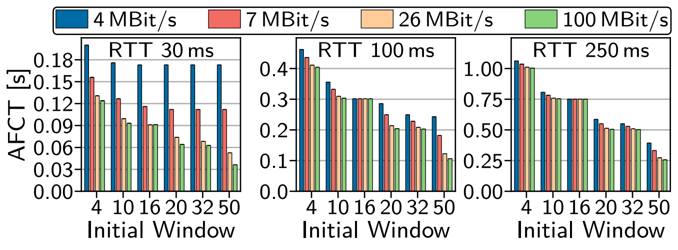

Fig. 1: Average flow completion time (AFCT) when varying bandwidth, latency, and IWs. Horizontal lines mark roundtrips. Larger IWs *can* improve the latency.

A. Testbed Study: Impact of IW Size on Internet Performance

To highlight the impact of different IW sizes on Internet performance, we conduct a testbed study. The testbed involves two directly connected Gigabit Ethernet Linux hosts whose link bandwidth and latency are controlled by NetEm in each host. We select four bandwidth configurations (i.e., 4, 7, 26, and 100 MBit/s) and three delay configurations (i.e., 30, 100, and 250 ms) to reflect typical Internet access characteristics reported by Akamai [17]. We further choose six IWs: 4, 10, 16, 20, 32, and 50 segments, to reflect the current standard of 4 segments [9], the current IETF recommendation of 10 segments [11], and larger IWs. In each experiment, we transfer a single flow of size 71 kB, i.e., 50 frames of data (the average size of the Google landing page in 2017). For each configuration, each experiment is repeated 30 times.

Flow Completion Time. Given its relevance to web browsing, we first measure the TCP flows' average flow completion time (AFCT) subject to the different parameters (i.e., IW size, RTT, and bandwidth). We define the AFCT as the average time to the last byte of the flow. The AFCT for the different parameters and its standard deviation is shown in Figure 1. It shows that increasing the IW reduces the AFCT if the link speed or RTT is sufficiently high. For low bandwidth connections with low latency, larger IWs have effectively no impact on the AFCT as these connections are limited by throughput. However, when higher speeds are available, increased IWs can effectively shorten the required roundtrips to finish the data transfer—a key motivation for CDNs to configure larger IWs.

Retransmissions. Large IWs, however, yield more bursty traffic that can lead to temporary phases of congestion more easily, reflected in higher loss rates. To highlight this effect, we conduct a second experiment which measures the average retransmission rates of the TCP flows subject to different bottleneck link configurations. We realize this setting by now connecting the hosts via a bottleneck router with different bandwidth capacities and queue sizes (QLIMs) of a regular drop tail FIFO queue. We again transfer 71 kB and vary the IW configurations for each bandwidth, queue size, and IW triple,

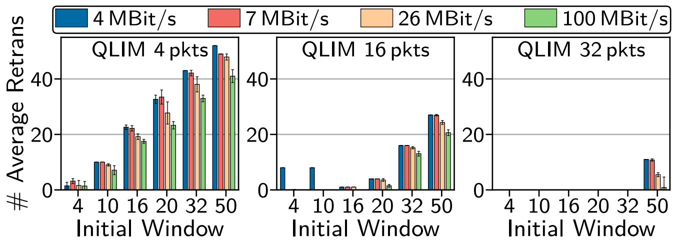

Fig. 2: Losses increase when increasing IWs depending on bottleneck bandwidth and queue sizes. Benefits of increased initial congestion windows are highly network-dependent.

testing every configuration 30 times. We show the average retransmissions required and standard deviation in Figure 2.

As the figures show, increasing the IW can have detrimental effects on the connection. We observe that larger IWs cause higher loss rates when either the bottleneck bandwidths or the bottleneck queue sizes are too small. Considering the retransmissions for the smallest queue size, we can see that large IWs cause heavy losses. Increasing the queue size helps in buffering the IWs, yet at the cost of added latency, e.g., a 7 Mbit/s link with a buffer of 16 packets can add up to 27 ms of delay to a packet. Thus, simply increasing the queue size is not a desired solution. While these motivating measurements neglect multiple users sharing the bottleneck, loss-based congestion control of multiple users will lead to full queues all of the time leading to tight buffer space for new flows as shown in these measurements.

Takeaway. *Our study shows, similar to related works [10], [18], that the IW size can strongly influence flow performance but can also overload congested or low-bandwidth connections. It is thus key to congestion control to correctly set an initial congestion window that adequately balances throughput and loss to bootstrap a TCP connection.*

B. Related Work

The relevance of TCP's initial congestion window size is reflected in an extensive debate and a successive evolution of its value in the TCP standards over the last decades. Initially, the IW was set to 1 segment in 1988 [7] and 9 years later standardized in 1997 [19]. This setting was experimentally extended to 2-4 segments (or 4380 byte) in 1998 [8] and later moved to a proposed standard [9]—a setting that remained untouched for a decade. Motivated by the increase of network access speeds and the desire to reduce web page loading times, [10] proposed in 2010 and later RFC 6928 [11] recommended in 2013 to increase the IW to ten segments. Most recently, Allman [14] even argues for abandoning a specification of the IW size and thus ending a decades-long debate. This argument is motivated by allowing hosts to configure more tailored IWs.

Given the relevance of the IW on both flow completion times and Internet traffic burstiness leading to losses, an empirical study of the IW is necessary to understand current network performance aspects. This understanding has been gained in both active and passive measurement studies. With regard to active measurements, Medina et al. [20] probed 85 k servers in 2004 and found most servers to be on an IW of one or two with only 1% of hosts having an IW larger than four. Our measurements are similar to those of Medina in terms of methodology but we especially focus on CDNs which were still on the rise in 2004 and did not have as much footprint as today. Regarding passive measurements, Qian et al. [21] inferred IW distributions from several traces in 2009. While their dataset covers traces captured in a diverse set of networks and also covers non-publicly visible hosts they did not discuss the impact of CDNs in their study. A small-scale study by CDNPlanet [13] probed 15 CDNs via HTTP and found 6 to use IW10 and others to use larger IWs. Our work is similar to that of CDNPlanet, we share the same goal to shed light on CDN IWs but their methodology (manual inspection of packet traces) limits a broad assessment of CDN IW configurations which is one focus of this work. An unknown vantage point and limiting TCP receive windows further limits the comparability of their study to ours. In [15], we proposed an approach to estimate TCP's IW for all reachable IPv4 HTTP and TLS hosts. We used this approach in [16] to estimate CDN IW configurations. This paper extends upon these works as we use our findings that 1% random samples of IPv4 are enough to estimate the IW distribution in IPv4 and provide a longitudinal analysis of the evolution of IWs in IPv4 over the course of 1.5 years. Further, we extend our analysis and provide IW estimations for more CDNs while also extending our result discussion, e.g., providing more insights into the packet pacing that we found. Moreover, this paper extends these works by projecting our real-world findings back into a testbed to evaluate the effects of increased IWs on flow completion time when subject to competing traffic. In this regard, we examine the advances of TCP pacing. The idea of pacing TCP flows goes back to the observations of ACK clocking in [22] which motivated pacing in [23] to clock data until ACKs arrive but has first been extensively studied in [24]. Their simulative results employ a non-bursty pacing implementation and their focus is on long-lived connections in contrast to the impact of pacing at the start of the connection. Similarly, work on router buffer sizing [25] has shown that tiny buffers can only be realized when some form of pacing is used, an observation that we are able to qualitatively validate. Wei et al. [26] replicate many experiments from [24], e.g., with different TCP variants, and their simulations show that the congestion control algorithm itself has a large impact. Motivated by this, we rely on emulation of real implementations to study how pacing affects the slow start behavior, in this regard TCP Jump Start [27] is similar by abandoning an IW and simply pacing out all data. Yet, again, this work is based on simulations, to the best of our knowledge the behavior of Linux's pacer has not been analyzed in academic work which is a new contribution of this paper.

III. Measuring IW Configurations in the Wild

We next describe our approach to estimate the IW size, its validation, our overall scanning architecture, and results in the Alexa Top 1M and IPv4.

A. Measuring IWs

We begin by summarizing our IW size estimation approach which is based on our previous work in [15]. To also enable

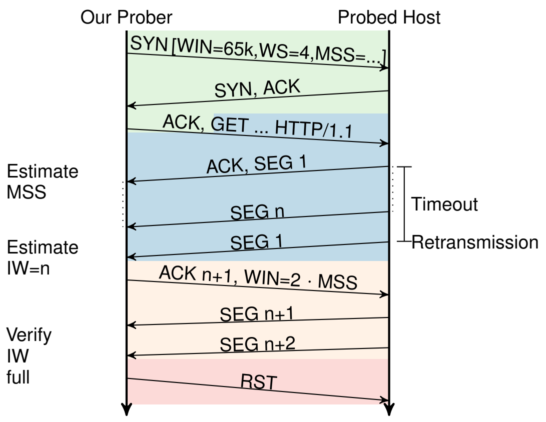

Fig. 3: Scan procedure: MSS and large receive window are announced and no ACKs are sent until a retransmission. The estimated IW is the # bytes received before the retransmission.

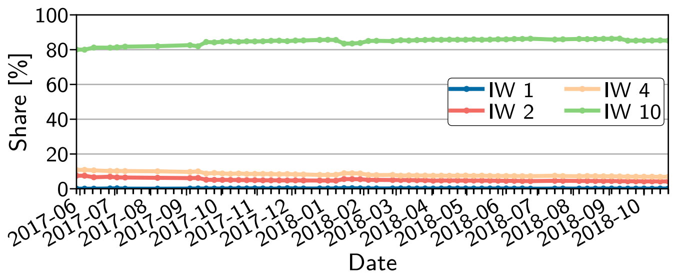

Fig. 4: Evolution of TLS server IWs in Alexa Top 1M since June 2017. HTTP evolution similar.

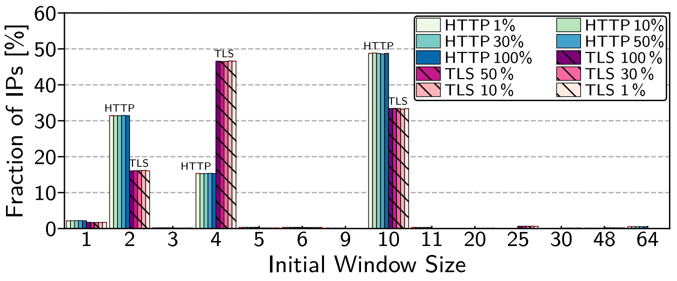

Fig. 5: IPv4 IW scan taken from [15]. Results from August 2017 indicate that 1% random samples closely estimate the overall IW distribution.

measuring CDNs, we extend our scheme to also account for virtualization by incorporating per-CDN target URLs and hostnames. This is needed to fetch large content from CDNs for IW estimation. We next describe its general procedure and details that we modify to account for CDN properties.

The IW estimation procedure (visualized in Figure 3) can be split into four phases: *First*, a regular TCP handshake is performed announcing the largest possible receive window of 65 kB and, to account for overly large IWs configured at some CDNs, also a window scaling option (shifting the window by 3) to never block the data transmission due to flow control. Since IWs can be configured depending on the MSS, we additionally set a MSS (varied by the later measurements). No further options like Selective Acknowledgements, which can for, e.g., cause TCP tail-loss probes that would challenge IW estimation, are activated. After establishing the TCP connection, the *second* phase starts by transmitting an HTTP GET request in hope to trigger a response that exceeds the configured IW at the probed host. The probed host will now commence sending the requested resource, however, we are not going to generate acknowledgments for any segment that we receive, thus the initial congestion window will not increase and the host can only send as many bytes as the IW. By not acknowledging segments, the probed host will eventually initiate a retransmission of the first (from its point of view) lost segment, which heralds the start of the *third* phase. Either, the sending host was in fact limited by the IW or it ran out of data to send. To test for this, we start acknowledging the last segment enabling the host to continue sending data and if the host does so we know that the host did not run out of data. At this point, we are able to estimate the IW by observing the sequence number space and segment sizes that we received before the retransmission. Finally, the *last* phase consists of tearing down the connection with TCP's RST mechanism. As this IW estimation methodology fails when tail-loss occurs (i.e., loss of the last packet in IW), we recommend to perform multiple scans of the same host.

Implementation & Validation. We implement our approach in go-lang (source available at [28]) to benefit from its multiprocessing capabilities and also reuse our ZMap-based scanner from [15] (source available at [29]). To test both implementations and to validate the correctness of the IW estimation, we run them in a mininet [30]. We use iptable's statistic module to drop packets at the head, within, and at the tail of the IW to validate the estimation correctness, i.e., correct estimations for the first two, and a reduced IW for the latter case (tail-loss). Further, we vary the IW size and available bytes on the server-side and the announced MSS at the probing client to validate non-standard IWs and out-of-data situations in various settings. Our tools always estimated the IW correctly except for tail-loss (as expected).

B. *Measuring IWs in IPv4 and Alexa Top 1M*

To estimate how IETF-recommended values find adoption in the Internet at large, we perform regular scans of HTTP and TLS servers in the IPv4 address space and in the Alexa Top 1M list using our ZMap scanner.

Alexa Top 1M. We start by investigating how IW configurations evolved in "popular" Internet infrastructure by looking at the Alexa Top 1M list. To do so, we have been scanning the hosts on the list roughly every week since June 2017 enabling us to investigate how IW configurations have changed.

To this end, Figure 4 shows the evolution of IW configurations at the example of TLS hosts (HTTP is similar) in the Alexa list. As the figure shows, the current IETF-recommended value of 10 segments prevails and is slowly but steadily increasing. In contrast, the legacy values of IW1, IW2, and IW4 diminish in favor for IW10. We observe little ditches and jumps in our data which we believe is due to the nature of the list having churn [31].

IPv4. In [15], we found that scanning a 1% random sample of IPv4 is enough to gather a good estimate of IETF-recommended

IW sizes. Figure 5 shows the results of a full IPv4 scan and subsamples of the IW distribution. For multiple 1% random samples, we found that all of them are very close to the true 100% scan. Back then we already found that TLS and HTTP show significantly different distributions.

Driven by these findings and to lower the footprint of our scans onto the Internet, we scheduled regular 1% scans of IPv4 to track the IW evolution at large. Figure 6 (circles) shows the results for HTTP (tcp port 80) scans with an MSS of 64 byte. In contrast to our Alexa scans, we can see a steeper increase of IW10 hosts. At the end of May 2017, we find that ~54% of hosts utilize an IW of 10 segments, one year later, this has increased to ~67%. Similar to Alexa, the increase of IW10 results in a steady decline of IW1, IW2, and IW4.

Looking at TLS, things continue to look differently as indicated in Figure 6 (triangles). At the beginning of our observations, IW10 has a lower share than IW4. However, in the beginning of 2018, this changes and we observe more IPs using IW10 in combination with TLS. We believe that these large-scale observations are mainly driven by adoption of more recent operating systems that default to the current recommended value of IW10.

Limitations. While these ZMap-based IPv4 scans are useful to investigate IW configurations at large, they have an inherent disadvantage. Given the nature of ZMap to enumerate IP addresses, these scans are unable to penetrate through any kind of virtualization or utilize additional information. So, for example in [15] we found that 47.6% (13.3%) of hosts do not offer enough data for an IW estimation when using HTTP (TLS). Further, HTTP can be virtualized via the `Host`-header, i.e., when sending a request to a server, the server can deliver different content subject to this header, so thereby, a single IP can serve multiple websites. This is a challenge for the scanner since it does only have an IP and not a fully qualified hostname, on the one hand, this can result in the IP not delivering enough data for an IW estimation (a challenge that we address in [15]) and, on the other hand, a server could adjust the IW depending on which website is requested. Similar virtualization exists in TLS through server name indication (SNI), similarly challenging an IW estimation. This is especially problematic in presence of CDNs that typically heavily virtualize their infrastructure to serve multiple customers with the same hardware. Thus, these scans are either not representative for CDNs or they heavily underestimate IWs when surfing the web. To this end, we next focus on how to estimate IWs at CDNs.

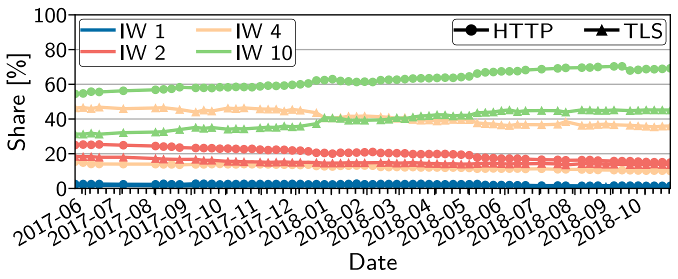

Fig. 6: Evolution of IETF-recommended TCP IWs in IPv4 since June 2017 for HTTP (circles) and TLS (triangles) established through periodic 1% random samples.

C. Measuring CDN IWs

This part of our measurement study is structured into three phases. At first, we gather lists of target URLs that are served by CDNs. In a second phase we derive the initial windows of the hosts serving the URLs. At last, we use VPNs to derive the IWs from different networks for a subset of these URLs.

Target Addresses. The first phase is relatively straightforward, we utilize data published by the HTTP Archive [32]. The HTTP Archive crawls websites while recording diverse information about the websites, for their bi-monthly crawls they visit all websites included in the Alexa Top 1M list. We utilize the crawl data from the 15th of January 2018 and extract all URLs that are loaded during the crawl, i.e., the landing page URLs as well as subsequently requested objects such as images or Javascripts. Even though the HTTP Archive already marks CDNs in their data, we repeat this step as the CDN choice could be geo-location dependent and as the HTTP Archive data can be up to half a month old the CDN operator could have changed in the meantime. To do so, we apply the domain list [33] published by the WebPagetest [34] framework (the framework driving the HTTP Archive), which enables to classify a URL by resolving its domain using DNS. Many CDNs utilize the DNS to redirect (using CNAME records) a user to the CDN server. Thus a CDN can be identified by its CNAME pattern in the DNS resolution step. The result is a rather large list of URLs which we filter to include only URLs hosted at CDNs and only one URL per domain. For each domain, we choose the URL with the largest object size. This results in a list of $\approx$ 227k URLs (available at [28]) hosted on 69 CDNs used to establish initial windows. 116K objects (25 CDNs) that are too small to reliably estimate an IW (for large segment sizes, see Section IV) are removed from the results.

Scanning Architecture. We use the architecture depicted in Figure 7 to structure and perform our scans. To enable concurrent scanning at multiple vantage points, we make use of OpenVPN and Linux's network namespaces. A network namespace can be seen as a shallow copy of the network stack with its own interfaces and routing tables. As many VPNs apply Network Address Translation (NAT) to assign IP addresses to their peers, we experienced that different VPNs assigned the same IP or the same subnet to us. To overcome this issue, we override OpenVPN's device creation and insert a script to manually create network devices in a new network namespace identified by the VPNs publicly facing IP. This enables to completely disregard any routing or name clash issues when using multiple VPNs in parallel. We then start one instance of an IW-prober in each namespace and feed it with the URLs.

To not put a large burden on the VPNs, we perform a preprocessing step. Instead of querying all 111k URLs (potentially multiple times to account for tail losses) through the VPNs we first derive a list of candidate URLs in our campus network. We select query candidates by grouping URLs hosted at the same CDN using the same IW and select a random sample of URLs for each (CDN, IW) pair.

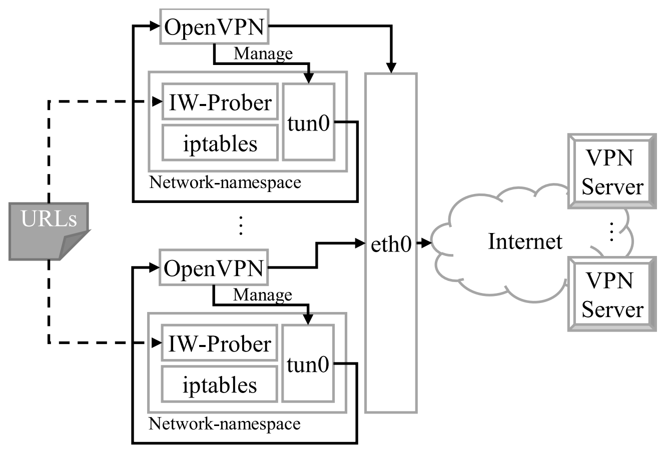

Fig. 7: Overview of our scanning architecture. We leverage Linux network namespaces to scan concurrently and to easily manage multiple VPN connections.

Vantage Points. Gathering globally distributed vantage points that grant packet-level access is hard. To do so for our measurements, we make use of the VPN Gate [35] project by the University of Tsukuba. This project's goal is to give access to the Internet without censorship. To this end, the project manages a list of thousands of relay VPN servers around the globe, many of which are operated by volunteers via their private Internet uplinks. While the site lists many VPNs, we found only a small set of them to reliably work for our measurements which might also be due to the implemented censorship protection announcing false gateway servers. To account for our scan methodology, we only use VPN connections made through TCP, thus loss between our VPN client and the VPN server is automatically resolved and is not accounted as loss for our prober. According to [35], most of the VPN servers only have a relatively small bandwidth capacity mostly below 10 MBit/s. Consequently, to not disturb the regular VPN operation, we implement a rate shaper into our prober that smooths burst and limits the outgoing bandwidth. We configure it to transmit at most 100 packets per second, thus we limit the prober to ≈1.2 Mbit/s for full-sized frames and much less for smaller frames. Further, through local experiments we found that excessively parallelizing IW estimations challenge NATs easily causing exhausted NAT tables, therefore, we limit ourselves to a handful of parallel estimations per VPN. We chose both, the parallelism and the rate shaper, such that only short bursts are shaped and no long standing queues are formed that could impact our measurements.

IV. Campus Network Perspective on CDN IWs

We next explore CDN IW configurations from the perspective of a well-provisioned campus network (RWTH Aachen University) (worldwide perspective follows in Section V) to set an upper bound on the expected IW sizes. As our network's upstream ISP peers with DE-CIX (at which many CDNs peer as well) and our network offers at least one order of magnitude higher capacity than typical consumer Internet connections, CDNs could potentially adapt by serving content with higher IWs thus providing an upper bound on the expected IW sizes.

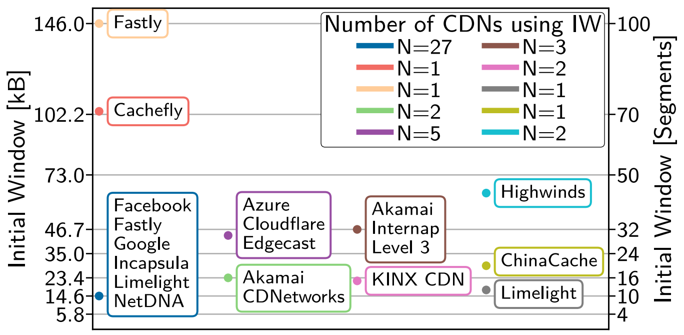

Fig. 8: CDN IWs as seen from our university network. IETF sizes 4 (not shown) and 10 are present but also larger IWs.

IW Probe Procedure. As IWs can be configured in bytes or segments, we scan each URL (see Section III-C) with different maximum segment sizes of 64, 128, 536, 1200 bytes, ten times each. This enables to derive if the scanned host changes the total number of bytes delivered in the IW, i.e., the IW is fixed to a certain number of packets (we refer to the segments) or if it is fixed to a certain amount of bytes (we refer to the bytes). To account for tail-loss, we perform a majority vote for each segment size and regard a scan as successful if > 50 % of the votes agree on the largest observed IW (97% of measurements). To derive the final IW, we inspect the number of packets and bytes received over the four different segment sizes: if the IW depends on the segment size, we calculate an IW (in bytes) as if we were using maximum-sized segments (1460 byte), otherwise, we directly use the fixed amount of bytes. Note that we refrain from showing quantities in which we observed certain IWs as they could be biased by the choice of URLs. Furthermore, we are not able to estimate IWs for all URLs, since their object size can simply be too small to fill a larger IW. This would bias the results towards smaller IWs.

A. *IW Sizes*

Figure 8 shows the resulting IW sizes in bytes and segments assuming 1460 byte packets from our local campus network. Each dot represents an IW configuration, the adjacent box lists a selection of CDN providers that deliver URLs with this IW (a CDN can occur in multiple boxes). Even though we find many CDNs offering URLs via IW10, we also find much larger sizes. This is in contrast to our prior IP only scans over the IPv4 address space (see Section III-B and [15]), which found IETF-recommended IW sizes to dominate most likely due to the number of deployed legacy systems (e.g., DSL gateways).

Our findings show that CDNs do in fact depart from IETF-recommended IW sizes and customize the IW. For example, we observe IW16 and IW32 for the probed Akamai URLs[1], both larger than the current IETF recommendation of IW10. However, we also find very large IWs. For example, the largest IW that we observed is by Fastly, they deliver *some* URLs using an IW of 100 segments. Cachefly also shows a larger than usual IW of 105 kB, notably, Cachefly uses a fixed IW

[1]We remark that each CDN can use *additional* IW configurations beyond the configurations discovered in our measurements.

Operating System	RWIN [B]	WS	WIN [B]	Segs.
Linux 4.4	58	512	29.696	20
Android 6.0 (Linux 3.4)	685	128	87.680	60
Android 7.0 (Linux 3.18)	641	128	82.048	56
iOS 11.2.5	2.058	64	131.712	90
Mac OS 10.9.5	8.235	16	131.760	90
Mac OS 10.13.2	4.117	32	131.744	90
Windows 7 (SP 1)	256	256	65.536	44
Windows 8.1 / 10	1.024	256	262.144	179

TABLE I: TCP receive window (RWIN), window scaling (WS), resulting window (WIN) in bytes and full-sized segments on different operating systems as reported on an HTTP GET request from an otherwise idling system.

configured in bytes which leads to many transmitted segments when small segment sizes are used. On the opposite end of the spectrum, we find URLs hosted on CDNs that deliver data with a smaller IW than currently recommended. For example, we find URLs hosted on ChinaCache (not shown) that are delivered with an IW of 4, yet, we can again observe that ChinaCache customizes as well, as they also deliver URLs with IW20.

Can Increased IWs be Utilized? The actual amount of data that is transported is of course not only dependent on the server's congestion window. The client permanently announces a receive window (RWIN), TCP demands that no more than the minimum of the advertised RWIN and the congestion window is in flight. Table I shows the client's advertised receive window on an HTTP GET request for a selection of client operating systems. As the table highlights, the largest IWs that we measured would not be effective for a couple of operating systems. Linux 4.4 shows the lowest advertised receive window which would not be able to utilize many of our discovered CDN IWs. We found a git commit [36] documenting this receive window in response to the IW10 increase. Interestingly, Android, even though using an older Linux kernel, has increased the receive window and would be able to utilize most of the IWs measured, the same holds for iOS and all other tested Mac OS variants. Apart from Windows 7, all recent Windows variants announce receive windows large enough to not thwart even the largest observed IW.

Takeaway. *We observe CDNs to configure IW sizes beyond IETF recommended values, highlighting that i) Internet reality departed from standardization and ii) IW sizes larger than standardized are of practical relevance. Their actual impact on network performance, in terms of losses, fairness and flow completion is practically unexplored by current research, highlighting that Internet reality also departed from research. We found that CDNs do customize IWs, however, it remains unclear when a CDN decides to utilize which IW.*

B. Are IWs Content Dependent?

One way to customize IW sizes is by delivery service class (e.g., low latency web delivery vs. elastic download), which can explain multiple observed IWs per CDN. Since we cannot directly identify service classes, we analyze IWs for typical *content types* by filtering the HTTP Archive for Akamai-served URLs according to their mime type. We focus on Akamai, as one of the largest CDNs for which we already observed multiple IW sizes. For each domain, we take the largest URL of the following mime types: *i)* `application/mp2t` (62 URLs) typically employed for streamed video streaming applications, *ii)* `image/png` (1812 URLs) for images, *iii)* `application/javascript` (1395 URLs) for regular website content, and *iv)* `application/octet-stream` (67 URLs) for any binary data (download). We expect that interactive content uses the larger of the two initial windows as e.g., the play-out of a video should start as fast as possible.

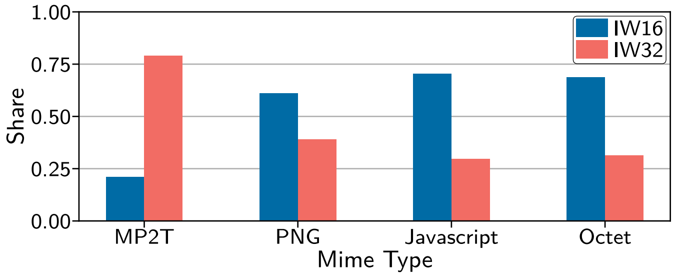

Fig. 9: IW distribution for Akamai URLs per mime type.

Figure 9 visualizes the analysis. Our expectations are partially met, i.e., streamed video content (MP2T) is in fact mostly delivered with IW32, yet not exclusively. This and also the other mime types highlight that the mime type does not determine the initial window per se. For PNGs, Javascripts, and binary data we observe that the majority is served via IW16, the quantity of IW32 varies between 30 % (Javascript, binary) and 40 % (PNG). These observations highlight that it is more likely that an IW is not set depending on the mime type but is rather dependent on the service class (product) that has been purchased at the CDN. Of course, some products are designed for interactive delivery and others not, yet, in the end, this non-strict assignment of IWs to mime type shows that the customers decide what they deliver through which product.

Takeaway. *Different content types can benefit from different IW sizes and our results suggest that content dependent customizations (e.g., for interactive video streaming) exist. Yet, they cannot be purely detected by mime type since they rather depend on the delivery strategy selected at the CDN.*

V. Worldwide Perspective on CDN IWs

To investigate if CDNs tailor IWs to networks, we probe the same URL from multiple vantage points. To do so, we utilize the public VPNs listed at VPNGate [35]. As the service lists thousands of VPNs, we concentrate on a small subset of 14 VPNs all located in different countries and ASs. For these VPNs, we test samples of URLs (5 per IW/CDN combination) for which we have already established an IW locally, thus enabling to compare if other networks are subject to different IW configurations.

Table II gives an overview of the VPN locations (as reported by VPN Gate), networks as well as a manual classification of their link's nature. We classified the link type by inspecting *i)* the AS and *ii)* the reverse DNS name of the VPN host and check if it includes keywords such as: cable, (A)DSL, dynamic, etc. Most of our VPNs are located in residential

#VPN	AS	AS Name	Country	Link Type
1	AS1221	Telstra	Australia	Consumer
2	AS3303	Swisscom	Switzerland	Consumer
3	AS3326	Datagroup	Ukraine	?
4	AS4766	Korea Telecom	South Korea	?
5	AS7552	Viettel	Vietnam	Consumer
6	AS7922	Comcast	USA	Consumer
7	AS9198	Kaztelecom	Kazakhstan	Consumer
8	AS12389	Rostelecom	Russia	Consumer
9	AS16276	OVH	France	Datacenter
10	AS17534	NSK	Japan	?
11	AS24560	Airtel	India	Consumer
12	AS24620	Riga Tech. Univ.	Latvia	University
13	AS28548	Cablevisión	Mexico	Consumer
14	AS28885	OmanTel	Oman	Consumer

TABLE II: Classification of VPNs used to estimate CDN IW configurations.

VPNs	Akamai		Azure	Cachefly	Cloudf.	Edgecast	Fastly		Highw.	Level 3
	16	32	30	105 kB	25	30	100	10	64 kB	32
1,9	1	1	1	1	1	1	1	1	1	1
2-6,8	16	32	30	105kB	25	30	61-62	10	64kB	32
7	16	32	20-30	105kB	25	20-30	61-63	10	20-60kB	32
10	16	32	30	105kB	25	30	1-100	10	83kB	32
11	16	32	30	105kB*	25	15-30	2-61	10	4-49kB	32
12	16	32	30	105kB	25	30	87-99	10	64kB	32
13	16	32	6-30	105kB	25	2-30	6-73	10	64kB	32
14	16	32	30	105kB	25	30	61-75	10	58kB	32

TABLE III: IW configurations observed at the VPNs. The top row shows the CDNs with their IWs as discovered within our campus network. Each field marks the IW we discovered through the VPN or a range if we saw consistent losses. Results marked with (*) experienced packet loss but no tail-loss.

access networks, with the exception of one datacenter (#9), one university network (#12) and three links (#3, #4, #10) that could not be classified due to missing hints.

Table III summarizes our IW estimations through these VPNs. We were able to build classes of VPNs that perform similarly, already indicating that many of our VPNs show a similar performance and we see a similar IW configuration. The first class for VPNs #1 and #9 show the largest divergence from our campus network. Here we measured an IW of only 1 segment for all CDNs contacted via both the consumer (#1) and the datacenter (#9) link. Especially for a datacenter link this seems too low and does not fit the rest of our data. When more closely inspecting both VPNs, we found that both VPNs seem to heavily rate-limit the packet-rate. Even when performing a regular download of the URLs, we are unable to ever get more than two segments in a roundtrip at any time. Thus, we believe that the IW estimation here is unable to determine the actual IW due to the rate-limiting which highlights the challenges when using vantage points that are out of direct control.

The second, largest class, of VPNs, paints a similar picture to that of our local observations. We observe for all but one CDN provider the same IWs as seen from our university network. The only difference being Fastly, for which we have measured IWs between 61-62, we consistently measured IWs in that range indicating that our measurements are subject to loss. We take this as an indication that it is likely that 62 is not the actual IW that should have been delivered but rather a larger IW was subject to heavy tail-loss, especially since all other IWs are configured similarly to our local observations.

This impression continues when observing the remaining VPNs, there the IWs for Fastly also reach up to 100 segments (VPNs #10 and #12) but with consistent losses between multiple measurements. For many, we observe IWs in the range of 60 to 70 segments. We take this as an indication of a service specific configuration rather than a network-dependent one.

But we also find patterns of network-dependent configuration, e.g., for the Highwinds CDN that we measured with an IW of 64kB locally. For VPNs #10 and #14, we consistently observe different IWs. For VPN #10 we observe a larger IW of 83 kB and for #14 only 58 kB. Yet, also for Highwinds, we can observe losses at VPN #7 and #11.

Especially, VPN #11 observes the highest losses throughout our measurements. Here, also Cachefly with the second largest IW (equaling to 72 full-sized segments) that we observed shows losses (which does not show any losses at other VPN).

Limitations. Even though it is likely that we see signs of network-driven tailored behavior, we could be hitting legacy systems that are just differently configured. Furthermore, since we use public VPNs, other requests could affect our IW-estimations that were done over this VPN. However, this would actually strengthen our observations that CDNs in-fact do tailor.

Takeaway. *Overall, we observe that many CDNs use the same IWs regardless of the network and are successful in delivering it without losses. Interestingly, we find that Level 3 and Akamai both deliver content without loss using IW32, while others like Edgecast and Azure experience loss over the same links despite using a smaller IW of 30.*

Motivated by these observed losses, we want to investigate the burstiness of IWs. To this end, a recent proposal [14] recommends using TCP pacing to evenly space out packet delivery over the RTT when exceeding an IW of 10 segments to be less aggressive towards queues. This has also been proposed in [37] after idle slow-start restarts. Since Linux Kernel 3.11 (released in September 2013), it offers pacing support via a special packet scheduler in the traffic control (TC)-subsystem, starting with Linux Kernel 4.13 (released in September 2017) also directly from within the TCP stack. Thus, we continue to investigate the temporal characteristics of the packets transmitted in the IW to investigate the use of pacing at CDNs.

VI. Burstiness of the CDN IWs

To investigate the use of TCP pacing by CDNs we again focus on our university network as we require fine-grained packet arrival times which are not preserved through the VPNs.

Traffic Shape of the Linux Pacer. Linux implements pacing in TCP either via TC as a queuing discipline or directly from within the TCP stack. For TC, the default queuing discipline must be exchanged to use the TC Fair Queuing (FQ) discipline. TCP interacts with FQ by setting an appropriate pacing rate on the socket that is used by FQ to spread out packets. TCP re-calculates the pacing rate with every incoming ACK and after the initial handshake as $\frac{MSS \cdot CWND}{sRTT} \cdot RATIO$

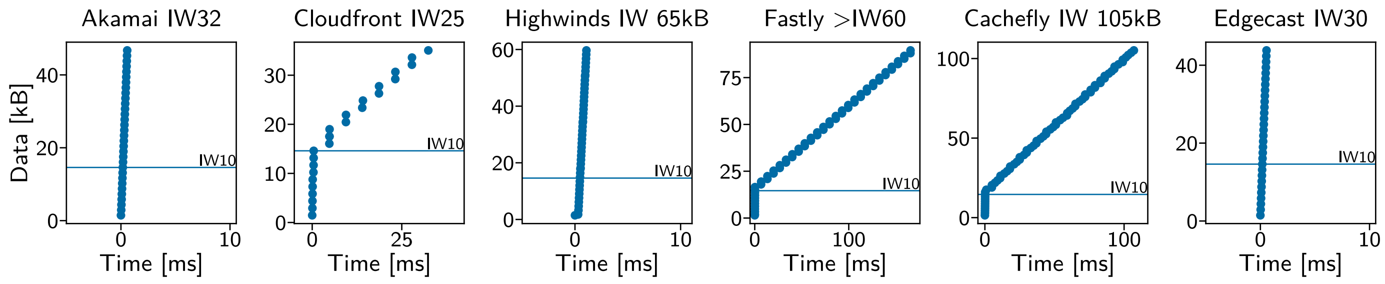

Fig. 10: IW burstiness for a subset of the observed CDNs and URLs (each with an RTT of 60 ms-70 ms). The arrival time of full-sized 1500 byte packets (dots) in the entire IW is shown on the x-axis, the IW size (in kB) on the y-axis (e.g., IW10 = 15 kB). Note different axis scalings due to different IW sizes. Some CDNs seem to utilize packet pacing while others do not.

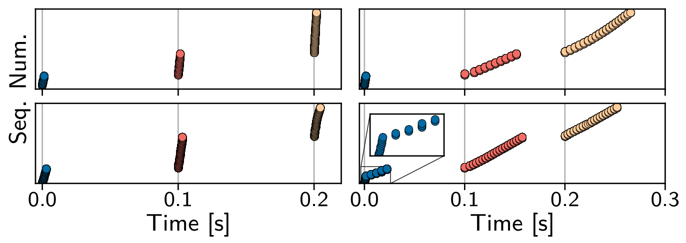

Fig. 11: Linux default traffic pattern on the left, pacing using FQ on the right. Each marker denotes one packet. Packets arriving in the same roundtrip have the same color. Top row uses IW10, bottom IW20.

with maximum segment size (MSS), the congestion window (CWND) in segments, the smoothed roundtrip time (sRTT) and a RATIO defaulting either to 200% in slow start or 120% during congestion avoidance. The ratio is used to accelerate the pacer in different connection phases and can be adjusted via sysctl parameters for both slow start and congestion avoidance. The pacing rate defines the amount of data that can be sent and the pacer uses this rate to calculate the departure of the next packet such that the rate is enforced on the connection. Yet, in addition to enforcing a bandwidth, the pacer also allows for a certain burstiness of the traffic. This is enabled by combining the previously discussed rate with a token bucket scheme, where every flow is assigned a *credit* that is initialized by an *initial quantum*, and every time a packet is dequeued and the credit is zero or below, the packet is enqueued again and the credit is increased by a *refill quantum*. Thus, new flows, i.e., at the start of the connection, may directly burst the initial quantum and are then subject to the rate limiter and will continue sending bursts of packets governed by the refill quantum (assuming enough data is available for dequeuing).

Figure 11 shows the resulting traffic pattern with the pacer's default values for the initial quantum, i.e., 10x MSS, and the refill quantum, i.e., 2x MSS, for a server serving data at the start of the connection with IW10 and IW20. We used *netem* to add a delay of 100 ms at the egress of the client to simulate an RTT of roughly 100 ms in our local network and then measured the packet arrival. As we found that the packet coalescing of the NIC, which reduces the interrupt-rate, causes imprecise software timestamps of the arriving packets, we instruct our NIC to perform hardware timestamping at packet arrival. To illustrate the difference the left side of the figure shows the regular, bursty behavior without packet pacing. As we can see on the right side, the default pacer parameters are chosen in a way to allow bursts of 10 packets, thus allowing an IW of 10 segments (i.e., the IETF-recommended value) to seamlessly pass through the pacer, while bytes larger than IW10 are subject to pacing as visible in the lower right plot that uses IW20. These trains of two packets (refill quantum) correspond to TCP's self clocking [22] behavior when having delayed ACKs that generate an ACK for every second segment thus causing the release of two packets with every incoming ACK. We next empirically probe CDNs for this pattern to detect pacing.

Measuring the Packet-Pacing. To measure if the CDNs utilize pacing, we take a look at the packet arrival-times when executing an IW scan. To do so, we simply record packet traces (with tcpdump) but instruct our IW-prober to delay the ACK following the SYN/ACK by roughly 50 ms to emulate a larger RTT to the measured CDN. We again use hardware timestamps for precise time keeping.

Similar to Figure 11, Figure 10 shows all packets (dots) sent during one initial window after connection start for a selected subset of CDNs and URLs. Their arrival time is depicted on the x-axis and the IW size in kB on the y-axis. Please note the different x- and y-axis scaling due to the different IW sizes. We can visually observe two different patterns. The first, here presented by Akamai, Highwinds, and Edgecast, shows close to no temporal distribution of packets. The second, presented by Cloudfront, Fastly, and Cachefly, shows a stream of packets arriving virtually at the same time followed by a temporally skewed train of other packets. The latter follows the expected output of Linux's packet pacer as described before (Figure 11). This is best visible in the example of Cloudfront, where a burst of ten segments is almost perfectly followed by delayed trains of two packets. Thus, we can see that some CDNs are likely utilizing pacing during slow start for IWs larger than IW10 as recommended in [14].

IW-Skew by Pacing Rate. When looking at the two largest IWs that we observed by Cachefly and Fastly[2], we can see that both pace their IWs, however, we observe that Cachefly is more aggressive in doing so as they spread their IW over roughly 1.5x the RTT while Fastly does it over roughly 2.5x the RTT. Yet, compared to smaller IWs like observed at Cloudfront that

[2]Please note that while measuring pacing, we experienced heavy tail-loss with Fastly leading to the reduced bytes.

pace the IW over roughly 0.5x RTT (i.e., very close to the Linux pacer's default), it might not be obvious why spreading the data transmission over more than an RTT would make sense. We speculate it could be favorable in situations where the initial RTT sample from the 3-way handshake is a bad estimate for the RTT or when there is congestion on the reverse path. When the initial RTT is lower than the RTT that is to be expected during transmission, e.g., when there is other traffic filling queues or one must compete for airtime in wireless setting, it might take more time for ACKs to arrive than the initial RTT sample predicted. Similarly, when there is loss on the reverse path, ACKs might not arrive rendering the connection idle, thus, in both cases prolonging the IW transmission duration *could* be favorable.

Yet, this kind of pacing challenges the traditional notion of IWs. Typically, the IW is thought of as the number of bytes sent in the first RTT without requiring an acknowledgement. Since traditional TCP is bursty, pacing skews this notion as the data must not necessarily arrive within the same RTT as we already observe for some CDNs. Thus, e.g., when an IW of 100 segments is paced over 2 RTTs, the bytes arriving at the receiver in the first RTT are effectively half of what is received with a bursty IW100. Therefore, depending on the pacing rate, a paced IW100 might better compare to a bursty IW50 or even less. Given this observation, discussing IWs and simply looking at the number of segments is insufficient to reason about its appropriateness. IWs should be regarded with respect to time and rate, e.g., an IW of 100 segments paced over 2 RTTs with an RTT of 30 ms corresponds to a rate of ~20 MBit/s which seems reasonable when looking at the capacity of current user access speeds. While this does not capture the rate on a sub-RTT level, e.g., when no pacing is used, a data rate better captures the demand of a new connection on the network than a fixed number of segments.

Takeaway. *We find it is likely that pacing is used by some CDNs. In fact, the two largest IWs show clear pacing patterns. Past research suggests that pacing can help to bootstrap new or idle connections, however, there is currently only a limited understanding of the impact of pacing on networks and of its benefits and drawbacks especially as current pacers deviate from perfectly paced packet streams found in literature. Additionally, pacing challenges the way IW values should be regarded, we find a data rate to better capture the demand on a network.*

VII. IW Performance when Competing for Traffic

While we initially investigated the theoretical advantages and disadvantages of using larger IWs (Section II-A), these measurements were idealized. Further, our previous CDN measurements have shown that some CDNs utilize pacing to distribute the initial load on the network over a longer period. We take these observations and investigate how the parameters affect performance in a more realistic setting, i.e., when having to compete for bandwidth. Even though, we are not the first to investigate the performance impact of larger IWs, other studies often use simulations and do not rely on the actual code and configurations that we observed in the wild. Yet, there are still many possibilities to investigate this, e.g., how do the parameters affect a congested peering link or how does it affect performance in a data center, we opt to investigate the effects from a user's perspective. Even though it is known that there are instances of persistent inter-domain congestion [38], it is still widely believed that a lot of congestion happens at the network edges and more specifically at the end-user's access link [39], i.e., the last mile. We use this setting and investigate the performance on an emulated link where a new, comparably short flow must compete against an elephant flow, i.e., a bulk transfer. This reflects a typical situation at home with a shared access link, i.e., a bulk download competes against web traffic. We start our evaluation by describing our testbed before continuing on with discussing our results.

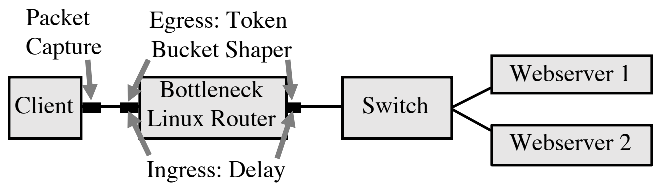

Fig. 12: Testbed topology with the client requesting traffic from the two webservers. Bottleneck characteristics are configured on the dedicated bottleneck machine. Packet traces are captured on the client machine.

A. Testbed and Parameterization

We extend the testbed used in Section II-A by implementing a simple dumbbell topology, illustrated in Figure 12. It allows us to reflect our end-user setting and enables us to adequately investigate congestion control [40]. For our measurements, the client machine to the left of the figure requests a high volume elephant flow from Webserver 2. After it has reached the bottleneck's capacity, the client requests a second, small volume flow from Webserver 1.

We again vary between four different bandwidth and three different delay configurations motivated by Akamai's report [17]. Even though CDNs in general strive for low RTTs of a couple of 10th of ms, wireless links and areas with suboptimal CDN coverage may still face higher RTTs. Additionally, we size the bottleneck's queue in accordance to the bandwidth delay product (BDP) rule of thumb. Since, we know of no studies that investigate how router buffers are sized in the Internet, especially at the edge, we use 0.5x BDP, the BDP itself, and 1.5x the BDP.

We configure the bandwidth and queue size using a token bucket filter with a burst size of a single frame at our Linux-based bottleneck machine while using a traditional drop-tail FiFo queue. Even though Internet access links are often asymmetrical, we disregard this fact as we are not interested in investigating reverse path congestion and use the same bandwidth in both directions.

To add delay to our testbed, we modify our bottleneck's ingress packet processing. There, we artificially redirect traffic to an intermediate queue disc enabling us to use NetEm to add delay before we release the packet for forwarding to the actual

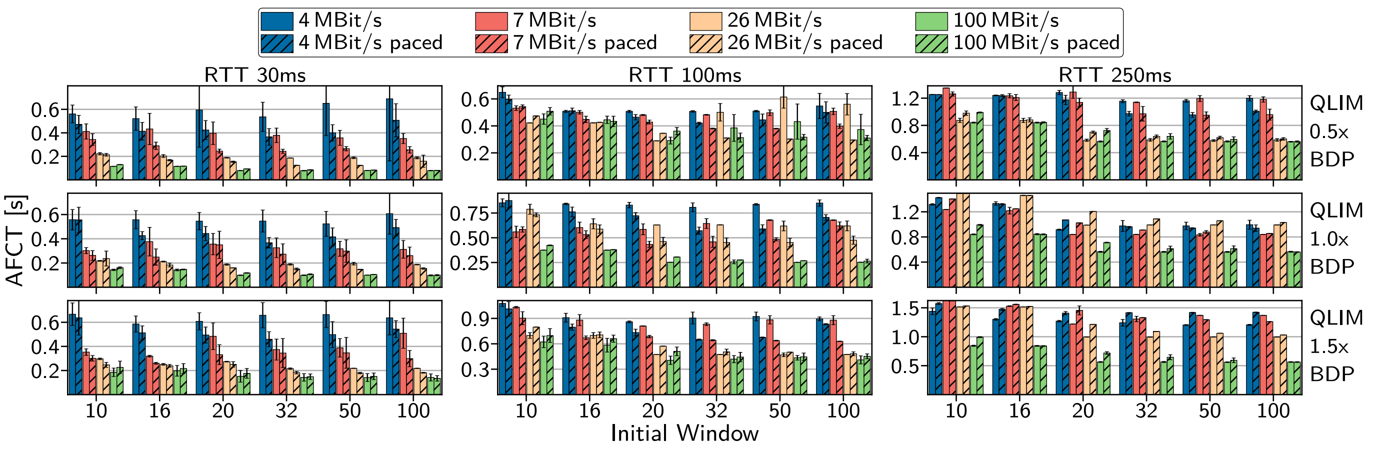

Fig. 13: Average flow completion time for a 73 kB Cubic flow (solid) and a paced Cubic flow (hatched) when competing against a Cubic elephant flow for different RTTs (columns) and different queue sizes (rows) subject to bottleneck bandwidths (colors) when using different IWs (x-axis), errors bars denote 95% confidence intervals over 30 measurements.

egress queue. While care needs to be taken to size the NetEm queue to not cause artificial packet loss this approach has the advantage that the end-host stacks are not involved in the delay which is known to badly interfere with congestion control when Linux detects queuing pressure (TCP small queues). To have a symmetric delay, we add half of the configured delay to each ingress of the bottleneck. We do not configure any artificial jitter using NetEm as this causes packet reordering, the delay and jitter are thus only caused by the egress queue.

All our machines are connected via Gigabit Ethernet and use a Linux 4.13 kernel, we further make sure that initial receive windows are large enough to not limit the IW (see Section IV-A). Additionally, we clear all TCP metrics after each measurement and make sure that send and receive buffers are sizes such that the machines can fully utilize their Gigabit Ethernet connection.

We capture traffic at the client using tcpdump to compute the average flow completion time (AFCT) of the short flow.

B. Increasing Cubic IWs and Applying Pacing

Our first study investigates the advantages when utilizing increased IWs as observed in our CDN measurements and the additional implications that come with pacing. To this end, we investigate how Cubic, the Linux default congestion control algorithm, performs subject to larger IWs and pacing. Slightly different to our initial investigations, we use 73 kB of data for our short flow (to have slightly more than 50 frames) and this time our sole focus is on the flow completion time. After the elephant flow has started, the client requests the short flow which then competes for bandwidth and for which we measure the AFCT. We repeat each configuration 30 times to be able to statistically investigate the results. Figure 13 visualizes our study, each row showing the same measurements subject to different bottleneck queue sizes, each column for different RTT configurations.

Bursty TCP. We first focus on the non-paced performance (i.e., all non-hatched bars), starting with the 30 ms column reflecting a well connected CDN. Our first observation is best visible at 4 MBit/s, we observe that for this low-end bottleneck speed, increasing the IW results in decreased performance. On average the FCT increases with increasing the IW, especially when the queue size is small, yet regardless of the queue size, the stability decreases as indicated by the increasing confidence intervals. Thus, while some measurements show increased performance, some show extremely worse performance indicating the unsuitability of large IW for these network configurations.

Similarly, no real gains are observable for 7 MBit/s bottlenecks, only starting with 26 MBit/s, especially for larger queue sizes. Looking at larger RTTs things do slightly change, at 100 ms slight IW increases to 16 or 20 segments still yield performance increases even for low bandwidths. Yet, we observe that IWs such as 50 or 100 cause problems at 26 MBit/s and 100 MBit/s indicated by increased variance especially visible for short queue sizes. With an RTT of 250 ms, our measurements indicate that for small queues, again sufficient bandwidth is required to utilize larger IWs to not hurt performance. When increasing the queue size performance generally decreases as in all other settings, however, we see much clearer that the long flow hogs the queues leading to reduced performance even when having larger bandwidths.

Paced TCP. We next shift our focus to the performance of the paced Cubic short flow (i.e., all hatched bars). Looking at 30 ms RTT, we find that pacing, in comparison to its bursty counterpart, enables TCP to utilize an increased IW already for 4 MBit/s bottlenecks. First, the FCT, on average is generally lower than the bursty variant, second, at times where the IW starts to worsen the performance in the unpaced setting, it still improves the performance up to an IW of 32 segments. Until then, the FCT even compares to the unpaced variant with 7 MBit/s. After that, the performance again starts to worsen, notably with an IW of 100 segments (recall we are only transmitting slightly more than 50 frames), the performance notably worsens in comparison to the IW50 case. We believe this is due to the fact that TCP uses the IW of 100 segments to calculate the packet departure times even though much less is transmitted leading to a smaller inter-packet gap and overall

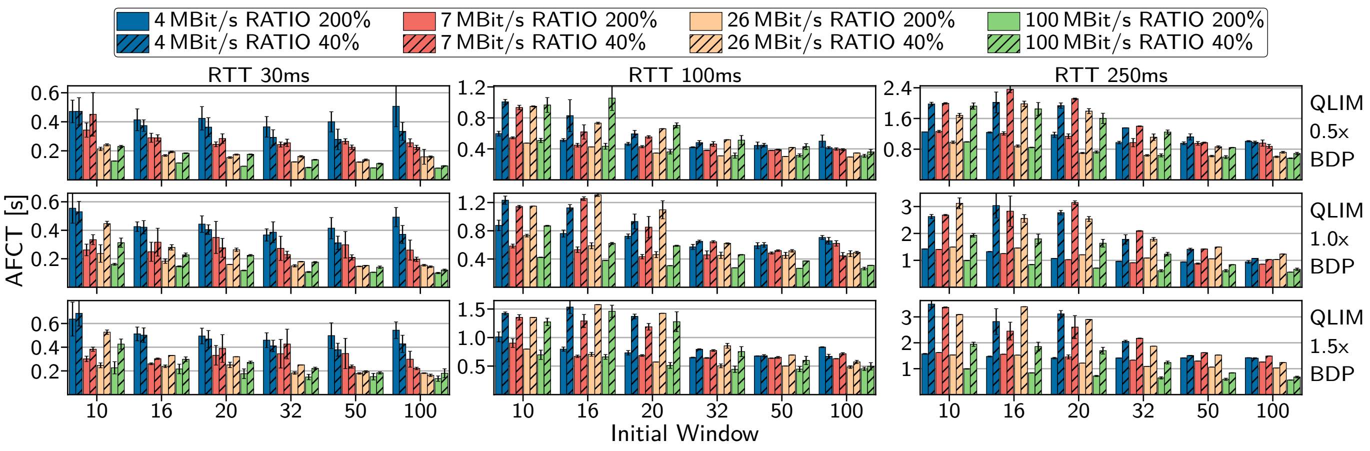

Fig. 14: Average flow completion time for a 73 kB paced Cubic flow with a slow start RATIO of 200% (solid) and a paced Cubic flow with a slow start RATIO of 40% (hatched) when competing against a Cubic elephant flow for different RTTs (columns) and different queue sizes (rows) subject to bottleneck bandwidths (colors) when using different IWs (x-axis), errors bars denote 95% confidence intervals over 30 measurements.

shorter time frame which seem to put too much pressure on our bottleneck. This observation carries over to larger bandwidths as well, however, not as drastically visible. For 100 MBit/s, we even see that the performance slightly worsens, this is not without surprise, the last packet will depart roughly half an RTT later in the paced variant. This especially makes a difference when there is less congestion. Pacing's advantages slightly diminish over larger buffer sizes, yet, in most cases it is still beneficial.

For larger RTTs of 100 ms and 250 ms, pacing does not always yield an improved performance. Especially, when having larger queue sizes that absorb bursty traffic (see 1.0x and 1.5x BDP at 250 ms), it seems that the competing elephant flow leaves enough buffer space during collision avoidance that the short flow can pass through without requiring pacing. This is in line with simulation results from related works that show pacing's benefits especially when having small buffers.

Takeaway. *Our investigations indicate that pacing enables TCP to utilize larger IWs when competing against an elephant flow. These advantages are especially apparent with small buffers and short RTTs while they diminish for larger bandwidths, because pacing, by its principle, worsens the flow completion time in uncongested settings.*

C. Pacing Aggressiveness in Slow Start

In our CDN observations about pacing, we already saw that some CDNs spread their IW over different fractions of the RTT. By default, as used in the previous section, Linux's pacer paces over 0.5 of the RTT by setting the RATIO to 200%. Since we already observed how larger IWs that are not utilized by application data affect the pacing performance in the previous section, we want to more thoroughly investigate how this *agressiveness* of the pacer affects the performance. To this end, Figure 14 compares our previous paced variant with a RATIO of 200% against one where we set the RATIO to 40%, i.e., such that the IW is paced over 2.5x the RTT, which was the largest spread observed in our previous measurements. For the 200% RATIO, we reuse the data from before (now non-hatched part of the plot).

Looking at the first column (30 ms RTT), we observe that spreading the IW over a larger time can yield a slight performance increase for low-bandwidth settings even though large IWs are used. We believe that the large spread leads to a later exit of slow start even though a large IW is used, ultimately shortening the FCT for this low bandwidth setting. When we look at larger bandwidths, we can see that the decreased pacing rate hurts performance especially for small IWs. Interestingly, the reduced pacing rate is beneficial when the IW is larger than the application data (IW 100 in this case), however, we think it is not the correct measure to reduce the aggressiveness of too large IWs when there is fewer application data. Rather, the pacer should be informed about application data limitations and use the available amount of data as the IW if it is smaller than a preconfigured IW.

Focusing on the larger RTTs, we can clearly observe that in nearly all tested settings the prolonged transmissions cause significant increases in the FCT. Only for IWs of 50 or 100 segments, the prolonged transmission time seems to nearly amortize. Thus, the ideal configuration of the pacer seems to be very dependent on the concrete application and scenario.

Takeaway. *We tested a very extreme acceleration change of 200% down to 40%. Although a smaller RATIO, hence longer duration, seems to be beneficial in some situations, it is disadvantageous in most of the investigated settings. Nevertheless, accelerating or decelerating the pacer in different settings or over the course of the connection seems to be an area worthwhile to further explore, a property inherent to the BBR congestion control.*

D. Increased IWs with BBR Congestion Control

For our final investigation of IW performance we switch the congestion control algorithm. Now, we utilize the recent BRR algorithm that is known to be used by Cloudflare and Google. Its design is build around the idea of pacing, however,

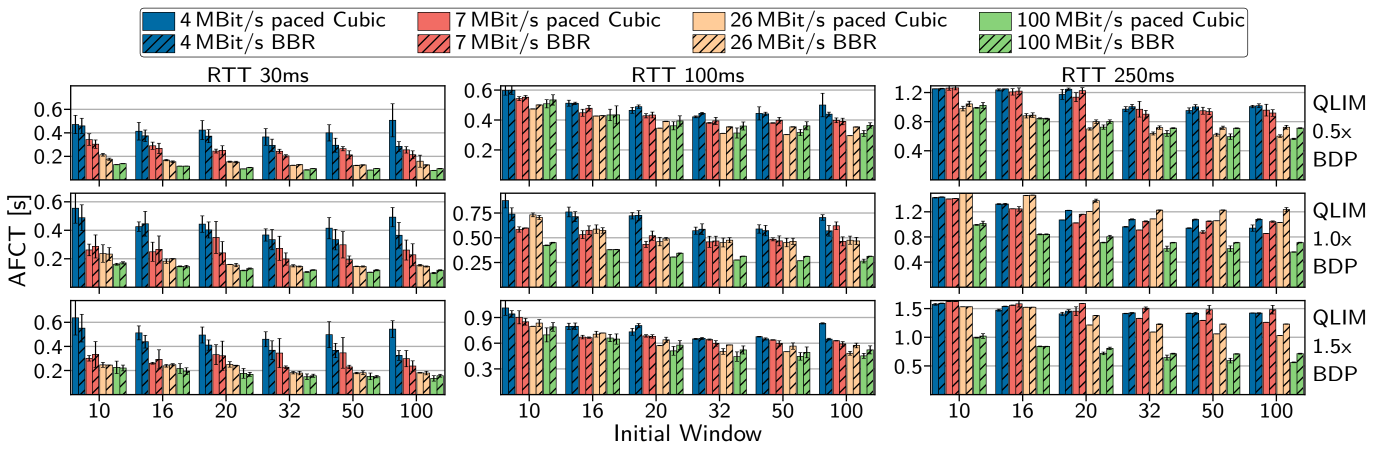

Fig. 15: Average flow completion time for a 73 kB paced Cubic flow with a slow start RATIO of 200% (solid) and a BBR flow (hatched) when competing against a Cubic elephant flow for different RTTs (columns) and different queue sizes (rows) subject to bottleneck bandwidths (colors) when using different IWs (x-axis), errors bars denote 95% confidence intervals over 30 measurements.

compared to our paced Cubic that we tested before we cannot adjust the pacing rate in slow start which is called startup in BBR, see [41] for an in-depth analysis of the BBR startup parameterization. Yet, we remark that compared to paced Cubic, BBR chooses the pacing rate and calculates the congestion window, such that it doubles with every RTT even though not all packets have arrived yet (due to the pacing), effectively leading to a slightly different packet release mode than paced Cubic.

Despite these small differences, BBR is still very similar to paced Cubic which is why we compare our default paced Cubic variant (200% RATIO) to BBR in Figure 15. BBR overall compares well to our paced Cubic variant, yet, there are subtle differences. Looking at an RTT of 30 ms, we see that for many parameterizations BBR's average in our experiments is below that of paced Cubic. However, as indicated by the often largely overlapping confidence intervals this is not necessarily statistically significant. IW100 stands out, it seems that for lower bandwidths, BRR is able to better scale to the actual application demand (remember we are only sending slightly more than 50 frames). For an RTT of 100 ms, paced Cubic and BBR perform very similar. Looking at 250 ms, we find many instances where BBR's FCT is higher, especially when looking at the larger buffer sizes. It seems that the delay brought in by the Cubic elephant flow seems to further inflate the RTT observed by BBR causing it so reduce its sending rate.

Takeaway. *BBR's startup compares to our paced Cubic variant. BBR seems to have slight advantages at lower bandwidths and especially if congestion window exceeds the actual application demand. For large RTTs with larger buffers, it seems that our paced Cubic variant has advantages when the IW increases.*

VIII. Conclusion

This paper's goal is to better understand the current configuration of TCP's initial congestion window (IW). The IW is a long-debated performance parameter. Its size is in principle network and application dependent, where too small IWs can add unnecessary latency and too large IWs can cause congestion and thus loss. Yet, the IW is regarded as a *static* parameter that fits all networks and applications. Its IETF-recommended size has only changed infrequently in its history.

To this end, we utilize longitudinal IP-based scans of IPv4 and the Alexa list to study the adoption of IETF-recommended IW-values and find that IW10, as the current recommended value, gains more and more track. Further, we modify our scanner architecture to measure CDNs from our University network and from globally distributed vantage points. We find that CDNs are well ahead of current IETF-standardized practices by using *custom* IW configurations. In our measurement study, we observe IW configurations that are up to ten times higher than the most recent experimental standard. Our results suggest that CDNs do customize IWs for different services or customers, yet while advantageous for some content types, the content type does not enforce the IW. On a larger scale, we survey if CDNs adjust IWs depending on the end-user's network. We find some CDNs for which we can show that IWs vary depending on the network, but not for all. Driven by losses in our measurements, we analyze the burstiness of the IW delivery and find that some CDNs utilize pacing to space out packets over time. We find that the largest IWs in our study utilize this feature, which especially challenges the notion of IWs as they must not arrive within the first RTT, making it difficult to compare IWs just by the number of segments. We find it reasonable to couple an IW to a data rate, i.e., over which time is this IW transmitted, to better understand the IW's demand on the network.

Since our measurements do not show if pacing actually enables these large IWs, we carry our real-world observations to a controlled lab evaluation and investigate their impact on flow completion time when competing against a long-lived flow. Our testbed study indicates that pacing is a key component to enable larger IWs, yet, it still shows that blindly increasing the IW without regarding the actual network and application results in lower performance.

While our study focuses on TCP, QUIC borrows TCP's congestion control and startup phase including initial windows highlighting its future relevance (also in light TCP-BPF [42]). We thereby aim to inform standardization and academia about current global trends and CDN practices that depart from current knowledge and IETF-recommendations. We posit that further research needs to be dedicated to understand the implications of this new reality opening up the question if these customizations need to be reflected in RFCs.

References

[1] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is The Web HTTP/2 Yet?" in *PAM*, 2016.

[2] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, "A First Look at QUIC in the Wild," in *PAM*, 2018.

[3] T. Zimmermann, J. Rüth, B. Wolters, and O. Hohlfeld, "How HTTP/2 Pushes the Web: An Empirical Study of HTTP/2 Server Push," in *IFIP Networking Conference*, 2017.

[4] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet Inter-domain Traffic," in *ACM SIGCOMM*, 2010.

[5] T. Zimmermann, B. Wolters, and O. Hohlfeld, "A QoE Perspective on HTTP/2 Server Push," in *Internet QoE*, 2017.

[6] T. Zimmermann, B. Wolters, O. Hohlfeld, and K. Wehrle, "Is the Web ready for HTTP/2 Server Push?" in *ACM CoNEXT*, 2018.

[7] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM*, 1988.

[8] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC Editor, RFC 2414, 1998.

[9] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC Editor, RFC 3390, 2002.

[10] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin, "An Argument for Increasing TCP's Initial Congestion Window," *SIGCOMM CCR*, vol. 40, no. 3, pp. 26–33, 2010.

[11] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window," Internet RFC, RFC Editor, RFC 6928, 2013.

[12] M. Flores, A. R. Khakpour, and H. Bedi, "Riptide: Jump-Starting Back-Office Connections in Cloud Systems," in *IEEE ICDCS*, 2016.

[13] CDNPlanet, "Initcwnd settings of major cdn providers," Feb. 2017. [Online]. Available: https://www.cdnplanet.com/blog/initcwnd-settings-major-cdn-providers/

[14] M. Allman, "Removing TCP's Initial Congestion Window," Working Draft, IETF, Internet-Draft draft-allman-tcpm-no-initwin-00.txt, 2015.

[15] J. Rüth, C. Bormann, and O. Hohlfeld, "Large-Scale Scanning of TCP's Initial Window," in *ACM IMC*, 2017.

[16] J. Rüth and O. Hohlfeld, "Demystifying TCP Initial Window Configurations of Content Distribution Networks," in *IFIP Network Traffic Measurement and Analysis Conference (TMA)*, 2018.

[17] Akamai, "Q4 2016 State of the Internet - Connectivity Report," 2016.

[18] M. Scharf, "Performance Evaluation of Fast Startup Congestion Control Schemes," in *IFIP Networking Conference*, 2009.

[19] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC Editor, RFC 2001, 1997.

[20] A. Medina, M. Allman, and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet," *SIGCOMM CCR*, vol. 35, no. 2, pp. 37–52, 2005.

[21] F. Qian, A. Gerber, Z. M. Mao, S. Sen, O. Spatscheck, and W. Willinger, "TCP Revisited: A Fresh Look at TCP in the Wild," in *ACM IMC*, 2009.

[22] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM*, 1988.

[23] V. Visweswaraiah and J. Heidemann, "Improving Restart of Idle TCP Connections," University of Southern California Computer Science Department, Tech. Rep. 97-661, 1997.

[24] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the Performance of TCP Pacing," in *IEEE INFOCOM*, 2000.

[25] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Routers with Very Small Buffers," in *IEEE INFOCOM*, 2006.

[26] D. X. Wei, P. Cao, and S. H. Low., "TCP Pacing Revisited," 2006, [Online]. Available: http://people.cs.pitt.edu/ ihsan/pacing_cal.pdf.

[27] D. Liu, M. Allman, S. Jin, and L. Wang, "Congestion Control Without a Startup Phase," in *Protocols for Fast, Long Distance Networks Workshop*, 2007.

[28] J. Rüth, "Github: IW-Prober," 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1247327

[29] J. Rüth, "ZMap and Modules," 2017. [Online]. Available: https://github.com/COMSYS/zmap

[30] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *ACM Hotnets*, 2010.

[31] Q. Scheitle, O. Hohlfeld, J. Gamba, J. Jelten, T. Zimmermann, S. D. Strowes, and N. Vallina-Rodriguez, "A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists," in *ACM IMC*, 2018.

[32] S. Souders, I. Grigorik, P. Meenan, and R. Viscomi, "The HTTP Archive," 2018. [Online]. Available: http://httparchive.org/

[33] Google, "WebPagetest CDN domain list, cdn.h." [Online]. Available: https://github.com/WPO-Foundation/webpagetest/blob/master/agent/wpthook/cdn.h

[34] AOL and Google, "WebPagetest." [Online]. Available: https://www.webpagetest.org/

[35] D. Nobori and Y. Shinjo, "VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls," in *USENIX NSDI*, 2014.

[36] N. Dukkipati, E. Dumazet, and D. S. Miller, "TCP: increase default initial receive window." 2010. [Online]. Available: https://doi.org/10.5281/zenodo.1246469

[37] V. Visweswaraiah and J. Heidemann, "Improving Restart of Idle TCP Connections," University of Southern California Computer Science Department, Tech. Rep. 97-661, 1997.

[38] A. Dhamdhere, D. D. Clark, A. Gamero-Garrido, M. Luckie, R. K. P. Mok, G. Akiwate, K. Gogia, V. Bajpai, A. C. Snoeren, and K. Claffy, "Inferring Persistent Interdomain Congestion," in *ACM SIGCOMM*, 2018.

[39] S. Bauer, D. Clark, and W. Lehr, "The Evolution of Internet Congestion," in *Research Conference on Communication, Information and Internet Policy (TPRC)*, 2009.

[40] S. Floyd and E. Kohler, "Internet Research Needs Better Models," *SIGCOMM CCR*, vol. 33, no. 1, pp. 29–34, 2003.

[41] The Google BBR team, "BBR Startup Gain: a Derivation," https://github.com/google/bbr/blob/master/Documentation/startup/gain/analysis/bbr_startup_gain.pdf, 2018.

[42] L. Brakmo, "TCP-BPF: Programmatically tuning TCP behavior through BPF," in *NetDev 2.2*, 2017.

Jan Rüth received his B.Sc. and M.Sc. in Computer Science from RWTH Aachen University. In 2014, he joined the Chair of Communication and Distributed Systems at RWTH Aachen University as a Ph.D. student and researcher. His research interests include transport protocols, internet measurements, and performance evaluations in general.

Ike Kunze graduated from RWTH Aachen University with an M.Sc. in Computer Science and joined the Chair of Communication and Distributed Systems at RWTH Aachen University as a Ph.D. student in 2019.

Oliver Hohlfeld is a research associate in computer science and heads the Network Architectures group at the Chair of Communication and Distributed Systems at RWTH Aachen University. Before, he was with Anja Feldmann at TU Berlin / Deutsche Telekom Innovation Laboratories. He was a visiting scholar at the group of Paul Barford at the University of Wisconsin - Madison, USA. He studied computer science at Darmstadt University of Applied Sciences, Institute Eurecom (Sophia Antipolis, France) and Darmstadt University of Technology and holds a B.Sc. and M.Sc. degree. From 2004 to 2006 I was with the Fraunhofer IGD where I worked on network architectures for telemedical applications. I obtained a Ph.D. (Dr. rer. nat.) from TU Berlin in 2013.