CDN Judo: Breaking the CDN DoS Protection with Itself

Run Guo*§, Weizhong Li*§, Baojun Liu*, Shuang Hao†, Jia Zhang*¶✉,
Haixin Duan*¶✉, Kaiwen Shen*, Jianjun Chen‡, Ying Liu*¶
* Tsinghua University,
{gr15, lwz17, lbj15, skw17}@mails.tsinghua.edu.cn, {zhangjia, liuying}@cernet.edu.cn, duanhx@tsinghua.edu.cn
† University of Texas at Dallas, shao@utdallas.edu
‡ ICSI (International Computer Science Institute), jianjun@icsi.berkeley.edu
¶ BNRist (Beijing National Research Center for Information Science and Technology)

***Abstract*—A content delivery network (CDN) improves the accessing performance and availability of websites via its globally distributed network infrastructures, which contributes to the thriving of CDN-powered websites on the Internet. Because CDN-powered websites normally operate important businesses or critical services, attackers are mostly interested in taking down these high-value websites, to achieve severe damage with maximum influence. Because the CDN absorbs distributed attacking traffic with its massive bandwidth resources, it is commonly believed that CDN vendors provide effective DoS protection for the CDN-powered websites.**

However, we reveal that implementation or protocol weaknesses in the forwarding mechanisms of the CDN can be exploited to break this CDN protection. By sending crafted but legal requests, an attacker can launch an efficient DoS attack against the website *origin* behind it. In particular, we present three CDN threats in this study. By abusing the HTTP/2 request-converting behavior and HTTP pre-`POST` behavior of a CDN, an attacker can saturate the *CDN–origin* bandwidth and exhaust the connection limits of the *origin*. What is more concerning is that some CDN vendors use only a small set of traffic forwarding IPs with lower IP-churning rates to establish connections with the *origin*. This characteristic provides a great opportunity for an attacker to effectively degrade the global availability of a website just by cutting off specific *CDN–origin* connections.

In this work, we examine the CDN request-forwarding behaviors across six well-known CDN vendors and perform real-world experiments to evaluate the severity of the threats. Because the threats are caused by flawed trade-offs made by the CDN vendors between usability and security, we discuss possible mitigation and received positive feedback after responsible disclosure to the aforementioned CDN vendors.

I. Introduction

Through the deployment of massive surrogate servers in different geographical locations, often across multiple Internet backbones, a content delivery network (CDN) works as a geographically distributed network, supporting websites into having high capacities in terms of both computational resources and network bandwidth. Because of its traffic offloading benefits and global accessibility, the CDN has become an indispensable part of the Internet ecosystem. CDN vendors have also been advertising their capability to protect against DoS attacks, contributing to the successful expansion of CDNs over the Internet, where increasingly more websites are being deployed behind CDNs. For example, more than 50% of the Alexa 1K and more than 35% of the Alexa 10K websites are deployed behind CDNs [20].

However, in this paper, by empirically exploring the forwarding behaviors of six CDNs, we reveal that the CDN itself can be abused to attack the *origin* (website server) behind a CDN. By sending crafted but legal requests to a CDN, an attacker can initiate a DoS attack against the *origin*, breaking the CDN DoS protection. In short, our work reveals the following three threats:

- **HTTP/2 Bandwidth Amplification Attack.** We find that CDNs support only HTTP/2 in the *client–CDN* connection, and thus an attacker can abuse the HTTP/2–HTTP/1.1 converting behavior of a CDN to launch a bandwidth amplification attack against the *origin* (*e.g.*, reaching an amplification factor of 132 for Cloudflare). We analyze the HTTP/2-introduced HPACK compression mechanism, which contributes to the threat, and also reveal that the concurrent streams and Huffman encoding of HTTP/2 can be abused to further elevate the bandwidth amplification factor.
- **Pre-`POST` Slow HTTP Attack.** We find that three out of the six CDNs we analyze in this study start forwarding HTTP `POST` requests just upon receiving the `POST` header, without waiting for the whole `POST` message body. This pre-`POST` behavior can be abused to exhaust the connection limit of the *origin* and starve other legitimate user requests, resulting in a slow HTTP DoS attack against the *origin*. Even worse, the HTTP/1.1 `POST` forwarding and HTTP/2 `POST` forwarding behaviors of these CDNs are both susceptible to this threat.
- **Degradation-of-Global-Availability Attack.** By sending requests to the global surrogate IPs (ingress IPs) of each CDN to simulate global client accessing, we perform a large-scale measurement of the distribution of the traffic-forwarding IPs (egress IPs) of each CDN. Results show that CDNs will assign a small set of egress IPs to access the *origin*, presenting a lower IP-churning rate. Therefore, this

§ Equal contribution joint first authors.
✉ Corresponding author.

Network and Distributed Systems Security (NDSS) Symposium 2020
23-26 February 2020, San Diego, CA, USA
ISBN 1-891562-61-4
https://dx.doi.org/10.14722/ndss.2020.24411
www.ndss-symposium.org

characteristic can be leveraged by an attacker to efficiently degrade the availability of a CDN-powered website just by cutting off one or a small set of *CDN–origin* connections, thus preventing most global clients from accessing the services of the website. For example, with MaxCDN (which has now been acquired by StackPath [53]), if just one *CDN–origin* connection is cut off, more than 90% of global accesses are stopped from fetching resources from the *origin* behind the CDN.

In summary, we focus on how to break CDN security protection, which is assumed to be trustworthy by many websites. By performing empirical security analysis on the under-studied CDN back-to-*origin* connections, we explore the feasibility of abusing the forwarding behaviors of a CDN to launch DoS attacks against CDN-powered websites. The HTTP/2 amplification attack is built on a previous study [7], but whether it applies to CDN-protected websites has been unexplored, and thus we further present a real-world evaluation of the HTTP/2 amplification attack through CDN platforms, with an in-depth analysis on the HPACK mechanism. Furthermore, we find vulnerable HTTP `POST`-forwarding strategies of CDN vendors, which can be exploited to launch pre-`POST` slow HTTP attacks. Lastly, based on our large-scale measurements of CDN IP distribution, we exploit the low IP-churning rates of CDNs, which can be used to launch a degradation-of-global-availability attack. Our results show that these attacks pose a severe threat to CDN-powered websites.

DoS attacks are well known to cause severe damages against websites, resulting in losses in terms of both money and trust among the customers of these websites [22]. Because CDN-powered websites normally operate important business services (*e.g.*, banks, online shopping stores, news servers), a practical DoS attack against CDN-hidden *origins* can significantly disrupt the businesses and reputations of these websites [58].

Our work can help CDNs to raise security awareness and to enforce stricter secure validation that would result in the improved security of such critical Internet infrastructure. We have responsibly disclosed our findings to all affected CDNs and have received positive feedback for our work.

Roadmap. In Section II, we first present a background on CDNs and analyze the attack surface. We then sequentially expound in Sections III to V on the three threats that have been introduced earlier. Possible mitigation are discussed in Section VI, related works are described in Section VII, and our conclusion is presented in Section VIII.

II. Background and Threat Model

A. Background

Content Delivery Network. CDNs are widely used to improve the performance and security of websites. For a CDN-powered website, the CDN speeds up the connection performance by using request-routing mechanisms (*e.g.*, Anycast or DNS-based) [55] that redirect the web requests of clients to geographically distributed CDN surrogates (CDN ingress IPs).

Upon receiving a web request, a CDN surrogate first examines the HTTP header fields, especially the `Host` and the `URI` header fields. If the requested web resources are already cached in the CDN, the surrogate serves the contents directly to the client. Otherwise, the surrogate will forward the requests to the *origin* via egress IPs, as shown in Fig. 1. As a result, the CDN separates the traditional end-to-end connection into two stages, *i.e.*, *client–CDN* connection and *CDN–origin* connection, working as a man-in-the-middle between the client and the *origin*.

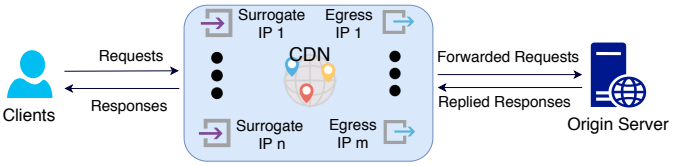

Fig. 1: *CDN forwards requests and responses between client and origin.*

Thus, a CDN, from its point of view, first has to work as a protocol converter when the protocol of the *client–CDN* connection differs from the protocol used in the *CDN–origin* connection, *e.g.*, the CDN converts *client–CDN* HTTPS connections to *CDN–origin* HTTP connections, as in [41]. Second, the CDN aims to speed up the request delay of the end user, and thus the CDN has to optimize the back-to-*origin* forwarding of the request to be as fast as possible. Lastly, to improve the CDN cache-hit ratio and reduce back-to-*origin* forwarding, the CDN may add an extra caching layer to cache the resource contents of the websites for its global surrogates [34]. In the following sections, we shall reveal how these three CDN features are able to lead to our attacks.

Request-Routing Mechanism. The request-routing mechanism is critical for a CDN to provide the optimal CDN surrogates for processing requests. However, this mechanism can be bypassed if the surrogate IPs are pre-known; normal users can directly send requests to a chosen surrogate IP without the request-routing stage, which otherwise maps the website domain name with the CDN surrogates. For example, Holowczak *et al.* has shown that CDN-powered websites can be accessed from *arbitrary* CDN surrogates [30].

HTTP/2 Protocol in CDN. The HTTP/1.1 protocol builds the foundation of the World Wide Web. However, the repeated redundant HTTP headers in each request and response wastes network bandwidth and slows down connection performance. Therefore, HTTP/2 was released to address those issues: header compression reduces the unnecessary network traffic in HTTP/1.1, and multiplexing streams allows multiple requests in a single TCP connection [8], [49]. Currently, almost all CDNs claim that they support the HTTP/2 protocol [60].

Brief Comparison of CDN Vendors. The global CDN service market is worth billions of dollars and is growing at an increasingly fast rate, with several CDN vendors competing in this booming market. According to CDN market share reports [18], [19], Akamai, CloudFront, Cloudflare, and Fastly are the key players in this market [33], and thus these vendors should naturally be in the scope of most research on CDN. However, because Akamai provides CDN services to enterprise customers only, it is not included in our study.

Thus, for this study, we choose six CDN vendors (CloudFront, Cloudflare, and Fastly, which are three of the key players mentioned earlier, together with CDNSun, KeyCDN,

and MaxCDN) that provide free-trial account registrations to individual users. Among these six CDN vendors, five require email registration only, and only CloudFront requires an extra credit card verification. From the point of view of attackers, these kinds of CDN vendors, that do not require stringent identity verification, enable attackers to reveal and exploit specific CDN forwarding behaviors, without exposing their sensitive personal information. Furthermore, these six CDN vendors involve two primary request-routing mechanisms: Cloudflare and MaxCDN use Anycast routing, whereas the other four CDNs use DNS mapping, which helps to increase our research coverage. In the following sections, although we explore only the feasibility of our attacks against these six CDN vendors, we believe that these attacks are also applicable to other CDN vendors not included in this study.

B. Threat Model

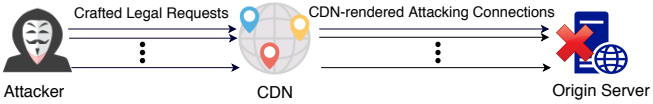

Fig. 2: *Launching a DoS attack against a CDN-hidden origin.*

In general, websites employ CDNs to improve their security and global availability. CDNs normally provide web application firewall (WAF) services to normalize requests to the website *origins*. Furthermore, CDNs can absorb distributed denial-of-service (DDoS) attacks by leveraging a large number of geo-distributed surrogates. Lastly, by hosting on CDNs, websites can hide their "true" *origin* IP addresses away from potential attackers.

In this study, we assume an attacker, as a normal client, is able to craft malicious but legal requests to the CDN. We also assume that the victim website is being hosted on the CDN (or being unwittingly hosted on the CDN by a malicious CDN customer, further explained in Section VI). Here, through an empirical study, we aim to discover some specific but fundamental CDN characteristics that can be abused. In particular, if the forwarding mechanisms of a CDN can be abused, an attacker may able to manipulate the *CDN–origin* connections. As a result, these malicious connections may exhaust the limited network resources of the *origin*, resulting in a DoS attack against the *origin*, as shown in Fig. 2.

III. HTTP/2 Bandwidth Amplification Attack

Up until now, from our experiments, we find that in their *client–CDN* and *CDN–origin* connections, CDNs support HTTP/2 only in *client–CDN* connection. Thus, when receiving an HTTP/2 request, a CDN has to convert the HTTP/2 request into an HTTP/1.1 request, which could introduce new attacking vectors during the protocol conversion process. In this section, by further exploring the protocol-converting behaviors across the six CDNs, we reveal that all six CDNs can be leveraged to launch a bandwidth amplification attack against the *origins* of the websites that they are servicing.

A. Attack Surface Analysis

Half-Done HTTP/2 Support. Almost all CDN vendors claim that they currently support HTTP/2 [60]. However, because a CDN has to maintain both the *client–CDN* connection and *CDN–origin* connection, the HTTP/2 forwarding behaviors have not yet been studied in detail. Here, we first explore the HTTP/2 support behavior of a CDN by setting up the *origin* of our website as an HTTP/1.1-only server, as an HTTP/2-only server, and as an HTTP/1.1&HTTP/2 server. We then use the tool *curl* as a client to access the CDN service in HTTP/2 protocol.

TABLE I: *CDN–origin* protocol. CDNs support HTTP/2 in client-facing connections but use only HTTP/1.1 to connect to the *origin*.

	client–CDN Protocol	CDN–origin Protocol (HTTP/1.1 origin)	CDN–origin Protocol (HTTP/2 origin)	CDN–origin Protocol (HTTP/1.1-2 origin)
CloudFront	HTTP/2	HTTP/1.1	HTTP/1.1	HTTP/1.1
CloudFlare	HTTP/2	HTTP/1.1	HTTP/1.1	HTTP/1.1
CDNSun	HTTP/2	HTTP/1.1	HTTP/1.1	HTTP/1.1
Fastly	HTTP/2	HTTP/1.1	HTTP/1.1	HTTP/1.1
KeyCDN	HTTP/2	HTTP/1.1	HTTP/1.1	HTTP/1.1
MaxCDN	HTTP/2	HTTP/1.1	HTTP/1.1	HTTP/1.1

Experiments have revealed that, as shown in Table I, CDNs support HTTP/2 in *client–CDN* connection but use **only** HTTP/1.1 in the *CDN–origin* connection, even when the *origin* supports HTTP/2. Consequently, these CDNs have to convert web requests between HTTP/2 and HTTP/1.1 protocols, which may introduce new security threats. Even worse, as shown in Table II, these CDNs, except Fastly, turn on HTTP/2 *client–CDN* connection support by default for their customer websites, directly exposing their customer websites to possible protocol conversion threats. Furthermore, the resulting severity increases because three of the CDNs (Cloudflare, CDNSun, and KeyCDN) do not even provide an option to turn off such HTTP/2 support.

TABLE II: HTTP/2 support statuses of the CDNs included in this study. Five of the six CDNs enable HTTP/2 support by default for their customer websites.

	CloudFront	Cloudflare	CDNSun	Fastly	KeyCDN	MaxCDN
HTTP/2 Support	Default On Configurable	Default On	Default On	Default Off Configurable	Default On	Default On Configurable

Primer on HTTP/2. The primary goals of HTTP/2 are to reduce latency and minimize protocol overhead. Primarily, the HTTP/2 protocol supports multiple concurrent bidirectional streams within a single HTTP/2 connection, thus reducing unnecessary TCP handshake processes and supporting full request and response multiplexing [8]. For example, in a *client–CDN* connection, a client makes one HTTP/2 connection with the CDN, using two streams to request resources through "path1" and "path2," as shown in Fig. 3.

In HTTP/1.1, header fields are not compressed. Because web pages have grown to require dozens to hundreds of requests, the redundant header fields in these requests unnecessarily consume bandwidth. Therefore, in HTTP/2, HPACK header compression is introduced primarily to reduce unnec-

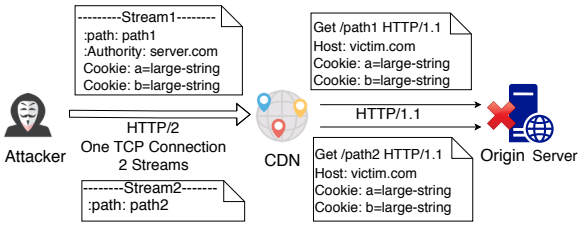

Fig. 3: *HTTP/2-HTTP/1.1 conversion has to decompress and expand HTTP/2 requests, resulting in bandwidth amplification.*

essary network traffic caused by the repeated request and response headers in HTTP/1.1 [49].

According to the HPACK mechanism, within the *client–CDN* connection, both the client and CDN (as an HTTP/2 server) maintain an indexed dynamic table of previously seen header values, and subsequent repeated header fields are substituted as an index referencing a value in the table. Because many header fields, *e.g.*, `:authority`, `cookie`, and `user-agent` are repetitive, this mechanism has a very high table-hitting ratio. Thus, instead of full header fields, the substituted indexes are transmitted in the network, reducing the transferred bytes.

Accordingly, when the client opens a second stream to send another "path2" request, the repeated header fields, such as `cookie`, are substituted as indexes (and thus these fields are not shown in "stream2" of Fig. 3). These mechanisms greatly reduce the header overhead and improve transfer performance.

Attack Principle. When a CDN forwards these requests to the *origin*, all header fields indexed in HTTP/2 must be expanded into HTTP/1.1 requests, leading to bandwidth amplification. As shown in Fig. 3, this mechanism results in two large-sized HTTP/1.1 requests with the same large-sized `cookie`, which leads to a bandwidth amplification in the *CDN–origin* connection, with an amplification ratio of almost 2. An unsymmetrical bandwidth-consuming attack that takes advantage of this mechanism was evaluated by Beckett *et al.* on an experimental testbed with proxy software *Nginx* and *nghttp2* [7], but to our knowledge, no real-world experiments on this kind of attack have been performed yet.

As we can see, within one HTTP/2 connection, the amplification ratio is linear with respect to the number of concurrent streams. The maximum values for concurrent streams are negotiated when an HTTP/2 connection is established. We measure the stream limits of the CDNs and list them in Table III. Across all six CDNs, the maximum allowed concurrent streams are all bigger than 100 (the recommended value in the RFC [8]).

TABLE III: Limits set by CDNs on HTTP/2 streams.

	CloudFront	Cloudflare	CDNSun	Fastly	KeyCDN	MaxCDN
Max Concurrent Streams	128	256	128	100	128	100
Dynamic Table Size	4KB	4KB	4KB	4KB	4KB	4KB
Max Entry Size	3072B	3072B	3072B	3072B	3072B	3072B

Therefore, for an attacker to achieve the maximum amplification ratio in *CDN–origin* connections, crafted attacking HTTP/2 requests can all use a header field with the same large-sized value, *e.g.*, `cookie` with a large-sized value, given that it is widely used in HTTP requests. Besides the `cookie` field, the attacker can also use other header fields defined in the HTTP/2 protocol, such as `user-agent` and `referer`, which are also forwarded to the *origin*. The size of the header field value is limited by the size of the indexed dynamic table, which is also negotiated during the HTTP/2 connection. As shown in Table III, the maximum table entry size across the CDNs is 3072 B, and the table size is 4 kB. Thus, crafted attacking HTTP/2 requests can use two header fields to fill the indexed table, resulting in the converted HTTP/1.1 *CDN–origin* requests to have the maximum size.

B. Real-World Attack Analysis

Experiment Setup. Based on the previously explained analysis, we further evaluate the severity of such an amplification attack across the six CDNs. After deploying an *Apache* server behind each CDN, we initiate an HTTP/2 connection to each of the six CDNs to send attacking requests which are crafted as

```
:path: /?<random_string>          (or /)
:scheme: https
:authority: victim.com
:method: GET
cookie: A=X...X  (a large-sized string)
cookie: B=X...X  (a large-sized string)
```

To achieve the maximum amplification ratio, we use two `cookie` fields with large-sized strings to fill the 4 kB HTTP/2 dynamic table. Given that the maximum table entry size is 3072 B, the lengths of two `cookie` values are calculated by subtracting additional overhead bytes from the total 4 kB dynamic table size. The additional overhead bytes are determined by table entry overhead and other header field values, *e.g.*, `:authority` and `user-agent`. These two cookies stay the same in all concurrent streams, thus they will be transferred in the same way as indexes except for the first stream. Note that we actually use two types of `:path` header field values to evaluate the amplification ratio; the reason for this will be discussed later in this section.

In our experiments, to explore the impact of concurrent streams on the amplification ratio, we change the number of concurrent streams within one HTTP/2 connection and use *tcpdump* to capture the traffic in both the *client–CDN* connection and *CDN–origin* connection to evaluate the amplification factor.

Experiment Results. According to Fig. 4, when the number of concurrent streams grows, the bandwidth amplification ratio also grows. As shown in Fig. 5, when the number of concurrent streams grows, the packet amplification ratio also grows. When the concurrent streams reach the maximum allowed number for one HTTP/2 connection, the amplification ratio reaches the maximum. When the stream number grows beyond the maximum allowed number for one HTTP/2 connection, our HTTP/2 client has to wait for the previous streams to close, and the packets ratio drops, as shown in Fig. 5. Meanwhile, the bandwidth amplification ratio fluctuates after the maximum number of allowed concurrent streams is reached.

The bandwidth amplification ratios are summarized in Table IV; we will illustrate the difference between the 2nd

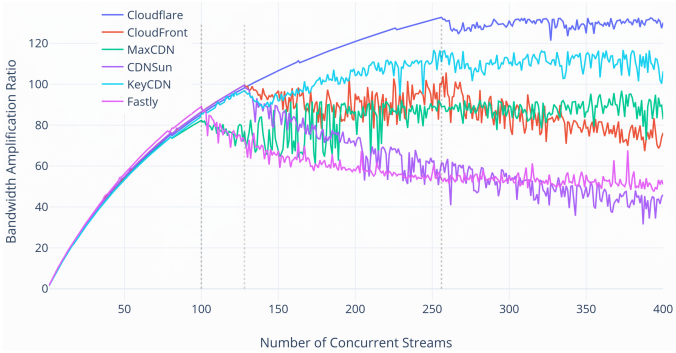

Fig. 4: *Bandwidth amplification ratio when the number of concurrent streams increases (*`:path: /?random_string`*).*

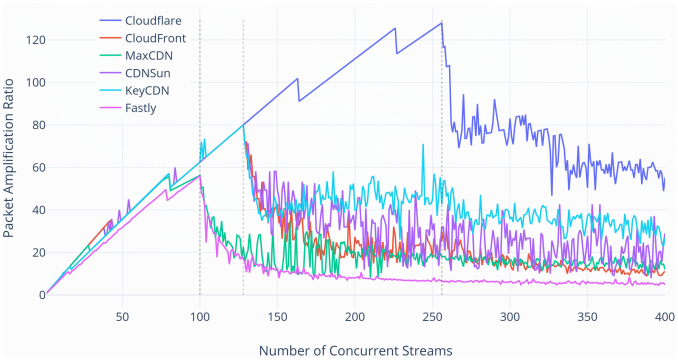

Fig. 5: *Packets amplification ratios when the number of concurrent streams increases (*`:path: /?random_string`*).*

and 3rd rows later. We can see that this HTTP/2–HTTP/1.1 conversion threat is realistic; it can break the CDN protection and cause a severe DoS attack against the *origin*.

Analysis of Amplification Factors. From the given illustration, we can see that the bandwidth amplification ratio is determined primarily by the number of concurrent streams. From further analysis of the HTTP/2 specification, we also find other influencing factors, such as the Huffman encoding and the `:path` header field, that contribute to the bandwidth amplification ratio.

- *Huffman encoding:* In the HPACK compression mechanism, Huffman encoding is applied to further compress the header values. This Huffman code is statistically generated for HTTP headers, wherein ASCII digits and lowercase letters are given shorter encoding [49]. The shortest encoding for one byte is 5 bits long; therefore, the highest compression ratio achievable for one byte is 8 bits : 5 bits Huffman code (37.5% smaller). Thus, besides the concurrent streams, Huffman encoding can also be abused to maximize the amplification ratio. Because HTTP/2 headers are firstly compressed with Huffman encoding and will be decompressed in the *CDN–origin* connections, the resulting HTTP/1.1 headers will be nearly $8/5 = 160\%$ larger in size.

Therefore, in our experiments, the two cookie values are composed of the characters `0`, `1`, `2`, `a`, `c`, `e`, `i`, `o`, `s`, or `t`, which have the shortest Huffman encoding (5 bits) defined in the RFC [49]. With Huffman encoding, we achieve the amplification ratios listed in Table IV.

TABLE IV: Maximum amplification ratios across the CDNs.

	CloudFront	Cloudflare	CDNSun	Fastly	KeyCDN	MaxCDN
Streams	128	256	128	100	128	100
Bandwidth Ratio (:path: /?random_string)	99.6	132.6	99.5	89.0	96.8	82.3
Bandwidth Ratio (:path: */*)	116.9	166.1	118.7	97.9	105.5	94.7

- *`:path` header field*: In our experiments, we also find that the `:path` header field contributes to the amplification ratio. If we use `:path: /` directly in all of our attacking requests, given that it is a predefined index in the static table[1], only 1 B needs to be transmitted in all of the HTTP/2 concurrent streams. However, in the World Wide Web, URLs can be of varying lengths, and an attacker normally uses random URLs to bypass CDN caching or WAF rules. Therefore, we also append different random strings to the `:path` header field in each HTTP/2 stream, in the form `:path: /?random_string`.

TABLE V: Length of the `:path` header field during HTTP/2 and HTTP/1.1 conversion.

HTTP/2 client–CDN Connection		HTTP/1.1 CDN–origin Connection	
Header Field	Transmission Length	URL	Transmission Length
:path: /	1 bytes	/	1 bytes
:path: /?xxxyy	8 bytes	/?xxxyy	7 bytes

As shown in Table V, the `:path: /` header field is also converted into 1 B in each resulting HTTP/1.1 connection. When we use a random value `/?random_string` in the `:path` header field in each HTTP/2 request, the random value in that field is a non-repetitive value and is therefore not present in the dynamic table. According to the HPACK mechanism, the value will be encoded in either its raw form or using the Huffman encoding form (the shorter of the two). In Table V, we can see that `:path: /?xxxyy` consumes 8 B in HTTP/2 (1 B for the index of `:path` field, and 7 B for `/?xxxyy` [2]), and the converted URL in each HTTP/1.1 request will be 7 B.

In our experiments, we send these two types of `:path` headers to evaluate the bandwidth amplification ratio and obtain different results, as listed in Table IV. The reason for these differences is that when the number of concurrent HTTP/2 streams grows, the length of the `:path` header field begins to influence the amplification ratio. In our experiments, when the `:path: /` form is used, the network traffic of the attacker–CDN connection *front-end traffic (FB)* is in the order of thousands, *e.g.*, 5,000 bytes per second, whereas the network traffic of the *CDN–origin* connection *back-end traffic (BB)* is in the order of millions, *e.g.*, 600,000 bytes per second. The amplification ratio is BB/FB. On the other hand, when the `:path: /?xxxyy` form is used, and we send n (*e.g.*, 100, 128, or 256, *i.e.*, the maximum values in Table III) concurrent HTTP/2 streams, the network traffic of the attacker–CDN connection, compared with for the '`:path: /` form, will be

[1]According to the HTTP/2 specification, the HPACK mechanism uses an additional static table to predefine common header fields associated with frequently occurring values, *e.g.*, `:path: /` is predefined in the indexed table as index 4 [49].

[2]We generate a different random string in each request. Here, we neglect the chances that the Huffman encoding may compress the random string to shorter than 7 B.

$(FB + 7n)$ (*n* HTTP/2 streams, $8 - 1 = 7$ B larger in each stream), and the network traffic of the *CDN–origin* connections will be $(BB + 6n)$, (*n* HTTP/1.1 connections, $7 - 1 = 6$ B larger in each connection). Thus, for the `:path: /?xxxyy` form, the amplification ratio will be $(BB + 6n)/(FB + 7n)$.

As we have illustrated, ***FB*** is in the order of thousands, whereas ***BB*** is in the order of millions. Therefore, we have the following mathematical inequality:

$$\frac{BB}{FB} > \frac{BB + 6n}{FB + 7n}. \quad (1)$$

For example, assuming $FB = 5,000$ and $BB = 600,000$ for simplicity, when we use 128 or 256 for *n* (the number of concurrent streams), the inequality becomes

$$\frac{BB}{FB} = \frac{600000}{5000} > \frac{600000+128*6}{5000+128*7} = 101.9 > \frac{600000+256*6}{5000+256*7} = 88.6. \quad (2)$$

Therefore, we can see that, to achieve the maximum amplification ratio, the HTTP/2 attacking requests should be specially crafted to use the HPACK indexing mechanism as much as possible.

Summary. For the attack, we conducted a controlled experiment to obtain the network traffic amplification ratio by establishing just one HTTP/2 connection with one CDN node. However, from the perspective of an attacker as a client, he can initiate thousands of HTTP/2 connections with different CDN nodes (*e.g.*, we found 128,906 CloudFront IPs, which can be used for the attack; please refer to Table IX for the number of IPs of other CDNs). According to the amplification ratio we obtained, the network bandwidth of *CDN–origin* connection can be seriously consumed, adversely influencing the performance of the *origin*.

Given that HTTP/2 support is turned on by default across five of these six CDNs, and cannot even be turned off across three of the CDNs, we can see that this threat is severe and affects all websites hosted on these CDNs.

IV. Pre-POST Slow HTTP Attack

In this section, we introduce the *pre-POST slow HTTP attack*, which leverages CDN infrastructure to perform a DoS attack against the *origin*. Compared with traditional DoS attacks that rely on massive bots [1], [43], [51], this attack is stealthier and harder for the *origin* to defend against, because the crafted requests are legal and are initiated from the CDN.

A. Attack Surface Analysis

The pre-POST slow HTTP attack aims to exhaust the connection limits of the *origin* and starve other legitimate user requests. To the *origin*, the attack acts the same as a traditional slow POST attack [23], [55]. Normally, as the CDN decouples the *client–CDN* (including attacker–CDN) and *CDN–origin* connections, the CDN naturally defends against traditional slow POST attacks. However, with experiments, we find that three out of the six CDNs start forwarding HTTP POST requests just upon receiving the POST header, without waiting for the whole POST message body. We reveal that this pre-POST behavior empowers an attacker to keep the *CDN–origin* connections to remain open as long as possible, thus allowing the attacker to exhaust the connection limits of the *origin*. In this section, we first review the traditional slow HTTP attack, and then we further analyze how three out of the six CDNs are susceptible to this pre-POST threat.

Primer on Slow HTTP DoS Attack. According to the Kaspersky Q4 2018 Intelligence Report [47], the total duration of HTTP-related attacks has been growing, accounting for about 80 percent of DDoS attack time for the whole year. This report finding reveals that attackers are turning to sophisticated, mixed HTTP attack techniques, such as slow HTTP DoS attacks.

Compared with brute-force flooding attacks, a slow HTTP DoS attack is stealthier and more efficient. The slow HTTP DoS attack takes advantages of the HTTP protocol having been designed to keep the connection open until the receiving of data is finished [23], [55]. Therefore, different stages of the request flow can be abused to launch slow HTTP DoS attacks. A slow Header attack sends the partial header, a slow Read attack intentionally receives response data slowly, and a slow POST attack sends the posted data at an alarmingly slow rate. All these attacks aim to keep massive connections with the target server for as long as possible, leading to an exhaustion of the concurrent connections of the target and starving other normal user requests [29], [48], [56].

Attack Principle. Generally, to prevent unavailability due to DoS attacks, the CDN decouples attacker–CDN and *CDN–origin* connections and absorbs any flooding traffic. However, the applicability of slow HTTP DoS attack against CDN-powered websites remains under-studied.

With our further analysis and real-world experiments, we find that each of the six CDNs forwards requests only until it receives the full HTTP header, and is therefore able to defend against slow Header attacks. Furthermore, when forwarding an HTTP GET request, the *CDN–origin* transmission is independent of the attacker client–CDN transmission; therefore, the CDN is able to stop slow Read attacks.

However, we find that CDNs present two different POST-forwarding behaviors. When a CDN receives a POST request for the *origin*, the CDN faces the choice of when to forward the POST request to the *origin*. For simplicity, the CDN can forward the POST request only after it finishes receiving the whole POST message. However, the POST request may contain a large-sized message body, which would take a long time to receive and therefore delay the request forwarding. The CDN can also start forwarding the POST request just upon finishing receiving the POST request header and then sequentially forward the subsequently received POST message within the same HTTP connection. This **pre-POST-forwarding** behavior can certainly facilitate the *origin* into receiving the POST request earlier; however, it also enables an attacker to keep the *CDN–origin* connections open for as long as possible.

B. Real-World Attack Analysis

Experiment Setup. In our experiment, we set up a self-built *Apache* web-server and deploy it as a website *origin* behind the six CDNs, one at a time. The concurrent connections limit of the *Apache* web server is configured with a default value of 1000 [3].

From the view of an attacker, we craft POST requests to explore the request-forwarding behaviors of the CDNs. In particular, to POST a large message, the attacker can specify the size directly in the Content-Length header field, or use Chunked-Encoding to send dynamically generated data, both aiming to send the POST message slowly. Here, for simplicity, we specify the exact size of the HTTP message body with the Content-Length field, and the POST message body is sent quite slowly, taking 300 s to finish transmission.

```
POST /login.php?<random_string> HTTP/1.1
Host: www.victim.com
Content-Length: 300

0101..... (300 bytes, 1 byte sent per second)
```

At the same time, at the website *origin*, we use the tool *tcpdump* to capture the timestamp (relative to our request sending time) upon receiving the CDN-forwarded HTTP POST request, and how long the *CDN–origin* connection is kept open. After sending 1000 concurrent POST requests, and repeating this procedure for 30 times, we obtain the averaged results shown in Table VI.

TABLE VI: Time data from sending slow POST requests (lasting 300 s). Three CDNs start forwarding POST requests as soon as they receive the POST header.

	CloudFront	Cloudflare	CDNSun	Fastly	KeyCDN	MaxCDN
Request Receiving Time	**0.87s**	300.29s	299.92s	**0.55s**	299.79s	**0.74s**
Connection Keep-open Time	**298.89s**	0.12s	0.34s	**299.32s**	0.37s	**15.01s**

We can see that CloudFront and Fastly start to forward POST requests as soon as they receive the forwarding request header, whereas CDNSun, KeyCDN, and Cloudflare start to forward a POST request only after receiving the whole message. MaxCDN also starts to forward POST requests 0.74 s later but aborts the connection when the kept-open time exceeds 15 s.

Apparently, for CloudFront, Fastly, and MaxCDN, the kept-open time of the *CDN–origin* connection depends on the kept-open time of the *client–CDN* connection, which is directly under the control of the client, and thus of a potential attacker. Therefore, this pre-POST-forwarding behavior can be leveraged to launch a slow HTTP DoS attack: an attacker can establish and maintain hundreds or even thousands of these POST connections concurrently, leveraging the CDN (and thus adversely affecting the origin). It will quickly exhaust all the connection resources of the *origin* and starve other normal requests, breaking the DoS protection given by the CDN.

Experiment Results. Further, we evaluate such pre-POST attack against our self-built *origin* web server (with a connection limit of 1000), through CloudFront, Fastly, and MaxCDN for 300 s. From another vantage point, as a normal client, we periodically measure the *client–CDN–origin* request delay every 5 s to probe whether the connection resources of the *origin* are exhausted or not.

For CloudFront and Fastly, to exhaust the 1000-connection limit of our *origin*, we concurrently send 1100 slow POST requests to the CDN, as shown in Fig. 6. At the *origin*, the

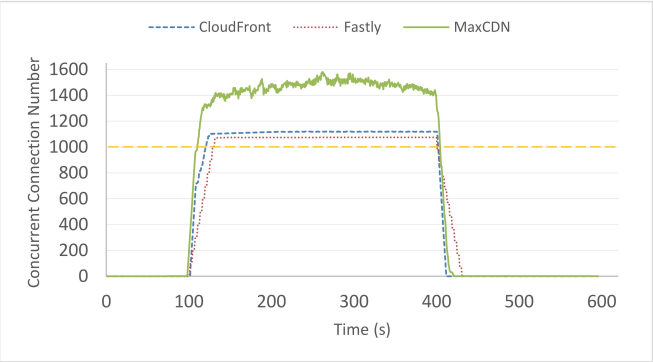

Fig. 6: *Establishing more than 1000 connections, from 100 s to 400 s.*

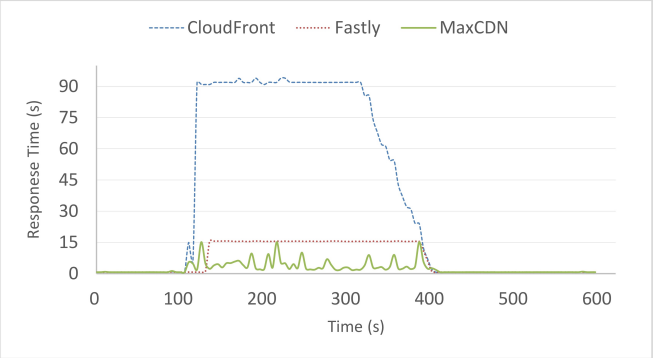

Fig. 7: *Response time of a normal client during a slow HTTP POST attack.*

connection resources are exhausted, and other requests are starving. Thus, as shown in Fig. 7, the request delays of a normal client rise to 90 s for CloudFront (returns HTTP 504 Gateway Time-out) and 15 s for Fastly (returns HTTP 503 Service Unavailable), demonstrating the success of the DoS attacks.

Because MaxCDN will abort the POST connection after 15 s, we periodically start 100 new concurrent connections every second during the attack period. As shown in Fig. 6, the connection number fluctuates at around 1500, as MaxCDN aborts the previous 15-s-lasting connections sequentially. Meanwhile, as shown in Fig. 7, the request delay of a normal client fluctuates below 15 s, as the normal client request competes with attacking requests for the released connection resources. This phenomenon of MaxCDN demonstrates a quality of service (QoS) attack, which aims to degrade performance rather than completely disable the service.

HTTP/2 Pre-POST Attack. Given that CDNs support HTTP/2 in *client–CDN* connections (as explained in Section III), we also further evaluate slow HTTP/2 POST attacks against the *origin*. To employ the multiplex stream feature of HTTP/2, we establish 10 simultaneous HTTP/2 connections with the CDN and send 100 POST requests in each HTTP/2 connection. The POST requests are crafted as follows:

```
:method: POST
:scheme: https
:authority: www.victim.com
:path: /login.php?cdn=<cdn>&a=<time>-<range(0,1000)>
Content-Length: 300
0101..... (300 bytes, 1 byte sent per second)
```

TABLE VII: Time data from sending slow HTTP/2 `POST` requests (lasting 300 s). Three CDNs start `POST` request forwarding as soon as they receive the `POST` header.

	CloudFront	Cloudflare	CDNSun	Fastly	KeyCDN	MaxCDN
Request Receiving Time	**0.42342s**	300.82689s	300.47039s	**1.42386s(10)** 300.50451s(990)	300.49957s	**0.91270s**
Connection Keep-alive Time	**300.48742s**	0.21612s	3.22843s	**299.41059s(10)** 0.84946s(990)	3.08003s	**15.01520s**

As shown in Table VII, we obtain the same `POST` forwarding behaviors as in HTTP/1.1, except for Fastly. The result reveals that Fastly starts the pre-`POST` forwarding of the first request for each connection, with 10 *CDN–origin* connections having an average kept-open time of 299.41059 s. Meanwhile, the subsequent `POST` requests within the same connection are queued in Fastly for 300 s, during which Fastly has to finish receiving the subsequent whole `POST` message, resulting in 990 *CDN–origin* connections having an average kept-open time of 0.84946s. We presume the reason for this phenomenon is that Fastly maintains a `POST` request queue for each HTTP/2 connection, and thus subsequent `POST` requests are to be forwarded only after the foremost `POST` request has been finished.

To the target *origin*, this connection-exhaustion attack works in the same way as direct slow HTTP attacks but consumes fewer resources from the attacker, *e.g.*, the attacker needs to maintain just one connection with the CDN, which is then abused to proxy and maximize simultaneous *CDN–origin* connections.

Summary. With real-world experiments, three out of the six CDN vendors are shown to support pre-`POST` forwarding. This pre-`POST`-forwarding behavior introduces a new attacking vector to break the CDN protection and enable resource exhaustion attacks against the *origins* of the CDN-powered websites.

V. Degradation-of-Global-Availability Attack

Because the request-routing mechanisms of a CDN can be bypassed, and CDN surrogate servers can be accessed directly (as explained in Section II-A), an attacker can directly send crafted attacking requests to the globally distributed ingress IPs to render the threats described in Section IV and Section III into DDoS attacks, as shown in Fig. 8.

After collecting a massive number of CDN ingress IPs (surrogate IPs), we evaluate the feasibility of a CDN-rendered DDoS attack. We find that, compared with the number of ingress IPs that we used, the number of egress IPs that a CDN uses to forward requests to the *origin* with is smaller, resulting in a much lower egress IP-churning rate. We therefore present the possibility that this lower IP-churning rate can be leveraged to effectively degrade the global availability of the CDN-powered websites.

In this section, we first reveal how to find the ingress and egress IP distributions of a CDN, illustrate the low IP-churning rate of the egress IPs, and explain the degradation-of-global-availability attack.

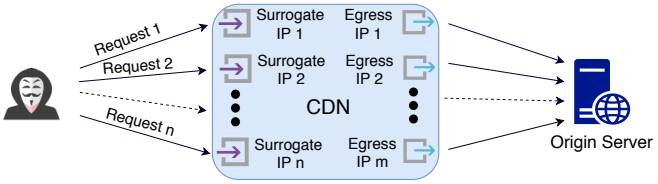

Fig. 8: *Through sending of requests to ingress IPs directly to simulate global access, a CDN is abused to proxy a DoS attack into a DDoS attack.*

A. CDN Ingress and Egress IP Distribution

Instinctively, to determine the ingress and egress IP addresses of a given CDN, we can directly find the IP addresses of the CDN either from the ICANN WHOIS database or from officially published information provided by some CDN vendors [14], [15], [21]. However, WHOIS information may be incomplete or obsolete (various European registrars have stopped collecting information for the ICANN WHOIS database because of the GDPR's principle of data minimization [17]), and the officially published addresses are just IP address ranges that do not separate the ingress IP addresses from the egress IP addresses.

Because we need an in-depth analysis of how a CDN assigns the ingress and egress IP addresses when receiving an end-user request, we explore the ingress IP distribution via an Internet-wide scan and unveil the egress IP distribution by sending requests to all ingress IP addresses directly to simulate global end-user accessing.

Ingress IP Distribution. With our website deployed behind a CDN, an Internet-wide HTTP scan is a direct method to collect the ingress IPs, through which we can access the contents of our website. To avoid offensive Internet scanning in this study, we first use *Censys* [11] Internet HTTP scanning data to filter possible ingress IPs. The *Censys* project scans TCP port 80 with the `Host` header filled with the scanned IP address, and the CDN surrogate servers will return distinctive error HTTP responses (header or body) when they are being accessed with this incorrect IP-form HTTP `Host` header, as shown in Table VIII.

TABLE VIII: Characteristics of HTTP response. CDN surrogate servers will return distinctive error HTTP responses when receiving incorrect `Host` headers.

CDN	Status Code	Response
CloudFront	403	Header: "Server: CloudFront"
CloudFlare	403	Header: "Server: cloudflare"
CDNSun	400	Header: "X-Edge-IP/X-Edge-Location"
Fastly	500	Body: "Fastly error"
KeyCDN	403	Header: "Server: keycdn-engine"
MaxCDN	200	Header: "Server: NetDNA-cache/2.2"

We then actively send requests to these filtered IPs with our website domain name in the `Host` header; the IPs through which we can access our *origin* are collected as ingress IPs.

TABLE IX: CDN IP distribution. CDNs employ much fewer egress IPs compared with ingress IPs (N1: number of IPs, N2: number of BGP Prefixes, N3: number of BGP ASes).

	Requesting Routing	Ingress IP			Egress IP			Percentage of requests when EgressIP=IngressIP
		N1	N2	N3	N1	N2	N3	
CloudFront	DNS	128906	720	29	862	160	3	0.06%
CloudFlare	Anycast	490309	93	28	242	72	1	0%
CDNSun	DNS	-	-	-	-	-	-	-
Fastly	DNS	64659	170	34	1136	56	1	0.02%
KeyCDN	DNS	-	-	-	-	-	-	-
MaxCDN	Anycast	300	16	2	12	12	2	3.82%

As shown in Table IX, we find a large number of ingress IPs [3].

Egress IP Distribution. For our experiments, we first set up an *origin* server, which will record incoming egress IPs and requested URLs. From a client, we directly send requests to all ingress IPs that we found. These requests are tailed with different query strings to avoid the cache-hit of the CDN, *e.g.*, `http://IngressIP1/i.php?IngressIP1`. The CDN then forwards these requests to our *origin* server through different egress IPs. Finally, from the data recorded at the *origin* server, we collect the egress IPs and extract the corresponding ingress IPs from the URL query strings.

To collect as many egress IPs as possible, we send requests hourly for 24 hours to the ingress IPs and record the resulting data at the *origin*. The number of the egress IPs, after duplicates are removed, are shown in Table IX, together with their BGP prefixes and ASes (determined by data from the Route View Project [50]). From Table IX, we can see that even if a CDN has a massive number of ingress IPs, the CDN will group incoming requests and assign a small set of egress IPs to forward the requests to the *origin*. We also find that, for each of the forwarded requests, the egress IP of the request is different from the ingress IP, *e.g.*, in the 24-hour scale, only 0.06% of the requests we send through CloudFront have the same ingress/egress IP, whereas the corresponding percentages for the other CDNs are 0% for Cloudflare, 0.02% for Fastly, and 3.82% for MaxCDN.

Further, within the same 24-hour measurement duration, we analyze the egress IP-churning rate (or occurrence ratio), which describes how frequently a CDN repeatedly assigns the same egress IP. In Fig. 9 (with the Y-axis in logarithmic scale), for simplicity, we just plot the ratios for the top 32 most assigned egress IPs [4]. From Fig. 9, we can see that *96.32% of the MaxCDN requests come from one single Egress IP*. In other words, MaxCDN has assigned most of the requests through just one single egress IP. For the other CDNs, on the other hand, we can see in Fig. 9 that their egress IPs are assigned more evenly or randomly, where no egress IP is charged with more than 10% of the requests.

Impact of *Origin* Location. We can see that some CDNs assign a small set of egress IPs to access the *origin* and do not

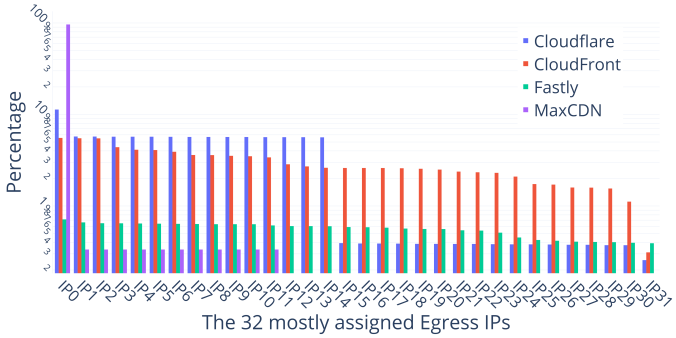

Fig. 9: *Occurrence ratios of the 32 egress IPs with the highest occurrences (in descending order).*

churn these IPs quickly. To verify if the location of the *origin* will affect the attack, or in other words, whether the MaxCDN-assigned egress IPs are a function of the *origin* IP, we set up and conduct experiments with *origin* servers located in Silicon Valley, Singapore, and Beijing, and determine that the most assigned egress IP is the same for these different locations. Further, our results are consistent with [34], which is published after our work. L. Jin *et al.* studied the address spaces of the ingress IPs of three CDN vendors (*i.e.*, Cloudflare, CloudFront, and Fastly) by resolving the IPs through public Open DNS resolvers, and further explored egress IP addresses using the same method as ours. They found fewer ingress IPs than we did and confirmed that the address space of the egress IPs is quite limited and churning at an extremely low rate. They also reveal that this lower egress IP churning rate is due to the internal two-layer structure of the CDN, composed of a client-facing layer for receiving client requests, and an *origin*-facing layer for fetching requested contents from the *origin*. This two-layer structure improves the CDN cache-hit ratio and lowers the workload of the *origin*. However, we can see that this lower IP-churning rate also makes attacks on *CDN–origin* connections much easier.

B. Attack Surface Analysis

A CDN provides a website with global availability via its massively geo-distributed surrogate servers. If an attacker wants to stop or degrade the global availability of a CDN-powered website, the most obvious method is to launch a DoS attack against the *origin* directly. However, the IP address of the *origin* is difficult to determine without relevant historical data [58] or accidental leakage of information [35]. Thus, to attempt the second-best method of attack, the attacker can try to invade and control some on-path network infrastructures (*e.g.*, routers or firewalls), to be on-path in *client–CDN* connections or *CDN–origin* connections to block the relevant IPs. However, because the CDN surrogates are globally distributed with massive numbers of IP addresses, it is impossible for an attacker (even for a state-sponsored attacker) to be on-path in all *client–CDN* connections or *CDN–origin* connections. Therefore, on the premise that the attacker can just block just a few connections, normally the attack can not affect the global availability of the website.

Here, as shown in Fig. 10, we visualize a threat model that an attacker can cut off only one or a small set of *CDN–origin* connections and who aims to degrade the global availability of a CDN-powered website. We argue that this

[3] We cannot filter any CDNSun IPs from the *Censys* data, and we can filter only 155 KeyCDN IPs but cannot access our website through these IPs. Querying the open DNS resolvers is another operational method to find ingress IPs [31], [34], but the result is totally determined by the data-set of open DNS resolvers (*i.e.*, how many and how globally geo-distributed these open resolvers are). We think the result of Internet scanning provides a better coverage, because we find more ingress IPs using the Internet scanning method than using the DNS querying method used in [34].

[4] These 32 IPs are obviously different across CDNs, we just use IP0 to IP31 to symbolize the mostly assigned IPs for each CDN.

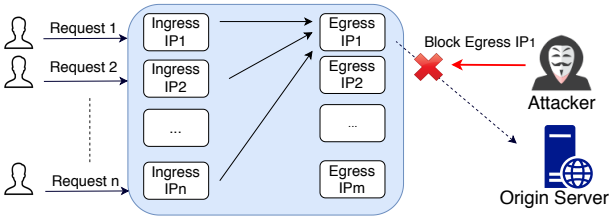

Fig. 10: *Degradation-of-availability attack. When a CDN assigns global requests through one or a small set of egress IPs, cutting off just one or a small set of CDN–origin network paths can effectively block most global requests to a website.*

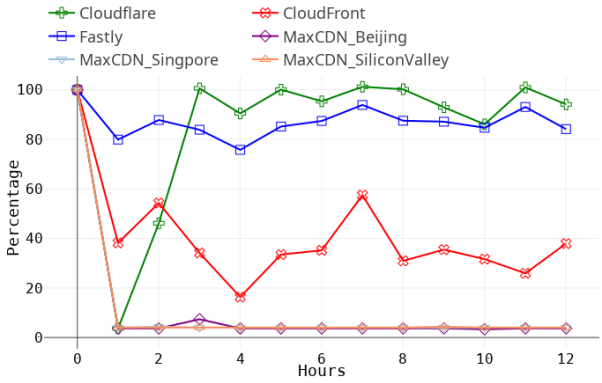

Fig. 11: *Accessing ratios when the most assigned egress IPs are blocked.*

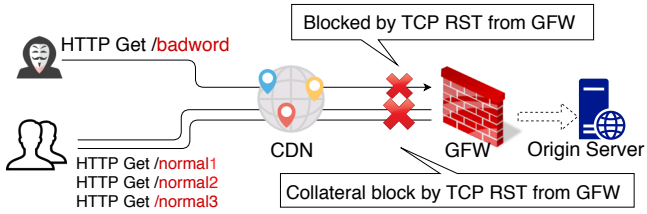

Fig. 12: *GFW collateral blocking attack.*

threat model is practical, because it requires an attacker to invade and control much fewer on-path network infrastructures (*e.g.*, router or firewall) to cut off one or a small set of *CDN–origin* connections, or an attacker can launch the **crossfire attack** [37] to stealthily cut off the Internet connections of one or a small set of CDN nodes by flooding network links around the CDN nodes. Based on this threat model, we further evaluate the feasibility and severity of the degradation-of-global-availability attack.

C. Real-World Attack Analysis

Experiment Setup. We can see that, according to Fig. 9, degrading the availability of our MaxCDN-powered website requires cutting off just one *CDN–origin* connection (*i.e.*, blocking one egress IP), whereas for the other CDNs, the attack requires cutting off more connections.

In our experiments, we send requests at hour 0 to all of the ingress IPs of a given CDN to simulate global clients accessing our website and obtain the number of successful requests as the base for calculating the following accessing ratios.

Afterward, starting from hour 1, the most assigned egress IPs are blocked to simulate the cutting off of *CDN–origin* connections (*e.g.*, a *crossfire attack*). For MaxCDN, we block just one egress IP in an on-path firewall, whereas for the other CDNs, we block the top 16 most assigned egress IPs. We still hourly send requests to the same sets of ingress IPs repeatedly for 12 hours, to simulate global clients accessing, and record the successful-access ratios. Note that all requests are sent with random URLs to bypass CDN caching.

Experiment Results. We plot the hourly accessing ratio in Fig. 11. Because MaxCDN still assigns the blocked IP to forward most requests, the accessing ratio drops to less than 10% within 12 hours. This phenomenon further reveals that MaxCDN lacks the mechanism to detect the attack when other egress IPs can still access the *origin*.

In Fig. 11, we can see that the other CDNs also lack the mechanism to detect the attack, *e.g.*, for CloudFront, the accessing ratio fluctuates at around 40% after blocking, whereas for Cloudflare and Fastly, the accessing ratios fluctuate at around 90%, which may be attributed to their capabilities of churning egress IPs more quickly.

Practical Analysis. Because the IP address of the *origin* is hidden behind a CDN, a direct DoS attack on the *origin* is impossible. We assume that an attacker (*e.g.*, state-sponsored) can cut off one or a small set of *CDN–origin* connections, either by invading an on-path network infrastructure (*e.g.*, router or firewall) or launching a crossfire attack to block the Internet access of the egress nodes. Here, we further reveal how a normal attacker can easily gain the power to cut off *CDN–origin* connections in a certain network scenario that makes the attack more practical.

It is well known that the on-path Great Fire Wall (GFW) will inspect the HTTP connections that may pass through it. Upon the detection of any sensitive banned words (*e.g.*, ultrasurf) within an HTTP connection, the GFW will inject TCP RST packets to shut down the connection, and the pair of IPs in this HTTP connection will be blacklisted for nearly 90 s by the GFW [59]. In this paper, we present how a normal attacker can abuse the power of the GFW to launch an availability degradation attack.

CDNs still support the plain text HTTP protocol in *CDN–origin* connections, which can be intercepted by on-path network infrastructures [41]. When the GFW is already located on-path between the CDN and the *origin*, as in Fig. 12, an attacker can deliberately send HTTP `GET` requests with GFW-banned bad words to activate the connection resetting mechanism of the GFW.

This mechanism leads to the pair of the CDN egress IP and *origin* IP being blocked for 90 s by the GFW. Within these 90 s, when normal clients try to access the website, and if the same CDN egress IP is assigned to access the *origin*, then the GFW will continue to reset the new TCP connections that follow, even when there are no GFW-banned bad words in these connections. Consequently, normal clients will be blocked from accessing the website, leading to a degradation of the availability of the *origin*.

The severity of such a degradation depends on how many CDN egress IPs can be added into the blacklist of the GFW. As we have illustrated, an attacker can easily harvest CDN ingress IPs and continually send GFW-banned requests to each of these ingress IPs. Depending on the egress IP assignment policy of the CDN, increasingly more CDN egress IPs will be added into the GFW blacklist. Note that the CDN assigning a small set of egress IPs further lowers the bar of such an attack, because adding these fewer egress IPs into the GFW blacklist consumes much less time. Consequently, when all of the egress IPs have been blacklisted, no clients will be able to access the target website, resulting in the service unavailability of the *origin*.

For simplicity and ethical concerns, we further evaluate the attack against our website located in China, which is hosted behind MaxCDN. Because the GFW resets TCP connections sent into or out of China, and the egress IPs are all located outside China, we set up a website in Beijing that is deployed behind MaxCDN. Later, from the vantage point of Singapore as the attacker, we continually send `GET /ultrasurf` HTTP requests directly to the MaxCDN ingress IPs we find earlier. Meanwhile, from another vantage point of Singapore acting like normal clients, we send `GET /normal` HTTP requests to verify whether the website is still accessible. As illustrated in Table IX, to access our website *origin*, MaxCDN assigns less than 12 egress IPs, which can be collaterally blocked by the GFW and, in that event, render our website totally inaccessible.

We admit that, as the GFW is abused, the attack can affect only the *origins* located in China, when they are being accessed through CDNs outside of China. However, we can see that the lower egress IP-churning rate of the CDN does lower the difficulty of the attack. As state-level Internet censorship systems and middle boxes become widespread on the Internet [4], [16], [32], [45], [61], the threat becomes applicable to more situations and will become more severe as time goes on.

Summary. First, we can see that, compared with the tens of thousands of ingress IPs that a CDN normally works with, the egress IP address space is much smaller, which helps an attacker to narrow down the attack targets. Second, the lower IP-churning rate lowers the difficulty of attacks on the *CDN–origin* connections, *e.g.*, access blocking (on-path blocking or off-path DoS attack, such as the "CrossFire" attack), traffic eavesdropping, or finding *origin* server IP via historical network traffic. Therefore, we believe that this threat may be more severe than one might suppose at first glance.

VI. Discussion

A. Severity Analysis

In this paper, we reveal, using real-world measurements across six CDNs, that the operational and architectural weaknesses of these CDNs can be exploited to break the DoS protection provided by these CDNs.

For a CDN-powered website, the CDN recommends that the *origin* enforce a firewall white-list to allow CDN-initiated connections only. Because the *origin* has to communicate only with the more trustworthy CDN, the *origin* may totally entrust the CDN for DoS protection without enforcing any local anti-DoS mechanisms. Thus, when the DoS protection provided by the CDN is bypassed, it can lead to more severe damage against the *origin*.

Because the CDN vendors do not validate the ownership of the *origins*, a malicious CDN customer can configure any other website as an *origin* behind the CDN [26]. Therefore, the threats in this paper can also be abused to attack not only the websites already hosted in the CDN, but also unwitting websites not hosted in the CDN. Fastly security team has expressed the same concern in their response to our responsible disclosure.

B. Causes and Mitigation

Generally, the threats exist, in part, because of market competition; the CDN vendors naturally want to provide more functionality and achieve maximum compatibility with customer websites of different configurations. However, the World Wide Web ecosystem is threatened by both network and application layer threats, and thus the full-featured functionality offered by these CDNs, with protocol flaws or implementation weaknesses, could be exploited to break CDN security.

HTTP/2 Bandwidth Amplification Attack. The threat arises from the half-done HTTP/2 support of CDNs. The fact that HTTP/2 is turned on by default for many CDNs makes this threat more severe. We assume the reason behind this vulnerability is that CDN vendors lack the motivation to enable HTTP/2 in *CDN–origin* connections. For example, Cloudflare states that, "The HTTP/2 protocol is focused on improving the browser behavior now, it's not necessary to make any modification to the origin for enabling HTTP/2." [25]. Another reason may be that HTTP/2 is still not widely deployed by websites on the Internet. According to CloudFront, "The connection from CloudFront back to your origin server is still made using HTTP/1. You don't need to make any server-side changes in order to make your static or dynamic content accessible via HTTP/2." [5].

Fundamentally, HTTP/2 specifications lack sufficient security consideration on HTTP/1.1-and-HTTP/2 coexistence environments [8], [49]. Meanwhile, when CDN vendors rush to support HTTP/2 to obtain efficiency-centered features, these CDN vendors support HTTP/2 only in the *client–CDN* connections, resulting in an HTTP/2–HTTP/1.1 conversion environment not clearly defined in the related specifications. These two factors contribute to the HTTP/2 threat. Thus, we think CDN vendors should be conservative in supporting this new protocol and make it as an "opt-in" option instead. Moreover, if HTTP/2 is turned on, CDN vendors should also further restrict the converted *CDN–origin* HTTP/1.1 connections.

Pre-`POST` Slow HTTP Attack. In general, a CDN decouples traditional client–website connections into *client–CDN* and *CDN–origin* connections, a set-up that naturally defends against slow HTTP attacks from the client side. However, the pre-`POST`-forwarding behaviors of some CDNs empower attackers to control back-end *CDN–origin* connections.

The pre-`POST` threat takes advantage of the intention of some CDN vendors to speed up `POST` forwarding, while introducing a new attacking vector. Our study shows that three out of the six CDNs that have been examined are vulnerable to the threat. The most obvious mitigation is for website administrators to implement a timeout on the *origin* side, although

this workaround requires configuration on every CDN-powered website. We suggest that CDN vendors implement a stricter `POST`-forwarding mechanism, such as the *store-then-forward*, which has already been applied by Cloudflare.

Degradation-of-Global-Availability Attack. The egress IP assignment strategy of a CDN is definitely implementation-dependent. Based on our measurements, the egress IP assignment strategies of some CDN vendors are predictable. MaxCDN, especially, assigns most global requests through the same egress IP even when the *origin* is located in a different region. Thus, degradation-of-global-availability attack is made more practical for an attacker, requiring the cutting off or blocking of fewer *CDN–origin* connections.

The threat exploits the emphasis of CDNs to access web resources efficiently with fewer IP resources, *i.e.*, to access and cache more efficiently [40], [52]), making degradation-of-global-availability attacks easier to perform. Therefore, to provide more robust network services, we suggest that CDN vendors adjust their egress IP assignment strategies to be more unpredictable, such as by assigning more egress IPs and churning them frequently.

Summary. The existence of these three threats unveil the pursuit by CDNs toward usability and efficiency, while apparently neglecting security. Overall, we suggest the following CDN-side mitigation, listed in Table X.

TABLE X: Recommended mitigation.

Threat	Recommendation
HTTP/2 Attack	opt out of the CDN HTTP/2 support, limit the CDN back-to-origin network traffic.
Pre-POST Attack	limit the number of CDN back-to-origin connections, enforce strict store-then-forward mechanism.
Global Availability Attack	apply unpredictable IP churning strategy.

Furthermore, as we show that CDN-forwarded requests can be abused to attack website servers, we also recommend that website servers enforce local DoS defenses, *e.g.*, requests filtering or bandwidth limiting, even if these website servers are deployed behind trustworthy CDNs.

C. Ethics and Responsible Disclosure

Throughout this study, we aim to achieve a balance between real-world severity evaluations and risks of impacting the CDN vendors, such as consuming too much bandwidth during our experiments, which may cause bilateral damage to the other CDN-powered websites and will introduce an ethical problem to our academic study. Thus, we take utmost care to prevent ethics problems in our experiments. First, our experiments are conducted with free trial CDN accounts and default configurations. Second, in exploring the various behaviors of the CDNs, we carefully use limited network resources to generate legal HTTP requests. Third, the *origin* website is implemented by ourselves. Through illustrated approaches in our experiments, we believe we have minimized the security risks of our experiments on the CDNs and other co-hosting websites, and results have shown that our experiments did not trigger any CDN anti-DoS mechanisms.

Meanwhile, we have already reported all our findings to these CDN vendors months ago. The responses are summarized in the following.

Fastly: Fastly confirms the HTTP/2 threat; they have analyzed the report and are working with our various internal teams to understand how they might address this issue. They confirm that slow `POST` issues are problems on their infrastructure and suggest the *origin* administrators implement a timeout first, which may also be followed by the addition of a CDN configuration option to implement a timeout on processing the entire request body. Furthermore, they express concerns regarding two attack scenarios: 1. Existing customers using Fastly; 2. Unwitting victims (*origins* that have been configured on a service by a malicious CDN customer). Fastly also offered us T-shirts for thanks.

Cloudflare: Cloudflare reproduced the HTTP/2 issue with a $126\times$ bandwidth amplification ratio, which is smaller than our resulting $132.6\times$. We believe this difference is due to the header difference with Huffman encoding and `:path` field. Their newest response demonstrates that their team has been trying to fix the threat by putting an upper bound limit on the size of the HTTP/2 dynamic table. This vendor also rewarded us with $200 for our efforts.

CloudFront: CloudFront have said that they thought the HTTP/2 issue is a product of the HTTP/2 standard, and when an *origin* believes that they are the target of abusive behavior, they can engage via the AWS Abuse process. Given that CloudFront will pass along all traffic (including `POST` requests), the *origins* could also make use of rate-based AWS WAF rules to specify the number of web requests that are allowed by a client IP to mitigate the attack. However, the WAF rules require being specifically configured by their customer websites according to their respective website needs, which is not a general solution.

MaxCDN: Months after we submitted our report, MaxCDN has responded that `POST` requests are not forwarded to the *origin* until the full payload data is received. We re-do the experiments and observe that the slow `POST` issue has been mitigated. Meanwhile, we find that the MaxCDN web user-interface has changed, and thus we believe that the threat is collaterally mitigated because of other upgrades. Later, they respond that the HTTP/2 threat is already known, although they did not respond further when we submitted the actual GFW-based proof of concept.

CDNSun and KeyCDN: These vendors thanked us for the messages and forwarded the issues to their CDN developers. However, we have received no further response.

VII. Related Work

CDN Security. By rerouting traffic to its globally distributed network infrastructures with higher bandwidths, a CDN offers a dedicated DDoS protection service to the websites that it supports [24]. Methods and mechanisms of breaking or bypassing CDNs are therefore a hot topic in the network security research area. A previous study reported on CDN forwarding-loop attacks, causing the request to be processed repeatedly and resulting in a DoS attack between CDNs [13]. Because the CDN-decoupled frontend connection and backend

connection can have asymmetrical bandwidths, an attacker can abort the frontend connection to exhaust the bandwidth of the backend connection [57].

Attackers have also been confirmed to be able to maneuver the mappings of CDNs between clients and surrogates via crafted DNS records [28]. With the high IP reputation and co-hosting of popular websites given consideration, the infrastructure of the CDN has also been leveraged to circumvent Internet censorship [30], [63]. Furthermore, a malicious CDN customer can configure a target website server as an unwitting *origin* behind a CDN, abusing the CDN resources to attack target servers [26]. Therefore, as Fastly confirmed, the vulnerabilities reported in this paper not only affect existing customer websites using the CDN, but can also affect other non-CDN-powered websites.

From the perspective of the CDN *origin*, prior works are focused mainly on sensitive information disclosure and misconfiguration. Attackers are highly interested in determining the IP addresses of CDN-powered website *origins* to directly bypass CDN protection [58]. Furthermore, the DNS resolution flaw of CDNs could also possibly leak the IP addresses of the *origins* [35].

Because of the conflict between the man-in-the-middle nature of CDN and the end-to-end encryption nature of HTTPs, prior researchers have explored the TLS key management problems, such as private key sharing and inefficient revocation, in CDN platforms [10], [41]. Further, by exploiting inconsistent interpretations of HTTP header fields between the CDN and *origin*, CDN caching mechanism can be abused to launch cache poisoning attacks [12], DoS attacks [46], and cache deception attacks [42].

Unlike previous researches, our work explores threats in CDN forwarding behaviors that have not yet been well studied, providing a complement to existing CDN security research.

HTTP-Related DoS Attacks. The HTTP DoS attack can be launched simply, when legitimate HTTP requests are initiated in large numbers [36]. Furthermore, configurations and functions related to the HTTP service can introduce new DoS attacking vectors [39]. The HTTP DoS attack is an application-layer attack, posing challenges for detection because the attacking requests appear similar to normal end-user requests [27], [54], [62]. In this paper, we present issues in the request-forwarding process of the CDN. Whereas the CDN is normally considered as an effective anti-DDoS solution for websites, our study shows that the CDN itself can be abused to launch the attack, breaking the CDN DoS protection.

In the HTTP/1.1 era, slow HTTP attacks are already well known [29], [48], [56], but the advent of HTTP/2 introduces new attacking vectors. Beckett *et al.* has reported that HTTP/2 helps to scale up the magnitude of HTTP flood DDoS attacks [6], [7]. We further extend their study on CDNs and analyze the impact of the HPACK mechanism, *e.g.*, Huffman encoding and `:path` header field, on the amplification ratio.

Meanwhile, botnets have historically been used to launch DDoS attacks [1], [43], [51]. Since the emergence of IoT devices, the costs required for these attacks have been decreasing, whereas traffic generated by these attacks has been increasing [2], [38]. Although attacks from botnets can be mitigated by IP blocking, the HTTP DoS attacks described in this paper are initiated from the CDN itself, making the attacks stealthier and more difficult to detect. Even worse, a CDN-powered website cannot apply IP blocking to mitigate the attacks, because blocking CDN IPs will make the website totally inaccessible for all clients.

Issues on IP Assignment Strategy. The IP assignment, which concerns how a network service assigns IP addresses, of a DHCP server is vulnerable to DHCP starvation attacks [44]. Borgolte *et al.* has revealed the IP use-after-free vulnerability in the cloud, which can be exploited by attackers to deceive domain-based certificate issuance [9]. In CDN, the egress IP assignment, which is related to how a CDN assigns an egress IP to forward requests to an *origin*, was also studied by Jin *et al.* [34], and they observed the same results that we did.

VIII. Conclusions

The CDN has become an indispensable part of the Internet, providing, among other benefits, anti-DoS services for its CDN-powered websites. However, through the exploitation of its architectural, implementation, or operational weaknesses, the CDN itself can also be leveraged to break the CDN DoS protection.

By revealing three relevant threats and presenting real-world measurements across six CDNs, this paper reveals the flawed trade-offs made by CDN vendors between security and usability. We report that, because of protocol or implementation weaknesses, full-featured HTTP forwarding support in CDNs can be abused to launch an efficient DoS attack against website *origins*. We envision our work being able to urge CDNs to raise their security standards, and inspire more researchers to explore CDN-related security.

Acknowledgment

We thank the anonymous reviewers for their insightful comments that helped improve the quality of the paper. We are grateful to our shepherd Ben Stock for his guidance on improving our work. This work is supported by the NSFC (Grant No. U1836213, U1636204), BNRist Network and Software Security Research Program (Grant No. BNR2019TD01004). This work is supported in part by the Office of Naval Research under ONR award number N00014-20-1-2738. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] E. Alomari, S. Manickam, B. B. Gupta, S. Karuppayah, and R. Alfaris, "Botnet-based Distributed Denial of Service (DDoS) Attacks on Web Servers: Classification and Art," *International Journal of Computer Applications*, 2012.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the mirai botnet," in *USENIX Security '17*, 2017.

[3] Apache, "Apache doc," https://httpd.apache.org/docs/, [Accessed Feb. 2019].

[4] S. Aryan, H. Aryan, and J. A. Halderman, "Internet censorship in iran: A first look," *USENIX FOCI '13*, 2013.

[5] J. Barr, "Http/2 support for amazon cloudfront," https://aws.amazon.com/blogs/aws/new-http2-support-for-cloudfront/, [Accessed Aug. 2018].

[6] D. Beckett and S. Sezer, "Http/2 cannon: Experimental analysis on http/1 and http/2 request flood ddos attacks," *Seventh International Conference on Emerging Security Technologies (EST)*, 2017.

[7] D. Beckett and S. Sezer, "Http/2 tsunami: Investigating http/2 proxy amplification ddos attacks," *17th International Conference on Emerging Security Technologies (EST)*, 2017.

[8] M. Belshe, R. Peon, and E. M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540.

[9] K. Borgolte, T. Fiebig, S. Hao, C. Kruegel, and G. Vigna, "Cloud strife: Mitigating the security risks of domain-validated certificates," *NDSS '18*, 2018.

[10] F. Cangialosi, T. Chung, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "Measurement and analysis of private key sharing in the https ecosystem," *CCS '16*, 2016.

[11] censys, "censys.io," https://censys.io/, [Accessed Aug. 2018].

[12] J. Chen, J. Jiang, H. Duan, N. Weaver, T. Wan, and V. Paxson, "Host of troubles: Multiple host ambiguities in http implementations," *CCS '16*, 2016.

[13] J. Chen, J. Jiang, X. Zheng, and H. Duan, "Forwarding-Loop Attacks in Content Delivery Networks," *NDSS '16*, 2016.

[14] Cloudflare, "Cloudflare ip address ranges," https://www.cloudflare.com/ips/, [Accessed Nov. 2019].

[15] CloudFront, "Cloudfront ip address ranges," https://ip-ranges.amazonaws.com/ip-ranges.json, [Accessed Nov. 2019].

[16] J. Cowie, "Egypt leaves the internet," https://dyn.com/blog/egypt-leaves-the-internet/, Jan. 2011.

[17] A. Dabrowski, G. Merzdovnik, J. Ullrich, G. Sendera, and E. R. Weippl, "Measuring cookies and web privacy in a post-gdpr world," in *PAM*, 2019.

[18] datanyze.com, "Cdn market share," https://www.marketsandmarkets.com/Market-Reports/content-delivery-networks-cdn-market-657.html, [Accessed Nov. 2019].

[19] datanyze.com, "Cdn market share," https://www.datanyze.com/market-share/cdn, [Accessed Nov. 2019].

[20] DynResearch, "Cdn adoption by the numbers," https://dyn.com/blog/dyn-research-cdn-adoption-by-the-numbers/, [Accessed Aug. 2018].

[21] Fastly, "Fastly ip address ranges," https://api.fastly.com/public-ip-list, [Accessed Nov. 2019].

[22] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic ddos defense," *24th USENIX Security Symposium*, 2015.

[23] R. Fielding, J. Reschke, and Ed., "Hypertext transfer protocol (http/1.1): Message syntax and routing," 2014.

[24] Y. Gilad, M. Goberman, A. Herzberg, and M. Sudkovitch, "Cdn-on-demand: An affordable ddos defense via untrusted clouds," *NDSS*, 2016.

[25] M. Gonlag, "Are the http/2 or spdy protocols supported between cloudflare and the origin server?" https://support.cloudflare.com/hc/en-us/articles/214534978, [Accessed Aug. 2018].

[26] R. Guo, J. Chen, B. Liu, J. Zhang, C. Zhang, H.-X. Duan, T. Wan, J. Jiang, S. Hao, and Y. Jia, "Abusing cdns for fun and profit: Security issues in cdns' origin validation," *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, 2018.

[27] G. D. Hakem Beitollahi, "Analyzing well-known countermeasures against distributed denial of service attacks," *Computer Communications*, 2012.

[28] S. Hao, C. Uc, S. Diego, and H. Wang, "End Users Get Maneuvered : Empirical Analysis of Redirection Hijacking in Content Delivery Networks," *USENIX Security '18*, 2018.

[29] T. Hirakawa, K. Ogura, B. B. Bista, and T. Takata, "A defense method against distributed slow http dos attack," *19th International Conference on Network-Based Information Systems (NBiS)*, 2016.

[30] J. Holowczak and A. Houmansadr, "CacheBrowser : Bypassing Chinese Censorship without Proxies Using Cached Content," *CCS '15*, 2015.

[31] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and evaluating large-scale cdns," *IMC '08*, 2008.

[32] S. Huang, F. Cuadrado, and S. Uhlig, "Middleboxes in the internet: A http perspective," *Network Traffic Measurement and Analysis Conference (TMA)*, 2017.

[33] hubspot.net, "2019 cdn market report," https://cdn2.hubspot.net/hubfs/4238862/2019 20Intricately 20CDN 20Market 20Report.pdf, [Accessed Nov. 2019].

[34] L. Jin, S. Hao, H. Wang, and C. Cotton, "Unveil the hidden presence: Characterizing the backend interface of content delivery networks," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, Oct 2019.

[35] L. Jin, S. Hao, H. Wang, and C. Cotton, "Your Remnant Tells Secret : Residual Resolution in DDoS Protection Services," *DSN '18*, 2018.

[36] Jin Wang, Min Zhang, X. Yang, Keping Long, and Chimin Zhou, "Http-scan: Detecting http-flooding attack by modeling multi-features of web browsing behavior from noisy dataset," *19th Asia-Pacific Conference on Communications (APCC)*, Aug 2013.

[37] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," *IEEE Symposium S&P*, 2013.

[38] M. Karami and D. McCoy, "Understanding the emerging threat of ddos-as-a-service," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats*. USENIX, 2013.

[39] K. K. Karanpreet Singh, Paramvir Singha, "Application layer http-get flood ddos attacks: Research landscape and challenges," *Computers and Security*, 2017.

[40] KeyCDN, "Origin shield - extra cdn caching layer," https://www.keycdn.com/support/origin-shield, [Accessed Oct. 2018].

[41] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When https meets cdn: A case of authentication in delegated service," *IEEE S&P '14*, 2014.

[42] S. A. Mirheidari, S. Arshad, K. Onarlioglu, B. Crispo, E. Kirda, and W. Robertson, "Cached and confused: Web cache deception in the wild," *USENIX Security*, 2020.

[43] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms," *SIGCOMM '04.*, 2004.

[44] H. Mukhtar, K. Salah, and Y. Iraqi, "Mitigation of dhcp starvation attack," *Comput. Electr. Eng.*, 2012.

[45] Z. Nabi, "The anatomy of web censorship in pakistan," *USENIX Workshop on Free and Open Communications on the Internet*, 2013.

[46] H. V. Nguyen, L. L. Iacono, and H. Federrath, "Your cache has fallen: Cache-poisoned denial-of-service attack," *CCS '19*, 2019.

[47] A. G. Oleg Kupreev, Ekaterina Badovskaya, "Ddos attacks in q4 2018," https://securelist.com/ddos-attacks-in-q4-2018/89565/, [Accessed Feb. 2019].

[48] J. Park, K. Iwai, H. Tanaka, and T. Kurokawa, "Analysis of slow read dos attack," *International Symposium on Information Theory and its Applications*, 2014.

[49] R. Peon and H. Ruellan, "Hpack: Header compression for http/2," 2015.

[50] RouteViews, "University of oregon route views project," http://www.routeviews.org/, [Accessed Aug. 2018].

[51] S. M. Specht and R. B. Lee, "Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures," *International Workshop on Security in Parallel and Distributed Systems*, no. 9, pp. 543–550, 2004.

[52] StackPath, "Origin shield: Protect your origin from traffic spikes," https://www.stackpath.com/products/cdn/origin-shield/, [Accessed Oct. 2018].

[53] Stackpath, "Maxcdn is now stackpath," https://www.stackpath.com/maxcdn/, [Accessed Oct. 2019].

[54] K. Subramanian, P. Gunasekaran, and M. Selvaraj, "Two layer defending mechanism against ddos attacks," *International Arab Journal of Information Technology*, 2015.

[55] S. Systems, O. Spatscheck, A. Barbir, and R. Nair, "Known Content Network (CN) Request-Routing Mechanisms," RFC 3568, Oct. 2015.

[56] N. Tripathi, N. Hubballi, and Y. Singh, "How secure are web servers? an empirical study of slow http dos attacks and detection," *11th International Conference on Availability, Reliability and Security (ARES)*, 2016.

[57] S. Triukose, Z. Al-qudah, and M. Rabinovich, "Content Delivery Networks : Protection or Threat," *ESORICS '09*, 2009.

[58] T. Vissers, T. V. Goethem, W. Joosen, and N. Nikiforakis, "Maneuvering Around Clouds : Bypassing Cloud-based Security Providers," *CCS '15*, 2015.

[59] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, "Your state is not mine: A closer look at evading stateful internet censorship," *IMC '17*, 2017.

[60] Wikipedia, "Http/2," https://en.wikipedia.org/wiki/HTTP/2, [Accessed Aug. 2018].

[61] Wikipedia, "Internet regulation in turkey," https://en.wikipedia.org/wiki/Internet_regulation_in_Turkey, Nov. 2019.

[62] X. Yuan, C. Li, and X. Li, "Deepdefense: Identifying ddos attack via deep learning," *IEEE International Conference on Smart Computing (SMARTCOMP)*, 2017.

[63] H. Zolfaghari and A. Houmansadr, "Practical Censorship Evasion Leveraging Content Delivery Networks," *CCS '16*, 2016.