Large-Scale Scanning of TCP's Initial Window

Jan Rüth, Christian Bormann, Oliver Hohlfeld
Communication and Distributed Systems, RWTH Aachen University
{rueth,bormann,hohlfeld}@comsys.rwth-aachen.de

ABSTRACT

Improving web performance is fueling the debate of sizing TCP's initial congestion window (IW), which is a critical performance parameter especially for short-lived flows. This debate yielded several RFC updates to recommended IW sizes, e.g., an increase to IW10 in 2010. The current adoption of IW recommendations is, however, unknown. In this paper, we therefore conduct large-scale measurements covering the entire IPv4 space inferring the IW distribution size by probing HTTP and HTTPS servers. We present an HTTP and TLS scanning method implemented in ZMap, enabling quick estimations of IW sizes at Internet scale. For the first time since the standardization and implementation of IW 10, we shed light on the rugged landscape of IW configurations on the Internet.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Network measurement**;

KEYWORDS

TCP Initial Window, Measurements

ACM Reference Format:
Jan Rüth, Christian Bormann, Oliver Hohlfeld. 2017. Large-Scale Scanning of TCP's Initial Window. In *Proceedings of IMC '17, London, United Kingdom, November 1–3, 2017,* 7 pages.
https://doi.org/10.1145/3131365.3131370

1 INTRODUCTION

For decades network protocol engineering has focused on improving *throughput*. Recent advances instead focus on *latency* reductions, e.g., fueled by attempts to optimize the interaction experience with web services. The results of these approaches are reflected in attempts to replace TCP as long-established transport protocol (e.g., by QUIC [12] or MinimaLT [19]) or attempts to reduce initial loading delay with handshake optimizations (e.g., by TCP Fast Open [21] or TLS False Start [14]). Since protocol extensions or even replacements can face slow adoption rates—especially for transport protocols—protocol *parameter optimizations* denote a popular area of performance optimization.

In this respect, one long-lasting debate concerns the size of TCP's initial congestion window (IW). This parameter controls the amount of unacknowledged data that can be sent after connection setup and therefore directly influences Internet traffic characteristics (e.g., traffic burstiness) and application performance (especially for short-lived flows). Concretely, small IWs can prolong transmissions during TCP's slow start, especially for request/response protocols such as HTTP or the TLS handshake. At the beginning of the connection, a small request triggers a potentially large response that does not fit into a small IW thus prolonging the transmission for at least one round-trip. On the other hand, large IWs generate traffic bursts that can overflow low-capacity links. The debate on configuring an Internet-optimal IW started in the late 90s currently yielded to the deployment of IW 10, initially proposed in 2010 [9] and enabled in the Linux kernel as a default in 2011 [16]. The advocators argued that IW 10 allows transmitting the initial response within the first round trip of the connection for large fractions of the web traffic. A recent IETF draft [2] questions this practice of standardizing an IW that is optimal for all applications and Internet hosts. Instead, this draft argues that the IW should be configured custom to each deployment and application. Despite this debate on the *proposed* IW, the *actual* IW values of Internet hosts remains largely unknown. Given the relevance of the parameter for both Internet traffic and application performance and in light of the ongoing debate on its optimal value, we posit that a deeper empirical understanding of current practices is needed.

This paper describes the first *comprehensive* assessment of TCP's IW configuration among TLS/HTTP hosts. The goal of our work is to inform the current debate with an up-to-date view on IW configurations resulting from a large-scale assessment of all public IPv4 hosts reachable on port 80/tcp (HTTP) and 443/tcp (HTTPS). This view is particularly relevant since it is the first assessment after the standardization of IW10. Our study is enabled by an extended measurement methodology that enables to scan HTTP and TLS hosts without any prior knowledge. We contribute:

- We conduct the first ever large-scale IW scan over HTTP/TLS reachable hosts in the entire IPv4 space. Yet we show that scanning a small random subset (e.g., as small as 1%) is actually enough to infer a representative IW distribution. Scanning smaller random sets helps in reducing the Internet scan footprint.
- We provide a measurement methodology utilizing HTTP and TLS to estimate IW settings without prior knowledge on an Internet scale. We make our implementation of this methodology in the ZMap scan tool openly available [22].
- We observe a rather rugged landscape of IWs; while server networks show high deployments of IW10, other (legacy) networks still use older values. Notably, some services run IW configurations customized to different services.

Structure. Section 2 presents related work on IW scans and Section 3 presents the design of our measurement. We then discuss our scan results in Section 4 and conclude the paper in Section 5.

IMC '17, November 1–3, 2017, London, United Kingdom

2 RELATED WORK

The relevance of TCP's initial congestion window (IW) 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 [13] and 9 years later standardized in 1997 [24]. This setting was experimentally extended to 2-4 segments (or 4380 B) in 1998 [3] and one year later moved to a proposed standard [4]—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 [8] recommended in 2013 to increase the IW to 10 segments. Most recently, Allman [2] 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, 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. [15] probed 85 k servers in 2004. The size of the probe set was limited since prior knowledge of the targets was required—a property that is not needed in our scan methodology which enables us to probe the entire IPv4 address space. With regards to passive measurements, Qian et al. [20] inferred IW distributions from several traces in 2009. While their data set covers traces captured in a diverse set of networks and also covers non-publicly visible hosts, our IW assessment based on active measurements allows probing of the entire IPv4 address space containing all publicly reachable hosts. A small-scale study by CDNPlanet [7] probed 15 CDNs via HTTP and found 6 to use IW10 and others to use larger IWs. Further, since no large-scale assessment is available to track recent changes in IW parameterization (e.g., IW10 [8] and the proposed abandoning [2] of default IWs), we argue that an updated view on the current IW deployment is required. We update this view with an assessment of TCP's IW for reachable IPv4 HTTP and TLS hosts.

3 INITIAL WINDOW SCAN DESIGN

We use active measurements to extensively assess TCP's IW configurations deployed by HTTP/TLS hosts in the IPv4 space. This enables us to assess *all publicly reachable IPv4 hosts*, including content infrastructure such as CDNs for which the IW can be a relevant performance aspect. Since the IW size is not advertised in the TCP headers (e.g., unlike the Maximum Segment Size), the IW size can only be *inferred* from the sender's behavior. This IW size inference is thus at the core of our methodology, which we detail next.

3.1 General Initial Window Size Inference

We base our scan on the method of Padhye and Floyd [18], which we summarize and extend next and depict in Figure 1. The IW estimation starts with performing TCP's 3-way handshake in which a certain MSS and a large receive window is announced within the SYN packet. Advertising a large receive window ensures that sending is only limited by the IW and not by flow control. To infer the IW size, we send a request to trigger a data transfer by the remote host upon completion of the handshake. The remote host will either have sufficient data to send, utilizing the full IW,

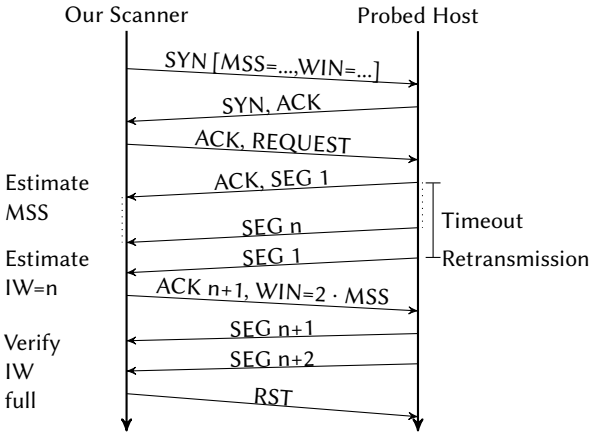

Figure 1: Scan procedure: A small MSS is announced and verified, preventing to run out of data prior to reaching IW. The estimated IW is the # bytes received before retransmission.

or stop sending before reaching the full IW size if the request did not trigger a large enough response. As in [18], we do not send acknowledgments causing the remote end not to increase the congestion window and to eventually trigger a retransmit of its first segment. Our scanner can then simply count the bytes and packets it received and assume this to be the IW.

This basic procedure is challenged by the presence of packet reordering and loss. To account for these challenges, we inspect the sequence numbers to detect both events. While this approach enables us to easily detect reorderings, the detection of packet loss can be more difficult. First, if one needs to further analyze the received data *besides* the IW assessment, lost packets would need to be retransmitted, which given the scanners methodology is impossible. Second, packet loss at the end of the stream (tail loss), i.e., in the last packet, cannot be detected and thus may lead to erroneous IW estimates. Furthermore, TCP tail loss probes could set off the estimated value, thus we do not enable selective acknowledgement effectively disabling tail loss probes. Performing multiple scans of the same host can increase the likelihood of detecting tail loss.

However, the biggest challenge when applying this technique to an unknown set of hosts is triggering large enough responses that fill up the senders IW. This is because, in the absence of prior knowledge, the response size to a generic request is *unknown*. In the event of responses smaller than the configured IW, the IW size cannot be estimated. We address this challenge in two ways.

First, we maximize the number of transmitted segments by limiting the MSS advertised in the TCP handshake. This is possible, since the IW is configured in bytes depending on the advertised MSS [8]:

$$\mathrm{IW} = \min(10 \cdot \mathrm{MSS}, \max(2 \cdot \mathrm{MSS}, 14600)).$$

This definition is twofold: on the one hand, it defines an upper limit in bytes. On the other hand, it suggests to reference the IW just by the factor employed to the MSS, i.e., by the number of packets. Thus, by announcing a *small* MSS, we can effectively lower the amount of response bytes that are required to fill the IW. However, no standard defines the smallest possible MSS, only a default MSS of 536 B is defined. We therefore examined fresh copies of multiple operating systems to test for the smallest possible MSS. We observed that

Linux will typically reject an MSS below 64 B. All tested variants of Microsoft Windows default to 536 B if the MSS falls below that value. Concluding, we announce an MSS of 64 B, but monitor the actually used segment size and use the observed maximum for our IW estimation.

Second, we derive HTTP and TLS-based probing methods to trigger large requests without prior knowledge which we detail in the succeeding sections. Yet, regardless of the method we can determine whether hosts are limited by the IW or simply because they are short of data to transmit. After having received the retransmission of the first segment, we will now acknowledge the last received segment, effectively acknowledging all data sent before. In case a host has more data, the acknowledgements cause a release of more segments as the congestion window increases and the number of unacknowledged bytes reduces, thus the host was in fact limited by the IW. In contrast, if the host is out of data, no new segments will arrive, thus we conclude that the estimation failed as we cannot be sure that the IW was filled. To limit the potential load of segments our scanner has to process, we leverage flow control by signaling a small receive window of only two segments to limit the sender while still having some redundancy for the detection.

3.2 HTTP-based IW Inference

To generate sufficient response data, the first method relies on inferring the IW by probing HTTP servers. This choice is motivated by the widespread deployment of HTTP as major application layer protocol, which thus provides a good candidate for our IW scans. According to [1], HTTP accounts for over 50% of traffic at a major European IXP and, according to our scans, we can successfully exchange data with ≈ 48.3 M hosts on port 80. For the same reasons, HTTP was used in prior works to infer the IW size, e.g., Padhye and Floyd [18] or Medina et al. [15]. Both works can only ensure large enough responses that fill the IW by providing URL lists defining an appropriate request for each probed host. However, an extensive assessment of the entire IPv4 space is *not* feasible by relying on prior knowledge. We therefore propose an extended approach that allows inferring the IW of HTTP servers *without* any prior knowledge such as precompiled URL lists triggering large responses.

Our proposed approach is as follows; we initially request the `/` page hoping that it contains enough payload to fill the IW. As we have no prior knowledge of the host, we can only include the IP in the mandatory HTTP `Host` header. Many (virtualized) servers will reply with a `301 Moved Permanently` error, thus we can extract a valid URI from the `Location` header in the error response. In these cases, the extracted URI will redirect us to a valid page. We can further provide a valid `Host` header, if the host's common name is included in the URI. These information enables to again issue a request that hopefully results in a larger response. So, we send a `RST` to quickly end the connection and issue another request on a new connection following the redirect.

If redirecting fails, we increase the response size by bloating possible error pages. This approach is motivated by an initial observation (not shown) that a substantial number of servers replies with `404 Not Found` pages that include the URI that could not be found. Thus, enlarging the request URI will enlarge the error response. We therefore request a long URI indicating the nature of the scan, anticipating a long enough response. We choose the URI to fill up the MTU of our connection, thus transmitting more bytes than we announced we would be capable of in the MSS.

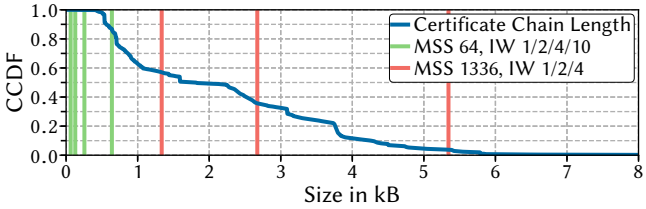

Figure 2: CCDF of certificate chain length of 36.5M hosts from censys.io data. TCP payload sizes covered with several IWs using MSS of 64 and a typical MSS of 1336 B.[1]

In addition to acknowledging segments to look for more data, we additionally infer if the IW was actually exhausted by exploiting HTTP characteristics. Concretely, we request that the remote end closes the connection upon finishing the transmission by including the `Connection: close` HTTP header. This should lead to a `FIN` once the remote end has transmitted all data. However, if the remote end has still data after it filled the IW, it cannot send the `FIN` as it still has data in its transmit queue. By receiving the `FIN`, we infer that the IW was not reached (in the absence of packet loss).

3.3 TLS-based IW Inference

Rising security and privacy concerns contributed to an increasing use of HTTPS, the TLS tunneled version of HTTP [17]. After port scans of the default HTTPS TCP port 443, we were able to successfully exchange data with ≈ 42.6 M hosts. This share is further expected to grow. Not only traditional services (e.g., banking or e-commerce) are using TLS also big players such as Facebook or Google have started to secure all of their traffic, further motivating others to switch. This trend is also manifested in HTTP/2 [6]—even though not mandated by the standard, practically all current implementations enforce TLS. Given this increasing relevance, we next detail a TLS-based inference method.

Our IW inference utilizes the TLS handshake in which a large response is sent by the server. In TLS, the handshake is initiated with a `client hello`, indicating, e.g., cipher suites or extensions. Upon reception, the server replies with a `server hello` choosing a cipher and depending on that, key material. Most importantly, the server continues to transmit its certificate chain that is required to validate its trust. Certificates typically dominate the server's answer in the number of bytes.

We analyzed TLS handshakes using the data provided at censys.io. Figure 2 shows the complementary CDF of server certificate chain length of 36.5 M hosts. On average, the certificate chain length was 2186 B (minimum 36 B and maximum 65 kB). For our scan to succeed, the remote host needs to send us at least IW · MSS bytes. Assuming an MSS of 64 B and IW 10, we only need 640 B of certificates which are supplied by more than 86% of the hosts. We can still reach 50% of the hosts even if they would use IW 34. These calculations neglect the actual size of the `server hello` and possible extensions that follow, yielding even more payload to rely on.

To scan a host, we initiate the TLS handshake after completing the TCP handshake. Since completing the TLS handshake relies on

the offered cipher suites by the client, we compiled a list of 40 TLS ciphers announced by Safari, Firefox, and Chrome and enriched the list with ciphers that we extracted from the censys.io data that were not already announced by the three browsers. To generate even more data, we included extensions for requesting OCSP stapling.

We rely on our acknowledging method to determine if the IW was filled or not. Yet in contrast to HTTP, we could have inspected the TLS length fields and use these to determine if we can still expect more data to be available. However, looking into payloads requires that we have no packet loss and it further complicates the implementation and we found no advantage doing so.

3.4 ZMap-based Implementation

Our ability to probe the entire IPv4 space in less than a day is based on ZMap [10]. ZMap is designed for a single packet exchange with the target to probe for open ports. Since this optimized port scan design is not capable of exchanging multiple packets with the target (as needed for TCP), we added this functionality in a lightweight fashion. We added a probe module to establish TCP connections and keep track of various per-connection properties such as the length of each segment and connection state. This design still allows us to perform fast scans, e.g., at a moderate scan rate of only 150 k packets/s, our HTTP-based IW scan only needs 7.5 hours to probe the entire IPv4 address space. An unmodified ZMap scanner performs a port scan at the same rate in only 6.8 hours—recall that the unmodified scanner performs only a single packet exchange with the host, instead of full TCP connections with subsequent exchanges. This highlights the efficiency of our scan method.

3.5 Validation

We manually validated our IW estimation approach in two controlled testbed experiments by running different versions of Linux and Windows. Each OS ran (TLS)-web servers, serving different files to trigger both cases of having *i)* enough and *ii)* insufficient data available (i.e., less than the IW) and captured packet traces. In the first experiment, we compared our estimator against ground truth by comparing against the true IW value configured in each OS. When enough data was available, the estimator provided the correct IW in all tested cases. In a second experiment, we added packet loss using NetEM to check its influence on the estimation accuracy. We restrict this experiment to only probe a single recent Linux and manually inspected each packet trace. All obtained IW estimates were correct in the absence of tail loss. Only instances with tail loss would lead to an *underestimation* of the IW. We argue that multiple scans per host can limit the likelihood of underestimated IWs.

4 RESULTS: IW DISTRIBUTIONS

Scan setup. To evaluate the IW distribution on the Internet, we operate a scanner within our University's network. This operation is closely coordinated with the University's IT Center to properly react to abuse emails and to have unfiltered access to the Internet (e.g., without transparent web proxies). We followed the guidelines in [10] and set up reverse DNS entries and a web page explaining the nature of the scans together with an opt-out mechanism. Unroutable or blacklisted IPs were not scanned.

[1]We implemented an ZMap based ICMP path discovery following RFC 1191 estimating typical MSS values, highlighting the IW requirements of TLS. We found 99% (80%) of all hosts support an MSS of 1336 B (1436 B).

Scan	Reachable	Success	Few Data	Error
HTTP	48.3 M	50.8%	47.6%	1.6%
TLS	42.6 M	85.6%	13.3%	1.1%

Table 1: Scan data set overview (rounded) scanned with MSS 64. Reachable meaning data exchange is possible.

Dataset. The presented results are based on two scans performed in the second and third week of August 2017 and are summarized in Table 1. We declare a success, if we are able to estimate the IW, we mark a scan as few data if we cannot be certain that the IW was actually exhausted, error marks all other cases (e.g., connection reset). For our measurements we decided to probe each host three times to account for tail loss and count it successful if at least two out of three probes yield the same result and as tail loss may occur, we require them to be the maximum of all three probes. To further test if hosts adjust their IW based on the announced MSS (recall that the standard also defines a byte limit), i.e., to always transmit a certain amount of bytes in contrast to segments, we scan with an MSS of 64 B and 128 B. To ensure no temporal changes at the host, all six probes (three for each MSS) are sent after each other.

HTTP Scan Ethics. Our HTTP probing methodology is arguably more intrusive than TLS-based probing. The reason is that HTTP probing is requesting actual web (error) pages and thus generates entries in server access logs. These entries triggered a significantly higher number of abuse e-mail than our TLS-based probing. As discussed in Section 4.1, the Internet-wide probing footprint can be drastically reduced by only probing a random sample of IPs to get stable IW distributions: currently, probing 1% IPs suffices.

Success rates. In total, we successfully probed 60.9 M distinct IPs, of which 7 M offered both services. Table 1 shows that TLS yields higher success rates than HTTP. HTTP probing of unknown hosts mainly suffers from not generating sufficient response data for IW inference. We tried to mitigate this by expanding error pages with long URLs, yet we found that, e.g., Akamai changed their default error page during our scans to not include the URL anymore. In contrast to HTTP, TLS returns more data (e.g., due to certificates) and is less intrusive. Still, around 13% of hosts return insufficient data. We attribute this to missing Server Name Indication information since connections are closed when no (forward) DNS names are presented, which are unavailable when only enumerating IPs.

4.1 Overall IW Distribution

We start by exploring the overall distribution of IW sizes for both HTTP and TLS. The reported results are based on successful IW estimations (see Table 1) with an MSS of 64 bytes. Figure 3 shows dominant IWs, i.e., observed at more than 0.1% of the hosts. We see that both scans are dominated by the IWs of 1, 2, 4, and 10 segments. These IWs are present at more than 97% of all scanned HTTP or TLS hosts. This finding is in line with recommendations in various RFCs. Out of 7 M IPs that appear for both HTTP and TLS, 6.2 M agree in their IW estimate and 858 k IPs yield different IW estimates for HTTP and TLS. Interestingly, we find that the TLS scan and the HTTP scan differ in the distribution of IW4 and IW10: we find more TLS hosts with IW 4 than HTTP. In contrast

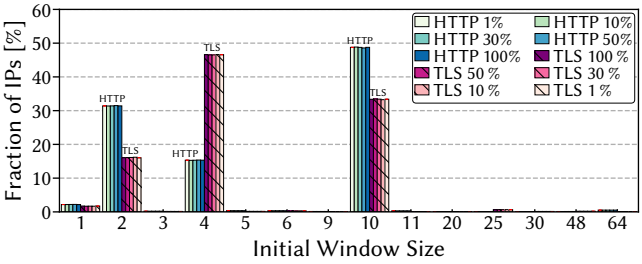

Figure 3: IW distribution in IPv4 of HTTP and TLS for IWs used by at least 0.1% of the hosts, probed with an MSS = 64 B.

Scan	NoData	IW1	IW2	IW3	IW4	IW5	IW6	IW7	IW8	IW9	IW10
HTTP	4.8%	16.5%	7.1%	7.2%	2.9%	3.6%	2.0%	45.0%	2.7%	1.1%	0.9%
TLS	17.8%	56.3%	5.6%	0.7%	1.9%	2.8%	2.4%	2.4%	3.4%	0.4%	0.8%

Table 2: Lower bounds of IWs for hosts that did not send enough data and the MSS observed in the connection.

to the measurement by Medina et al. [15] from 2005, we observe that IWs of 4 and 10 segments have gained the highest relative growth. When analyzing non-standard IWs, we observe two peaks, one at 25 (TLS) and one at 64 segments (HTTP). However, the low overall deployment of IW 10, especially on TLS-enabled host, is notable, given its standardization in RFC 6928 is already four years old and its implementation, released in the Linux kernel 2.6.39 from May 2011, is even older.

Lower IW bound for hosts with insufficient data. As indicated by our success rate, we are roughly missing half of the HTTP (and 13.3% TLS) hosts by not having enough data available for IW probing (see "Few Data" in Table 1). To better understand these hosts, Table 2 shows their minimum supported IW, i.e., before they run out of data. The picture is different for HTTP and TLS. For HTTP, we find that 45.0% of probed hosts run out of data after having transmitted data worth of an IW of 7. Given the current standards, it is very likely that these hosts are actually configured to use an IW of 10. For TLS, 17.8% do not sent any data (i.e., 4× more than for HTTP) and 56.3% run out of data after an IW of 1. This is likely caused by hosts not supporting our cipher suits or TLS versions offered by our probing module. Here, we are not receiving any certificates but only TLS error messages. In these cases, no speculations on likely IW configurations can be made.

Scanning 1% is enough! We next investigate if the Internet-wide scan footprint can be reduced by limiting IW probing to a smaller subset of hosts. We thus extracted a random subset of 50%, 30% and 1% of all successfully probed IPs for both the HTTP and the TLS scan and show their IW distributions in Figure 3. For the 1% sample we additionally show the mean of 30 random subsamples and the 99%-quantile in red (which is small and hardly visible). We observe stable distribution for any sample size. This indicates that only probing 1% of all IPs already yields a stable distribution—even for IWs only present at 0.1% of the hosts. Since the first sample requires knowing the set of all IPs reachable for HTTP/TLS services, we further took 30 random 1% samples of the entire probable address space and arrived at the same result. While probing the entire IPv4 space is possible, it is (given current host configurations) not required to obtain representative IW estimates; reducing the overall footprint by only probing a random subset of 1% suffices. We provide weekly results of 1% scans at https://iw.comsys.rwth-aachen.de.

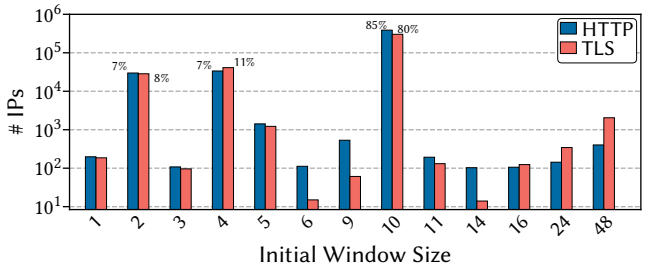

Figure 4: Alexa 1 M IW distribution of HTTP and TLS scan for IWs used by at least 100 hosts.

Since the overall distribution is likely to be impacted by (older) legacy systems, we next focus on assessing popular Internet infrastructures by scanning the Alexa top 1 M list. We show the IW distribution for the Alexa top 1 M list in Figure 4 (note the log-scale). In contrast to probing the entire IPv4 space, the success rate at popular hosts for HTTP increases to 80% yet TLS only gains marginally and succeeds at 85% of the hosts. We can now see that IW 10, as the currently recommended value, dominates the scans with a support of over 85% (80%) for HTTP (TLS). Yet, some hosts are still on IW 2 and IW 4. The IW distribution of TLS hosts is irrespective of their Alexa rank, only IW10 is more pronounced for higher ranked HTTP hosts. We believe that in contrast to the entire IPv4 space, hosts of popular domains are interested in performance optimizations or at least keep their systems up to date. Before we dig deeper into understanding these differences, we next discuss how hosts define their IW by looking at the differences when scanning with a larger MSS before we analyze the data on a per-ASN/service basis.

4.2 IW defined by Byte Limit

Until now, we have only shown the results for our scan with an MSS of 64 B. Only a marginal number (around 1%) of the scanned hosts adjusts their IW according to the announced MSS. Roughly 50% of these hosts send 64 segments, and when doubling the MSS to 128, the segment number halves to 32. These findings suggest that these hosts are configured to use 4 kB as their IW, i.e., the multiple of the MSS and number of segments. We randomly sampled the hosts and manually investigated if we can characterize the hosts. Eight out of ten hosts present a login interface to what seems to be residential access modems from Technicolor in different versions, most modems are hosted by the Mexican ISP Telmex. Among the others, we found publicly accessible power supply monitors that show the same behavior. The remaining 50% of the hosts cannot be clustered into large groups as above. One group that we found by randomly sampling are hosts that seem to adjust their IW in a way that the network MTU is filled, i.e., with a MSS of 64 they send 24 segments and on 128 they send 12, summing up to 1536 B.

4.3 IW Distribution by Network & Service

We start by analyzing the IW usage by *network* type represented by Autonomous Systems (AS). We therefore cluster our data by ASes with similar IW distributions using DBSCAN (wrt. IW 1, 2, 4, 10 and other). The lefthand side of Figure 5 shows particularly large clusters with similar IW distributions that represent a considerable fraction of all scanned IPs (HTTP 49%, TLS 48%). These clusters provide an intuition on per-service IW deployments. Clusters (HTTP

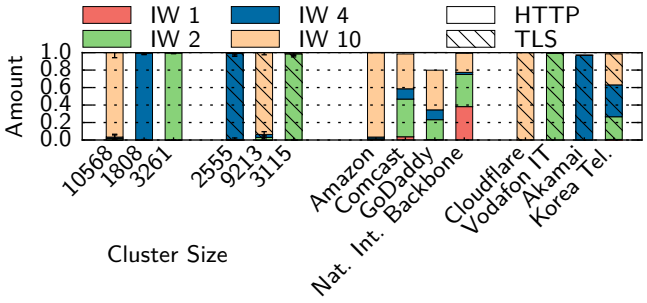

Figure 5: Distribution of IWs per AS. Left, 3 HTTP and 3 TLS clusters of ASs standing out. Right, representatives of these clusters or ASs that do not fit into the clusters.

and TLS) with nearly exclusive use of IW 10 mostly compromise *content provider*, e.g., hosters, cloud provider, and CDNs. ASes with many IW 2 based hosts belong to *ISPs* or in case of HTTP also to *universities*. The cluster for IW 4 is a mixture between *ISPs* and *hosters*. While the HTTP measurement shows more ISPs, the TLS measurement stands out with an AS from Akamai that use IW 4. In case of GoDaddy, 19.8% (32.7%) of the 137 k HTTP (193 k TLS) hosts that were announced by AS26496 (704 prefixes) use an IW of 48. We remark that the number of GoDaddy hosts is $\ll 1\%$ which is why this IW peak is not clearly visible in Figure 3. Unlike our previous observed 4 kB IW hosts, these hosts use a static configuration of IW 48, irrespective of the announced MSS. We found no obvious reason for these comparably large IWs.

We find a diverse picture of different IW configurations. To compare selected content and (residential) access networks, we show their IW distribution in Table 3. Content networks are classified by service-provider IP ranges (e.g., [5]) or the GHost HTTP server string in case of Akamai. Access networks are classified based on their reverse DNS record [23]: *i)* we extract hosts which encode their IP in the reverse DNS record, i.e., 38.6% (62.5%) of all HTTP (TLS) IPs. To exclude server networks (e.g., Amazon and Akamai) we further match their reverse DNS record against a manually created ISP domain list and against a keyword list (e.g., "customer", "dialin"). This way, we classify 16% (18.1%) of all HTTP (TLS) IPs as access. While content providers have largely adopted IW 10, older IW configurations are observed for networks with a potentially high share of legacy devices (e.g., home routers in access networks).

Besides differences between network types, content networks enable further per-service or even per-customer IW configuration (e.g., by Akamai [11]). Assessing these differences is beyond the scope of this paper since it requires presenting valid URLs hosted by Akamai to access different (virtualized) services. While our methodology cannot assess Akamai's different services without this prior knowledge, we used our scanner to manually probe few Akamai HTTP hosted sites and found different IW configurations (e.g., IW 16 and 32). This is a notable case of per-service IW customization.

5 DISCUSSION AND CONCLUSION

This paper presents the first large-scale measurement of TCP's Initial Window (IW) configuration of HTTP/TLS reachable hosts. Since our method does not require prior knowledge of the target host, it is applicable to the Internet at large and enabled us to probe

	HTTP				TLS			
Service	IW1	IW2	IW4	IW10	IW1	IW2	IW4	IW10
Akamai	–	–	–	–	0.0	0.0	100.0	0.0
EC2	0.0	1.8	3.4	94.7	0.2	1.3	2.6	95.8
Cloudflare	0.0	0.0	0.0	100.0	0.0	0.0	0.0	100.0
Azure	0.0	7.8	54.9	37.1	0.1	4.1	73.3	21.9
Access NW	3.5	50.2	20.8	21.7	4.5	17.6	67.1	10.4

Table 3: Per-service IW distribution [%] clustered by IP range (servers) or reverse DNS (access). Dominant IWs highlighted.

reachable hosts within the entire IPv4 address space. In light of the ongoing debate on IW sizes, our study provides an up-to-date view of the current Internet-wide IW configurations documenting the slow adherence to RFC recommended changes of the IW.

The main result of our study is a rather network dependent IW configuration. Since especially service providers can benefit from larger IWs, their IW 10 adoption (or even larger IWs) is high. We also noted service specific customizations, e.g., Akamai enables per-service and even per-customer specific IW configurations. Since these services are virtualized, analyzing such service (not host) -specific configurations requires prior knowledge to present valid host names/URLs—a setting our generalized methodology avoids to be applicable to the Internet at large. Circumventing this limitation by probing selected services with manually curated URL lists thus motivates future work. In contrast, networks with a larger fraction of legacy and other devices show a much lower IW 10 deployment and higher shares of older recommended IW sizes (i.e., 1, 2, 4). In case of Linux, this *can* be caused by outdated systems since IW 10 was enabled in 2011 [16]. Besides these RFC recommended IWs, we observed non-RFC configurations such as much larger IWs.

These observations motivate future work, especially in light of a recent proposal [2] to abandon general IW size recommendations in favor of per-service customized values. While we already observe a diverse landscape of different IW settings, this trend towards customized IWs is likely to continue. Monitoring and better understanding this trend thus motivates future research.

Concerning our methodology, we identified TLS-based scans as a promising alternative to traditional HTTP-based IW inference methods. Our ZMap-based implementation further shows that complex TCP probes going beyond single packet exchanges as needed for port-scans are feasible in a time-efficient manner. The implementation as a ZMap module therefore allows daily snapshots of the Internet. To reduce the footprint of Internet-wide scans, we find that already scanning a random subset of 1% of the IPv4 address space suffices to obtain representative IW distributions. We publish weekly results on these 1% scans on https://iw.comsys.rwth-aachen.de.

ACKNOWLEDGEMENTS

This work has been funded by the German Research Foundation (DFG) as part of project B1 within the Collaborative Research Center (CRC) 1053 – MAKI. We would like to thank the network operators at RWTH Aachen University, especially Jens Hektor and Bernd Kohler at RWTH Aachen ITC. We would like to thank Pascal Hein for enhancing the ZMap module. We further thank the anonymous IMC reviewers and our shepherd Monia Ghobadi for their valuable comments to improve this manuscript.

REFERENCES

[1] Bernhard Ager, Nikolaos Chatzis, Anja Feldmann, Nadi Sarrar, Steve Uhlig, and Walter Willinger. 2012. Anatomy of a Large European IXP. In *Proceedings of SIGCOMM '12*, Helsinki, Finland, August 13–17, 2012. 12 pages.

[2] Mark Allman. 2015. *Removing TCP's Initial Congestion Window.* Internet-Draft draft-allman-tcpm-no-initwin-00.txt. IETF Secretariat.

[3] M. Allman, S. Floyd, and C. Partridge. 1998. *Increasing TCP's Initial Window.* RFC 2414. RFC Editor. 1–14 pages. http://www.rfc-editor.org/rfc/rfc2414.txt

[4] M. Allman, S. Floyd, and C. Partridge. 2002. *Increasing TCP's Initial Window.* RFC 3390. RFC Editor. 1–14 pages. http://www.rfc-editor.org/rfc/rfc3390.txt

[5] Amazon.com, Inc. 2016. Amazon Web Service IP Address Ranges. https://ip-ranges.amazonaws.com/ip-ranges.json.

[6] M. Belshe, R. Peon, and Ed. M. Thomson. 2013. *Hypertext Transfer Protocol Version 2 (HTTP/2).* RFC 7540. RFC Editor. 1–96 pages. http://www.rfc-editor.org/rfc/rfc7540.txt

[7] CDNPlanet. [n. d.]. Initcwnd settings of major CDN providers. https://www.cdnplanet.com/blog/initcwnd-settings-major-cdn-providers/

[8] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis. 2013. *Increasing TCP's Initial Window.* RFC 6928. RFC Editor. 1–24 pages. http://www.rfc-editor.org/rfc/rfc6928.txt

[9] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. 2010. An Argument for Increasing TCP's Initial Congestion Window. *ACM SIGCOMM CCR* 40, 3 (2010), 26–33.

[10] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. 2013. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *Proceedings of USENIX Conference on Security*, Washington, D.C., USA, August 14–16, 2013. 16 pages.

[11] Akamai Community Forum. [n. d.]. Can we change initial CWIN for web experience products like DSA, Ion? https://community.akamai.com/thread/2694.

[12] Janardhan Iyengar, Ian Swett, Ryan Hamilton, and Alyssa Wilk. 2016. *QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2.* Internet-Draft draft-tsvwg-quic-protocol-02. Internet Engineering Task Force. https://tools.ietf.org/html/draft-tsvwg-quic-protocol-02 Work in Progress.

[13] V. Jacobson. 1988. Congestion Avoidance and Control. In *Proceedings of SIGCOMM '88*, Stanford, California, USA, August 16–18, 1988. 16 pages.

[14] A. Langley, N. Modadugu, and B. Moeller. 2014. *Transport Layer Security (TLS) False Start.* Internet-Draft draft-bmoeller-tls-falsestart-01.txt. IETF Secretariat.

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

[16] David S. Miller and Nandita Dukkipati. 2011. TCP: Increase the Initial Congestion Window to 10. http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=442b9635c569fef038d5367a7acd906db4677ae1.

[17] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. 2014. The Cost of the "S" in HTTPS. In *Proceedings of CoNEXT '17*, Sydney, Australia, December 02–05, 2014. 8 pages.

[18] Jitendra Padhye and Sally Floyd. 2001. On Inferring TCP Behavior. In *Proceedings of SIGCOMM '01*, San Diego, California, USA, August 27–31, 2001. 12.

[19] W. Michael Petullo, Xu Zhang, Jon A. Solworth, Daniel J. Bernstein, and Tanja Lange. 2013. MinimaLT: Minimal-latency Networking Through Better Security. In *Proceedings of CCS '13*, Berlin, Germany, November 04–08, 2013. 14 pages.

[20] Feng Qian, Alexandre Gerber, Zhuoqing Morley Mao, Subhabrata Sen, Oliver Spatscheck, and Walter Willinger. 2009. TCP Revisited: A Fresh Look at TCP in the Wild. In *Proceedings of IMC '09*, Chicago, Illinois, USA, November 04–06, 2009. 14 pages.

[21] Sivasankar Radhakrishnan, Yuchung Cheng, Jerry Chu, Arvind Jain, and Barath Raghavan. 2011. TCP Fast Open. In *Proceedings of CoNEXT '11*, Tokyo, Japan, December 06–09, 2011. 12 pages.

[22] Jan Rüth. 2017. ZMap and Modules. Retrieved 10.08.2017 from https://github.com/COMSYS/zmap

[23] Quirin Scheitle, Oliver Gasser, Patrick Sattler, and Georg Carle. 2017. HLOC: Hints-Based Geolocation Leveraging Multiple Measurement Frameworks. In *Proceedings of TMA '17*, Dublin, Irland, June 21–23, 2017. 9 pages.

[24] W. Stevens. 1997. *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms.* RFC 2001. RFC Editor. 1–6 pages. http://www.rfc-editor.org/rfc/rfc2001.txt