Demystifying TCP Initial Window Configurations of Content Distribution Networks

Jan Rüth and Oliver Hohlfeld
RWTH Aachen University
{rueth,hohlfeld}@comsys.rwth-aachen.de

***Abstract*—Driven by their quest to improve web performance, Content Delivery Networks (CDNs) are known adaptors of performance optimizations. In this regard, TCP congestion control and particularly its initial congestion window (IW) size is one long-debated topic that can influence CDN 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 of major CDNs. Our study uses a globally distributed infrastructure of VPNs giving access to residential access links that enable to shed light on network-dependent configurations. 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. Further, we find CDNs that utilize different IWs for different customers and content while others resort to fixed values. 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 drifted away from recommended and standardized practices.**

I. Introduction

Content Distribution Networks (CDNs) are a key component of today's Web. Their ongoing quest to serve web content from nearby servers has flattened the hierarchical structure of the Internet [1] and promises 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. For example, Google has pushed several improvements to web technology, including new protocols such as HTTP/2 (through SPDY) or QUIC, both have found swift adoption [2]–[4] by other CDNs. While adopting new technologies offer 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 [4], [5]. Some of these configuration challenges stem from the fact that they are depended on network and application characteristics.

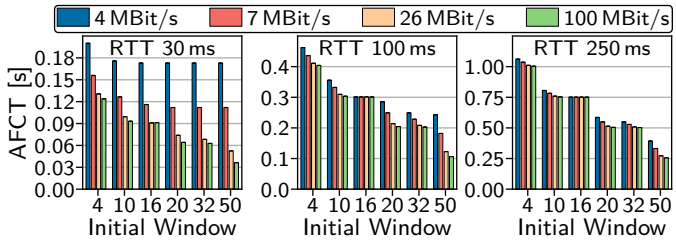
One long-debated performance configuration parameter is TCP's (and soon QUIC's) Initial Congestion Window (IW) size. The IW size controls the amount of unacknowledged data sent at connection start and thereby heavily influences the start-up behavior of new and especially short-lived connections (e.g., typical web transfers) or those that are revived from idle. A small IW can prolong transmissions and cause unnecessary latency as TCP needs to await feedback (ACKs) to increase the congestion window. Contrary, too large IWs can lead to loss and retransmissions when the network cannot handle large bursts of data. Thus choosing the optimal value for each network is critical for good performance—and 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 [6], its recommended size has only changed twice, to 2-4 segments in 1998 [7], [8] and—motivated by increasing access speeds and shorter page loading times [9]—to 10 segments in 2013 [10]. It was very recently shown that IW customization can help in reducing CDN latency [11]. In this regard, a small-scale study by CDNPlanet showed that half of the probed CDNs use IW10 as IETF recommended size while others already use larger IW sizes [12]. In this regard, others [13] even argue to abandon static IETF standardized values for the IW to enable customization already in the standards. Yet, little is known about CDN specific IW configurations.

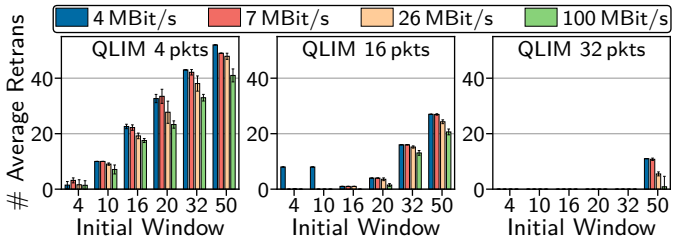
In this paper, we broadly probe CDNs to gather an empirical understanding on how IW customization already takes place in today's Internet. By scanning CDN IW configurations, also from globally distributed vantage points, we extend previous work [14] and shine a light on the degree of customization that CDNs show today. Our results show that IW customization beyond standardized practices is already common practice and there exists a gap between standardization, research, and Internet reality. Specifically, this work contributes the following:

- The first comprehensive analysis of current IW configuration practices of CDNs. We show that IWs are configured up to ten times higher than IETF's current experimental standard.
- Further, we observe that CDNs are variable in their IW usage. We find multiple instances of CDNs that deliver data using different IWs for different customers.
- 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.

Structure. Sec. II discusses related work and shows how IWs are defined, standardized, and how they impact performance. Our IW scanning methodology and architecture is introduced in Sec. III. Following, Sec. IV–Sec. VI paints the global CDN IW configuration space and Sec. VII concludes our findings.



(a) Average Flow Completion Time (AFCT) when varying bandwidth, latency, and IWs. Horizontal lines mark roundtrips.



(b) Losses increase when increasing IWs depending on bottleneck bandwidth and queue sizes.

Fig. 1: Improvements in latency (a) and increases in losses (b) when varying initial congestion windows for different network settings. Benefits of increased initial congestion windows are highly network-dependent.

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 web performance, 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 of cached TCP connections 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 [10]) 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.

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 chose 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 configurations reported by Akamai [15]. We further choose six IW configurations of 4, 10, 16, 20, 32, and 50 segments, to reflect the current standard of 4 segments [8], the current IETF recommendation of 10 segments [10], 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 average AFCT for the different parameters and its standard deviation is shown in Figure 1a. 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 latency 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 conducted 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. We repeated every configuration 30 times and show the average retransmissions required and standard deviation in Figure 1b.

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. When we observe 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 [9], [16], 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 [6] and 9 years later standardized in 1997 [17]. This setting was experimentally extended to 2-4 segments (or 4380 byte) in 1998 [7] and later moved to a proposed standard [8]—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, [9] proposed in 2010 and later RFC 6928 [10] recommended in 2013 to increase the IW to ten segments. Most recently, Allman [13] 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 understanding of the IW is necessary to understand current network performance. This understanding has been gained in both active and passive measurement studies. With regards to active measurements, Medina et al. [18] probed 85 k servers in 2004 and found most servers to be on an IW of one or two with only 1% of host having an IW larger than four. Our measurements are methodological similar to those of Medina, but with a direct focus on CDNs which were still on the rise in 2004 and did not have as much footprint as today. With regards to passive measurements, Qian et al. [19] 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 [12] 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 in that we share the same goal to shed light on CDN IW configurations, but their methodology did not allow CDNPlanet to grasp a broad assessment of CDN IW configurations which is the focus of this work. In [14] we proposed an approach to estimate TCP's IW for all reachable IPv4 HTTP and TLS hosts which forms the basis of this work. Our previous approach did enumerate the IPv4 space without host names as a-priori knowledge, thus it cannot measure CDN-hosted sites that use the server name (SNI) within the TLS handshake to indicate which site to deliver. Many CDNs will only respond with an error paper when no server name is presented, which is often insufficient data to establish an initial window for the probed host and therefore CDNs are not accurately represented in our previous work. The extension and application to measure CDN IW configuration thus is the focus of this work.

III. Measuring CDN IW Configurations

We next describe our approach to estimate the IW size, its validation, and our overall scanning architecture.

A. Measuring IWs

We begin by summarizing our IW size estimation approach. To enable measuring CDNs, we extend our IW-prober [14] to now account for SNI 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 from to account for CDN properties.

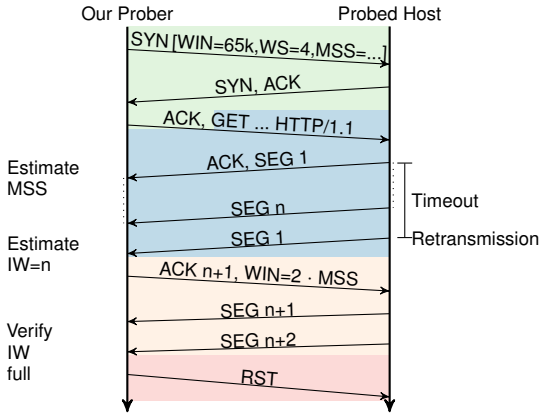


Fig. 2: Scan procedure: MSS and large receive window are announced and no ACKs are sent until a retransmission. The estimated IW is the # bytes received before the retransmission.

The IW estimation procedure (visualized in Figure 2) can be split into four phases: *First*, a regular TCP handshake is performed announcing a large receive window and, to account for overly large IWs configured at some CDNs, also a window scaling option 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, e.g., causing 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 methodology is prone to tail-loss, it is recommended to perform multiple scans of the same host.

Implementation and Validation. We implement our tool [20] in go-lang to benefit from its multiprocessing capabilities. To test our implementation and to validate the correctness of the IW estimation, we test our tool in mininet [21]. 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 configuration 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. Out tool always provided correct IW estimations except for tail-loss (as expected).

B. Measuring CDN IWs

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 [22]. 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 throughout the crawl, i.e., the landing page URLs as well as objects that are subsequently requested 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 [23] published by the WebPagetest [24] framework (the framework driving the HTTP Archive), this list 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 hosted on 69 CDNs for which we are going to establish initial windows. 116K objects (25 CDNs) that are too small to reliably estimate an IW (for large segment sizes, see Sec. IV) are removed from the results.

Scanning Architecture. We use the architecture depicted in Figure 3 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.

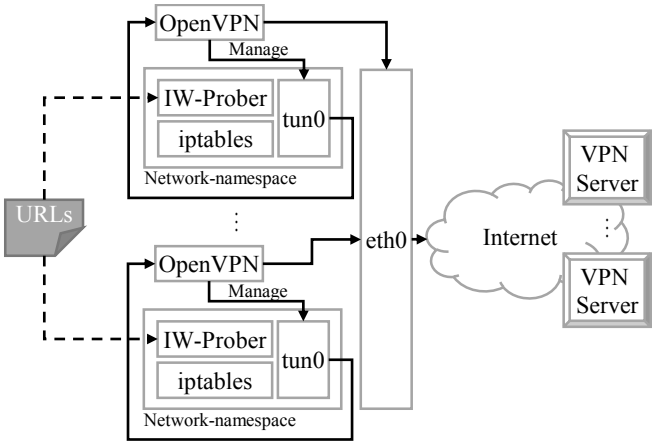


Fig. 3: Overview of our scanning architecture. We leverage Linux network namespaces to scan concurrently and to easily manage multiple VPN connections.

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 locally. We select query candidates by grouping URLs hosted at the same CDN using the same IW (derived locally) and select a random sample of URLs for each (CDN, IW) pair.

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 [25] 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 them 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 use only 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 [25], 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 smoothes burst and limits the outgoing bandwidth. We configure it to transmit at most 100 packets per second, thus we limit the prober to $\approx$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.

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 Sec. V) to set an upper bound on the expected IW sizes. As our network's

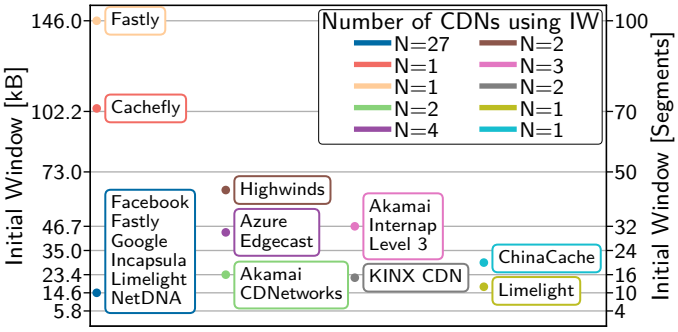


Fig. 4: CDN IWs as seen from our university network. IETF sizes 4 (not shown) and 10 are present but also larger IWs.

upstream ISP peers with DE-CIX (at which many CDNs peer as well) and our networks 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.

IW Probe Procedure. As IWs can be configured in bytes or segments, we scan each URL (see Sec. III-B) 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 fill a larger IW. This would bias the results towards smaller IWs.

A. IW Sizes

Figure 4 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 [14], which found IETF recommended IW sizes to dominate given the number of 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. This large IW has already caused drops during our measurement which might be an indicator that IW100 is too large. Cachefly also shows a larger than usual IW of 105 kB, notably, Cachefly uses a fixed IW 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.

[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.

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 [26] 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) and IW sizes larger than standardized are practically relevant. 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.*

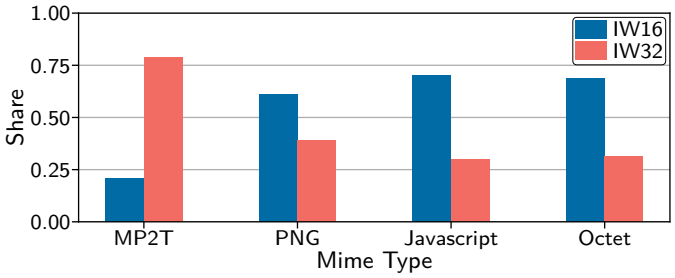


Fig. 5: IW distribution for Akamai URLs per mime type.

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.

Figure 5 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, 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 mine 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 [25]. 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.

#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.

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 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 for the consumer (#1) and for 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

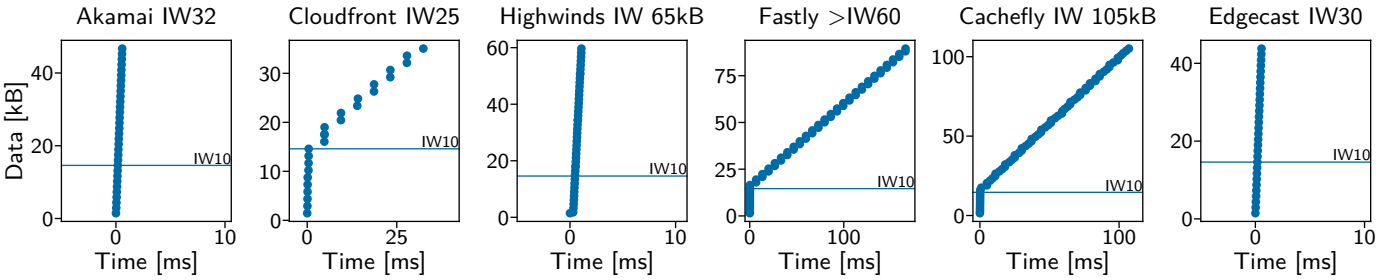


Fig. 6: IW burstiness for a subset of the observed CDNs and URLs (each with an RTT of 60 ms-70 ms). The arrival time of full-sized 1500 byte packets (dots) in the entire IW is shown on the x-axis, the IW size (in kB) on the y-axis (e.g., IW10 = 15 kB). Note different axis scalings due to different IW sizes. Some CDNs seem to utilize packet pacing while others do not.

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 observe different IWs consistently. 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 (equalling to 72 full-sized segments) that we observed shows losses (which does not show any losses at other VPN).

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 with IW32 and are successful in delivering it while e.g., Edgecast and Azure with IW30 show losses over the same links.*

Motivated by these observed losses, we want to investigate the burstiness of IWs. To this end, a recent proposal [13] 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 [27] 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 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.

Pacing Realization. The Linux pacer's default configuration works as follows (at the start of a connection): during the three-way handshake, the RTT is estimated as the difference between the SYN and the corresponding ACK. TCP then calculates a pacing rate as the ratio of the current congestion window (in this case the IW) and the estimated RTT which results in a rate at which data will be scheduled to leave the system. The pacer enforces the limit by delaying packets, however, it allows a configurable initial burst of ten full frames and the subsequent frames are sent in trains of two packets (also configurable). Furthermore, the pacer can be configured to be more or less aggressive during slow-start or congestion-avoidance by passing a sysctl parameter that is multiplied with the estimated pacing rate. Thus the expected outcome at the connection start is a burst of ten packets and following trains of two packets. 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 found that the packet coalescing of the NIC that reduces the interrupt-rate causes imprecise software timestamps of the arriving packets, we thus instruct tcpdump to configure our NIC to perform hardware timestamping at packet arrival.

Figure 6 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. This is best visible in the example of Cloudfront, were 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 [13].

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.

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 on the impact of pacing on networks as well on its benefits and drawbacks especially as current pacers deviate from a perfectly paced packet streams found in literature.*

VII. Conclusion

This paper's goal is to better understand the current configuration of TCP's initial congestion window (IW) by CDNs. 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 changed only infrequently in its history.

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. The largest IWs in our study utilize this feature, yet there is no clear indication if pacing enables these large IWs as the benefits or drawbacks of pacing require further research. While our study focusses on TCP, QUIC borrows TCP's congestion control and startup phase including initial windows highlighting its future relevance (also in light TCP-BPF [28]). We thereby aim to inform standardization and academia about current 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.

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

Acknowledgment. Funded by the Excellence Initiative of the German federal and state governments, as well as by the German Research Foundation (DFG) as part of project B1 within the Collaborative Research Center (CRC) 1053 – MAKI.

References

[1] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet Inter-domain Traffic," in *ACM SIGCOMM*, 2010.
[2] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is The Web HTTP/2 Yet?" in *PAM*, 2016.
[3] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, "A First Look at QUIC in the Wild," in *PAM*, 2018.
[4] 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.
[5] T. Zimmermann, B. Wolters, and O. Hohlfeld, "A QoE Perspective on HTTP/2 Server Push," in *Internet QoE*, 2017.
[6] V. Jacobson, "Congestion Avoidance and Control," in *ACM SIGCOMM*, 1988.
[7] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC 2414, 1998.
[8] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC 3390, 2002.
[9] 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.
[10] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window," Internet RFC, RFC Editor, RFC 6928, 2013.
[11] M. Flores, A. R. Khakpour, and H. Bedi, "Riptide: Jump-Starting Back-Office Connections in Cloud Systems," in *IEEE ICDCS*, 2016.
[12] CDNPlanet, "Initcwnd settings of major cdn providers," Feb. 2017. [Online]. Available: https://www.cdnplanet.com/blog/initcwnd-settings-major-cdn-providers/
[13] M. Allman, "Removing TCP's Initial Congestion Window," Working Draft, IETF, Internet-Draft draft-allman-tcpm-no-initwin-00.txt, 2015.
[14] J. Rüth, C. Bormann, and O. Hohlfeld, "Large-Scale Scanning of TCP's Initial Window," in *ACM IMC*, 2017.
[15] Akamai, "Q4 2016 State of the Internet - Connectivity Report," 2016.
[16] M. Scharf, "Performance Evaluation of Fast Startup Congestion Control Schemes," in *IFIP Networking Conference*, 2009.
[17] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001, 1997.
[18] 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.
[19] 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.
[20] J. Rüth, "Github: IW-Prober," 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1247327
[21] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *ACM Hotnets*, 2010.
[22] S. Souders, I. Grigorik, P. Meenan, and R. Viscomi, "The HTTP Archive," 2018. [Online]. Available: http://httparchive.org/
[23] Google, "WebPagetest CDN domain list, cdn.h." [Online]. Available: https://github.com/WPO-Foundation/webpagetest/blob/master/agent/wpthook/cdn.h
[24] AOL and Google, "WebPagetest." [Online]. Available: https://www.webpagetest.org/
[25] 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.
[26] N. Dukkipati, E. Dumazet, and D. S. Miller, "Tcp: increase default initial receive window." 2010. [Online]. Available: https://doi.org/10.5281/zenodo.1246469
[27] V. Visweswaraiah and J. Heidemann, "Improving Restart of Idle TCP Connections," University of Southern California Computer Science Department, Tech. Rep. 97-661, 1997.
[28] L. Brakmo, "TCP-BPF: Programmatically tuning TCP behavior through BPF," in *NetDev 2.2*, 2017.