Unveil the Hidden Presence: Characterizing the Backend Interface of Content Delivery Networks

Lin Jin
University of Delaware
linjin@udel.edu

Shuai Hao
Old Dominion University
shao@odu.edu

Haining Wang
Virginia Tech
hnw@vt.edu

Chase Cotton
University of Delaware
ccotton@udel.edu

***Abstract*—Content Delivery Networks (CDNs) are critical to today's Internet ecosystem for delivering rich content to end-users. CDNs augment the Internet infrastructure by deploying geographically distributed edge servers, which play a dual role in CDNs: one as *frontend* interface to facilitate end-user's proximal access and the other as *backend* interface to fetch content from origin servers. Previous research has well studied the frontend interface of CDNs, but no active approach has yet been provided to investigate the backend interface. In this paper, we first propose an active approach to measuring the backend interface of CDNs. Then, we present a large-scale measurement study to characterize the backend interface for three CDN platforms, so as to understand the CDN's globally distributed infrastructure, which is essential to its performance and security. In particular, we discover the address space and operation patterns of the backend interface of CDNs. Then, by analyzing the backend addresses and their associated frontend addresses, we study their geolocation association. Furthermore, we issue traceroutes from origin servers to the backend addresses of the CDNs to analyze their performance implications, and perform port scanning on the backend addresses to investigate their security implications.**

I. Introduction

Content Delivery Networks (CDNs) have been an integral component of the Internet ecosystem. They deliver rich content to end-users and significantly improve the performance, availability, and security of online services. CDNs have been widely deployed in the past decade and will continue their rapid growth in the future. According to the Cisco report [1], 52% of all Internet traffic was delivered by CDNs in 2016, and the percentage is expected to increase to 70% by 2021. The widely deployed CDNs and their high adoptions have been reshaping the landscape of the Internet.

Technically, a CDN is a geographically distributed network infrastructure with a large number of edge servers deployed at multiple strategically chosen locations, a.k.a., Points of Presence (PoPs). An edge server caches the requested content and shares it with neighboring servers for future requests. When a CDN service is enabled, an end-user's request will be redirected to one of the CDN's distance-proximity edge servers via the CDN load balancing system, and the cached content will be served directly from the edge server without fetching it from the origin server, which significantly reduces the end-to-end latency. Furthermore, when the origin server is temporarily disconnected from the Internet (e.g., under maintenance), the edge server can still serve the cached content so that it largely mitigates the service interruption.

978-1-7281-2700-2/19/$31.00 2019

A CDN can be seen as a middle-box between the content origin and end-users. Its frontend interface handles the requests from end-users, while its backend interface is responsible for fetching the requested content from the origin. Normally, the backend IP addresses are hidden from end-users. A prior study [2] examined the backend interface of one commercial CDN by analyzing its edge servers' logs. However, it is usually difficult to gain access to the edge servers' logs of multiple commercial CDN providers, and thus an active approach to studying the CDN backend interface is needed. Other existing CDN measurement studies [3]–[6] have explored the server deployment and performance of CDNs, including address space, geographical distribution, end-to-end latency, and availability, but these studies are limited to the frontend interface of CDNs only. However, to fully understand a CDN's globally distributed infrastructure, it is vital to characterize the backend interface of CDNs.

In this paper, we first design an active approach to discovering the backend interface of the CDN platform. The basic idea is to first collect the globally distributed frontend addresses and leverage them to uncover their associated backend addresses. In particular, we take advantage of the globally distributed DNS open resolvers to resolve the domain names of CDN customers. Since a CDN typically returns its frontend addresses based on the location of the open resolvers, we collect the globally distributed frontend addresses from the DNS responses. After that, we intentionally send HTTP requests to these frontend addresses and search for different and non-existent content of a website under our control. Such requests would force the CDN to fetch the content from our origin server so that we can collect the backend addresses as well as the associated frontend addresses.

We then present a comprehensive measurement study on the backend interface of three leading CDNs, including Cloudflare, CloudFront, and Fastly. In total, we collect 13,433 backend addresses in three months. In contrast to the large number of frontend addresses, the backend addresses are limited. In addition, the three CDN platforms show a significant difference in terms of the operation patterns of the backend addresses. Furthermore, we determine the geolocation of a backend address and its associated frontend addresses at the PoP level and present their geolocation association.

Next, we analyze the Round-Trip Times (RTTs) and Autonomous System (AS) paths between origin servers and CDN backend interfaces and explore their performance implications. We observe that the origins located at EC2's North America regions have the shortest average RTTs and that the majority of the AS path lengths are less than 4 after the removal of intermediate IXP ASes. Finally, we examine the port management of backend addresses and analyze its security implications. What we found is that some of the backend addresses of Fastly and CloudFront have their ports HTTP/80 and HTTPS/443 open to the public; we also noted that these backend addresses are allowed to directly process the client's HTTP requests and fetch the content for a client.

We summarize our major contributions as follows:

- We design an active approach to discovering the backend interface of CDN platforms, and such a discovery provides a more comprehensive understanding of CDN's infrastructure and deployment.
- We present a large-scale measurement to characterize the operation patterns of the backend interface of three large CDN platforms. Moreover, by analyzing backend addresses and their associated frontend addresses, we reveal their geolocation association.
- We issue traceroutes from origin servers to CDN backend interfaces and analyze their performance implications in terms of RTT and AS path length. Furthermore, we study the port management of backend addresses by probing the open ports. Based on the measurement results, we analyze the backend accessibility and its security implications.

The remainder of this paper is organized as follows. Section II introduces the background of CDNs. We describe our active approach in Section III and discuss our measurement in Section IV. The measurement results are analyzed in Section V. We explore the performance and security implications of the backend interfaces in Section VI. We survey the related work in Section VII, and finally, we conclude the paper in Section VIII.

II. Background

A. CDN Overview

A CDN is designed to reduce the end-to-end latency of fetching the content and lower the origin servers' workload burden. It works by caching and serving the content at its geographically distributed edge servers, thereby improving the performance and availability of online services that use CDNs.

Figure 1 illustrates how a CDN works. To enable a CDN service, a website needs to delegate its domain to a CDN by either changing its `CNAME` or `NS` record to a domain controlled by the CDN. After that, when a client accesses the website, it first resolves the domain name of the website to IP addresses. With the domain delegation, the IP addresses will be eventually provided by the CDN's authoritative nameservers (❶). We refer to these IP addresses as frontend addresses. Then, the client sends its HTTP requests to one of the provided

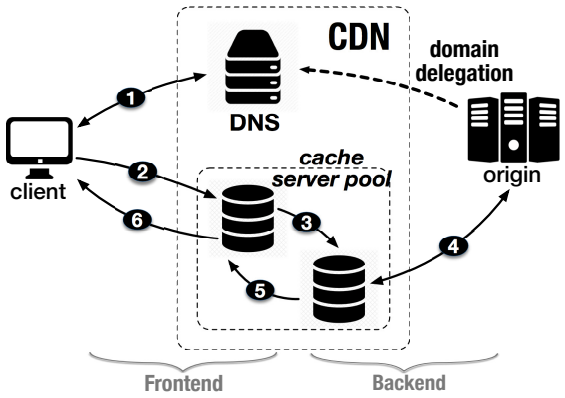

Fig. 1: CDN Illustration

frontend addresses to retrieve the web content (❷). An edge server would process this request and check with other edge servers to see if the requested content is cached (❸). Assuming that the content is not cached in the CDN, eventually one of the edge servers would fetch the requested content from the origin (❹).[1] We refer to the IP addresses that are used to fetch the content from the origin as the backend addresses. Once the edge server receives the content, it serves the content to the client and caches it (❺ and ❻). As a result, when other clients request the same content, the cached content will be served directly from the edge server without contacting the origin, which significantly reduces the end-to-end latency of serving the content and the workload at the origin.

Continuously evolving, CDNs have also begun to provide security-related services in addition to content delivery. The most representative service is the DDoS Protection Service (DPS) [7]–[9]. As DDoS attacks have become more powerful, it is insufficient to defend against them using only the traditional on-site protection systems. Having the feature of a highly distributed network infrastructure and high network capacity, a CDN becomes an ideal place to absorb a large amount of DDoS attack traffic. With the high demand for a dedicated DPS, the major CDNs have integrated the DPS within their platforms, and the DPS has become one of the core functions of CDNs. Furthermore, since a CDN acts as a reverse proxy for its customers, it can inspect the traffic sent to and from origin servers, even if the traffic is encrypted. With such a capability, CDNs carry out their Web Application Firewalls (WAFs) to block malicious web application traffic and stop web attacks (e.g., SQL injection and cross-site scripting) at the Internet edges.

B. CDN Load Balancing System

A CDN load balancing system redirects client requests to their distance-proximity edge servers, and it can be divided into two levels: PoP-level and server-level.

[1]Note that the edge server that receives the HTTP request and the edge server that fetches the content from the origin could be the same physical server, but with two logic interfaces (i.e., the frontend and backend interfaces).

1) PoP-Level Load Balancing: The PoP-level load balancing is used to balance the HTTP requests within a CDN's globally distributed infrastructure and redirect them to the distance-proximity PoPs. There are two types of mechanisms widely adopted in CDNs: DNS mapping [10] and anycast routing [10], [11].

DNS Mapping. When a CDN's authoritative DNS nameserver receives a DNS query searching for the `A` record of its customer, it selects the distance-proximity PoP and replies with the frontend addresses belonging to this PoP. Once the client receives the DNS response, it sends its subsequent HTTP requests to the replied frontend address, and hence the web traffic is redirected to the distance-proximity PoP.

Anycast Routing. To enable anycast routing, BGP routers in multiple PoPs announce the same set of IP prefixes to the Internet so that addresses within those prefixes would have multiple physical locations. By leveraging the BGP shortest path routing, a packet with an anycast address will be routed to its closest physical location. Consequently, the web traffic will automatically be redirected to a distance-proximity PoP.

2) Server-Level Load Balancing: Within a PoP, it is critical to evenly distribute the incoming requests to all edge servers. To achieve this goal, CDNs normally take advantage of the load balancers inside the PoP, since it is convenient to obtain the real-time network information (e.g., traffic load and ongoing connections) of the edge servers.[2]

For the TCP-based connections (e.g., HTTP connections), the server-level load balancer must ensure that all packets in the same connection are sent to the same edge server. Forwarding packets to a different edge server may reset the connection. Many CDNs achieve this by using flow hashing techniques such as Equal-Cost Multi-Path (ECMP) routing [13] and usually implement it as a software load balancer [14]. Typically, a flow hashing technique hashes the unique identifier (e.g., four tuples) of the flows and forwards the packets based on their hash value. The packets with the same hash value will be forwarded to the same edge server.

Due to the existence of a load balancer inside the PoP, packets with the same frontend address may not be redirected to the same edge server. Thus, there is no one-to-one mapping between a frontend address and an edge server.

III. Methodology

In this section, we introduce our active measurement approach to studying the backend interfaces of CDNs. The key idea is to collect the globally distributed frontend addresses and leverage them to uncover their associated backend addresses.

A. Discovering Frontend Addresses

For CDNs that adopt DNS mapping, their DNS systems return the frontend addresses based on the specific location from which the DNS query is sent. Therefore, we leverage open DNS resolvers located over the entire Internet to generate the geographically distributed DNS queries. Censys [15] conducted an Internet-wide scan of DNS using Zmap [16] once or twice a week. We take advantage of this passive DNS dataset and extract the open resolvers from it.[3] Note that the churn on open resolvers is significant [17], so we need to regularly update the freshest open resolver list during the measurement. Figure 2 illustrates our method of discovering the frontend addresses. Specifically, we send the DNS queries to the collected DNS open resolvers, trying to resolve the domain names of CDN customers (❶ and ❷). We use the method presented in [9] to obtain a large number of CDN customers. The nameservers of CDNs reply with the frontend addresses based on the geolocation of the open resolvers, and those open resolvers then relay the resolution result (❸ and ❹). However, there is no guarantee that the frontend addresses are active all the time or the DNS mapping system will reply with all available frontend addresses in one experiment. Also, the CDN providers may continuously assign new IP addresses to the existing PoPs or add new PoPs to their network infrastructures. Therefore, we update the frontend addresses once a week during our experiment.

Collecting frontend addresses from anycast-based CDNs is quite different. Due to the nature of anycast, those CDNs do not respond to DNS queries based on the queries' source IP addresses. Instead, it returns one or more static `A` records for each of the queried domain names. Therefore, we can collect frontend addresses by using our local DNS resolvers, instead of the distributed open resolvers. Besides, since the `A` records are static, we only need a one-time experiment to collect the frontend addresses.

Note that CDN customers may temporarily disable the CDN services during the experiments. As a result, the IP addresses provided by the authoritative nameserver of the CDNs may have several other possibilities, such as the origin address [9] or even a fake address. To validate whether the frontend addresses belong to the CDN providers, we match them with the IP ranges of CDNs to filter out the incorrect ones.

B. Discovering Backend Addresses

Backend addresses are used to fetch content from the origin server. Only when the cache miss happens can we observe the communications between the origin server and backend addresses. Therefore, we intentionally create cache misses at edge servers and collect the backend addresses at our origin server. Figure 3 illustrates our approach to discovering the backend addresses. First, we set up a web server as our origin and sign up CDN services with our domain. Then, we directly send HTTP requests to the frontend addresses and set our domain as the `Host` header so that CDNs will be able to process the requests (❶). For CDNs with a DNS mapping system, those HTTP requests can be delivered directly to all

[2]Note that Akamai completes the server-level load balancing in its DNS mapping. [12]

[3]The scan sends DNS queries to the whole IPv4 address space and resolves a domain controlled by Censys. Its authoritative nameserver always replies with a fixed IP address. By matching this IP address with the DNS responses, we can extract all of the open resolvers from the scan results.

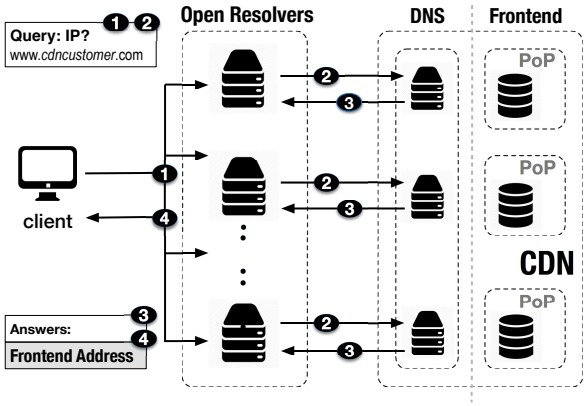

Fig. 2: Discovering Frontend Addresses

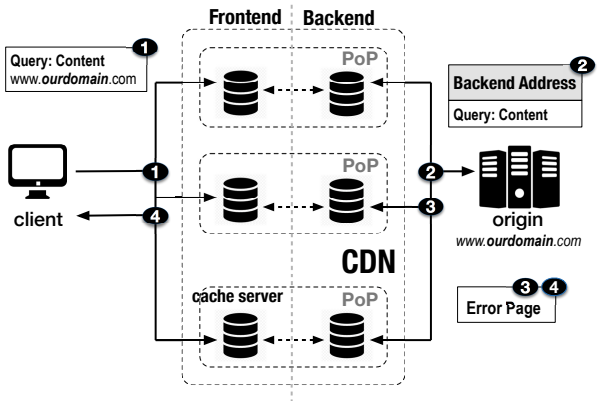

Fig. 3: Discovering Backend Addresses

of their PoPs. However, for anycast-based CDNs, all requests from one probing point will be routed to the PoP that is in proximity to that probing point. Because it is usually difficult to control sufficient probing points that are close to all of the PoPs of a large CDN, we make our best effort to cover the major vantage points on the Internet. To do so, we launch one instance at each of the 15 EC2 regions [18] and measure the PoPs of the anycast-based CDN that can be reached through those EC2 instances.

Within our HTTP requests, we deliberately craft the requested resources to ensure that requests always force the edge servers to fetch the content from our origin server and transmit the probing information to our origin server (❷). Specifically, we embed the frontend address and a random string into the name of the requested resource. Due to the random string, each name of a requested resource is unique. Thus, it is guaranteed that no requested resources are cached in CDN edge servers, which means that edge servers must fetch the content from our origin server. Consequently, HTTP requests, including the backend addresses and the name of requested resources, will be recorded by our origin server, so we collect the backend addresses from our server log file. More importantly, by extracting frontend addresses from the name of requested resources, we uncover the address mappings between the frontend and backend. Finally, our origin server returns `404 Not Found` error pages, since the requested resources are not in the server (❸ and ❹). We observe that CDNs sometimes cache such error pages. Thus, we customize these error pages with no message body to minimize the impact upon our measurement.

C. Limitations

While our approach is able to measure the backend interface of a CDN through its frontend interface, we cannot know the internal structure of the CDN (e.g., how the frontend connects with the backend). In addition, our approach cannot be applied to the web service providers that also have their proprietary content delivery infrastructures (e.g., Google and Facebook) since our approach requires service signup. Moreover, as previously mentioned, our approach requires a large number of globally distributed probes to view the whole platform of anycast-based CDN providers. It is often challenging to conduct a network probing in such a globally distributed manner.

IV. Measurement

We apply our method to CDNs to characterize their backend interfaces. We first describe the CDNs we selected in our measurement and their geolocation hints used for the later analyses. Then, we present the datasets we collected in our measurement.

A. CDN Providers

1) CDN Provider Selection: Our measurement focuses on three CDN providers: CloudFront, Cloudflare, and Fastly. We study these three CDNs with the following three considerations. First of all, they are the key players in the CDN market, since their market shares are ranked in the top 5 of CDN providers [19]. Second, they provide free services for personal usage. Some other top CDN providers, such as Akamai, only provide their services for a business purpose, so we are not allowed to sign up for their services. Third, all of their PoPs are accessible to customers with a free plan, which gives us a chance to assess their entire CDN platforms.

Among these three CDN providers, CloudFront and Fastly leverage the DNS mapping system to reroute web traffic to their distance-proximity PoPs, while Cloudflare adopts anycast routing. As previously discussed, for the anycast-based CDN, we measure the PoPs that can be reached through the EC2 instances. We show the mappings from EC2 regions to Cloudflare PoPs in Table I. Note that the BGP routing path (i.e., AS path) may change during our measurement, so one EC2 instance may reach multiple Cloudflare PoPs at different times.

2) Geolocation: To better understand the deployment of the backend and its association with that of the frontend, we determine the geolocation of the frontend and backend addresses at the PoP level. There are some geolocation databases (e.g., Maxmind [20]) that can convert an IP address to its geolocation. However, previous studies [21], [22] have

TABLE I: Mapping from EC2 regions to Cloudflare PoPs

EC2 Region	EC2 Zone	Cloudflare PoP
Ohio	us-east-2c	IAD
Virginia	us-east-1b	IAD
California	us-west-1c	SJC
Oregon	us-west-2a	SEA
Mumbai	ap-south-1a	BOM / MAA
Seoul	ap-northeast-2c	NRT
Singapore	ap-southeast-1b	SIN
Sydney	ap-southeast-2b	SYD / MEL
Tokyo	ap-northeast-1a	NRT
Central	ca-central-1b	EWR
Frankfurt	eu-central-1b	FRA / PRG
Ireland	eu-west-1b	DUB
London	eu-west-2a	LHR
Paris	eu-west-3c	CDG
São Paulo	sa-east-1a	GRU / EZE

TABLE II: Open Resolver Coverage

Regions	Open Resovlers	Countries and Regions
Africa	0.8M	56
Asia	12.2M	50
Europe	2M	53
North America	1.1M	39
Oceania	0.1M	24
South America	1.4M	14
Total	**17.6M**	**236**

demonstrated that geolocation databases have an acceptable accuracy at only the country level and that it is not reliable at the city level (i.e., the PoP level). Therefore, to achieve our goal, it is insufficient to use these existing geolocation databases. However, we observe that the three CDN providers embed the PoP information into either their hostnames or HTTP headers, which help us determine the geolocation of the frontend and backend addresses with an accuracy of the PoP level.

CloudFront embeds PoP information into the hostnames. By conducting the reverse DNS lookup, we obtain all hostnames of the frontend and backend addresses. CloudFront has 11 PoPs built inside its EC2 region while other PoPs are distributed across the world [23]. We refer to the former as an EC2 PoP and to the latter as a non-EC2 PoP. For an address from an EC2 PoP, its hostname includes an EC2 region code. Therefore, we locate the address by checking the location of the EC2 region. Similarly, CloudFront also embeds an airport code in a hostname of an address from a non-EC2 PoP, which reveals its geolocation.

We gather the geolocation information of Fastly and Cloudflare from HTTP headers. In particular, a Fastly edge server appends its server ID into a self-defined HTTP header, `Fastly-FF` [24], and the server ID has an embedded airport code. As our HTTP request traverses through the Fastly edge servers to our origin server, we can capture a server ID chain

TABLE III: Frontend Addresses

Provider	Addresses	/24 Prefixes	Covered PoPs
Cloudflare	97,829	1,869	149†
CloudFront	124,105	1,346	102
Fastly	10,938	188	48

†Note that Cloudflare's frontend addresses are anycast addresses announced from all of its PoPs. Theoretically, all PoPs can be reached from those addresses. However, due to limited probing points, PoPs that can be reached through our EC2 instances are listed in Table I.

TABLE IV: Frontend-to-Backend Address Pairs

Provider	Address Pairs	
	Total	**Avg. per Experiment**
Cloudflare	5M	147.6K
CloudFront	2.5M	90.8K
Fastly	168.8K	10.4K

from the `Fastly-FF` header. By extracting the first and last server IDs, we obtain the geolocation of both the frontend and backend addresses. Cloudflare embeds an airport code into a self-defined HTTP header, `CF-Ray`, which identifies the PoP from which the HTTP packet originates [25].

B. Datasets

1) Open Resolvers: As shown in Figure 2, the effectiveness of collecting globally distributed frontend addresses in non-anycast CDNs (i.e., CloudFront and Fastly) relies on the coverage of open DNS resolvers. Due to the high churn rate, we use the multiple snapshots from Censys to obtain the available open resolvers. On average, each snapshot contains about 3.6 million open resolvers, and in total, we use approximately 17.6 million open resolvers distributed in 236 countries and regions during the entire experiment period (Table II). The prevalence and wide coverage of open resolvers enable us to discover the entire frontend interface of CDNs worldwide.

2) Frontend Addresses: The dataset of collected frontend addresses is presented in Table III. In total, we obtain about 100K frontend addresses for Cloudflare and CloudFront (more than 1.8K and 1.3K /24 prefixes, respectively), and 10K frontend addresses (188 /24 prefixes) for Fastly.[4] We leverage these wide range frontend addresses to discover the backend addresses.

3) Frontend-to-Backend Address Pairs: We issue HTTP requests to the frontend addresses to collect the CDNs' frontend-to-backend address pairs at our controlled origin server, as described in Section III. Specifically, for CloudFront and Fastly, we issue one HTTP request to every frontend address. For Cloudflare, we send HTTP requests to 10,000 randomly chosen frontend addresses at each PoP. In total, we collect more than 7.6 million address pairs, which are listed in Table IV, through 35 measurement experiments done in a period of three months.

[4]Note that the number of the frontend addresses is not equal to the number of edge servers, due to the server-level load balancing described in Section II.

TABLE V: Backend Addresses

Provider	Addresses		
	Total	Avg. per Experiment	/24 Prefixes
Cloudflare	11,602	1,803	78
CloudFront	863	42	76
Fastly	968	931	48

V. Measurement Results and Analyses

In this section, we present our measurement results and analyses. We first analyze the address space and operation patterns of the backend interface in CDNs and then discuss the PoP association between the frontend and backend.

A. Address Space

Our measurement results on the backend addresses are listed in Table V. In contrast to the large frontend address space, the backend address space is relatively small, and the number of /24 prefixes is also small. Overall, Cloudflare has the largest backend address space, which is more than 10 times larger than that of CloudFront and Fastly. Although CloudFront has the largest frontend address space, the number of its backend addresses collected is the smallest among the three CDNs.

Next, we study the percentage of new backend addresses discovered in each experiment to explore the different operation patterns. The results are shown in Figure 4. We obtain the majority of Fastly's backend addresses in the first experiment, and the rest of the experiments do not contribute much. Meanwhile, the backend addresses of Fastly collected from any experiment do not differ much from its preceding experiment. Therefore, we can see that Fastly has a stable and fixed set of IP addresses as its backend. For CloudFront, every experiment has a non-trivial increase in the total number of backend addresses. The backend addresses from any experiment are almost completely different from its preceding experiment, which implies that CloudFront is constantly changing its backend addresses. As we do not observe a clear trend that the number of backend addresses will stop growing after all 35 experiments, our collection does not cover all of CloudFront's backend addresses. For Cloudflare, all of its backend addresses are mainly collected from six experiments, where their red bars are close to 100%, indicating that new backend addresses collected in these six experiments are completely new sets of backend addresses. Moreover, in comparison to the preceding experiment, the backend addresses of each experiment are either almost identical or almost completely different. Thus, we speculate that Cloudflare has six different sets of backend addresses in those measured PoPs and rotates them over time. After the 18th experiment, the total number of backend addresses no longer has any significant increase, implying that we have collected the majority of backend addresses in the PoPs we measured.

In summary, Fastly assigns its backend a set of stable and fixed addresses; however, Cloudflare rotates six sets of backend addresses and the CloudFront's backend addresses are highly dynamic in an irregular manner. Note that the dynamic IP assignment lowers the security risk of an edge server, as it is more difficult for adversaries to locate the server.

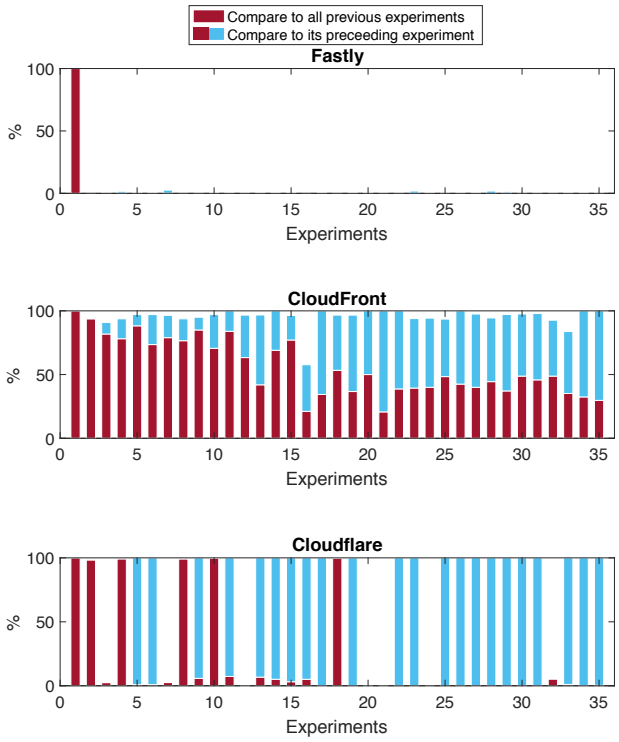

Fig. 4: Percentage of New Backend Addresses per Experiment

B. Backend Addresses of PoPs

Here we examine the backend addresses distributed at each PoP. In our experiment, we may have an incomplete view of some PoPs if (1) these PoPs are temporarily closed for maintenance, or (2) our requests are occasionally directed to these PoPs due to temporary misconfigurations or the oscillation of the BGP routing path. Therefore, in the following analysis, we show only the PoPs whose backend addresses can be observed in at least 80% of experiments. The results are presented in Figure 5.

For Fastly, the total number of backend addresses at each PoP obtained from all of the experiments is almost equal to the average number observed per experiment. This implies that Fastly's backend addresses are stable at every single PoP. Also, we observe that the backend addresses in every Fastly PoP are all in one /24 prefix, while the least significant byte of IPs falls into the range of `.20`$\sim$`.51` and its maximum number of backend addresses of a PoP is 32. For CloudFront, the average number of backend addresses per experiment is extremely low for each PoP. Its maximum number is only 4 (found at the EC2 PoP in Virginia). However, its total number is much higher than the average at all CloudFront PoPs, indicating that the backend addresses at all CloudFront PoPs are highly dynamic. The total and average numbers of backend addresses at each Cloudflare PoP are significantly higher than those of Fastly and CloudFront. For every Cloudflare PoP, it also rotates around six different sets of backend addresses over time, and the total

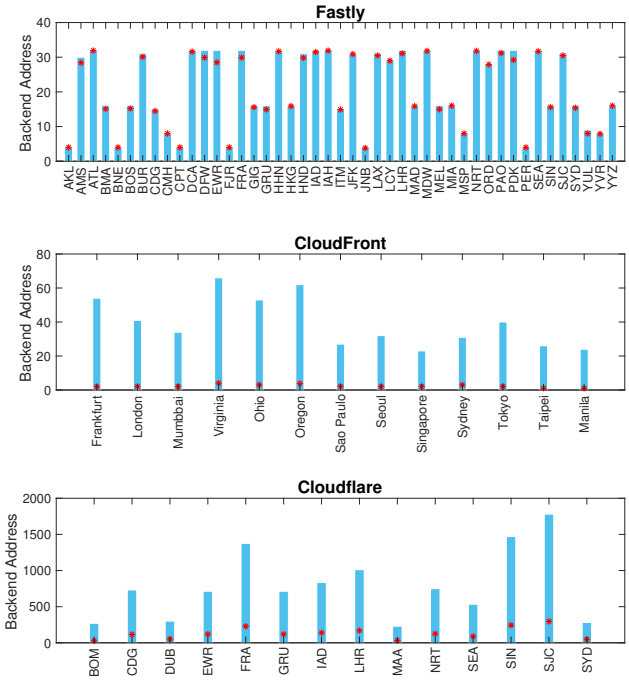

Fig. 5: Backend Addresses of PoPs. The * represents the average number of backend addresses per experiment at each PoP.

number of backend addresses is approximately six times the average. Hence, the operation patterns observed from Section V-A are also confirmed in each of the PoPs.

Furthermore, as we mentioned in Section II, with the current implementation of the CDN load balancing, the frontend address may not correspond to a particular edge server. However, we notice that the number of backend addresses collected from a PoP reveals the number of simultaneously working edge servers at that PoP. A recent study [14] investigates the network infrastructure of Fastly, where the most common PoP hosts 32 edge servers, confirming our observation on Fastly.

C. Co-PoP of Frontend and Backend Addresses

When a cache miss happens at a PoP, its edge server could (1) directly retrieve the content from the origin or (2) forward the request to another PoP. In the first scenario, the frontend and backend addresses will be observed in the same PoP. In the second scenario, the frontend and backend addresses will be observed in different PoPs.

In our experiments, we observe that in Cloudflare and Fastly, a backend address and its associated frontend address are always in the same PoP, which means that they would not forward the HTTP requests to other PoPs.[5] In contrast, more than 97% of CloudFront address pairs have their frontend and backend addresses located in different PoPs. More importantly, all of these backend addresses are in 11 EC2 PoPs while

[5]Note that the shielding services [26] from Fastly would allow its PoP to fetch content from another fixed and preconfigured PoP, but it requires a customer configuration.

Fig. 6: Association between EC2 PoPs and non-EC2 PoPs. The locations of EC2 PoPs are listed in the left column, and the number of associated non-EC2 PoPs is shown in the parentheses.

their associated frontend addresses are all in non-EC2 PoPs. Therefore, we speculate that CloudFront has a two-level PoP infrastructure, where the EC2 PoPs are second-level PoPs and are exclusively used for receiving requests from the first-level PoPs (i.e., the non-EC2 PoPs) and fetching content from the origins. We visualize the association between EC2 PoPs and non-EC2 PoPs in Figure 6. The circular markers on the map represent the locations of non-EC2 PoPs, and the colors represent different groups of non-EC2 PoPs associated with the same EC2 PoP. We can clearly see that the non-EC2 PoPs associated with the same EC2 PoP are geographically close to one another, and that the EC2 PoP is geographically close to its associated non-EC2 PoPs.

Given that the EC2 PoPs are essentially Amazon's cloud data centers hosting a large number of web domains that also adopt CloudFront, such a two-level PoP design improves performance by reducing end-to-end latency and augmenting cache storage. Furthermore, the two-level PoP infrastructure lowers the workload of the origin server and improves the cache hit ratio. Specifically, we assume multiple users request the same new content from multiple different first-level PoPs, and these first-level PoPs are associated with the same second-level PoP. In this case, the CDN will only need to fetch the requested content from the origin once (i.e., one cache miss) because the second-level PoP will reply to each first-level PoP with the requested content once it receives one copy from the origin. Under the same scenario, a CDN without such a two-level PoP (i.e., one-level PoP) infrastructure may produce multiple cache misses and has to fetch the requested content multiple times from the origin.

D. Address Expansion

The frontend addresses we collected are obviously not continuous, and here we investigate whether the other addresses in those /24 prefixes actually perform the same function and whether the expanded frontend addresses can give us more backend addresses. To do so, we apply the same technique described in Figure 3 by sending the HTTP requests to IP addresses in all frontend's /24 prefixes and examining received requests at our origin server.

TABLE VI: Address Expansion

Provider	Frontend Addresses	Frontend Addr. Coverage†	Backend Addresses
Cloudflare	476,094	99.5%	1,814
CloudFront	342,329	99.3%	55
Fastly	41,821	86.9%	977

†Frontend address coverage represents the percentage of the valid frontend addresses in IP addresses of all frontend's /24 prefixes.

The results are shown in Table VI. In total, we successfully discover 860,244 valid frontend addresses from the three CDN providers, which is 3.7 times the previously collected ones. More importantly, almost all of the addresses in the prefixes of Cloudflare and CloudFront are valid frontend addresses, and 86.9% of them are valid for Fastly. However, although we expand a significantly larger number of frontend addresses, the number of backend addresses that they probed is almost the same as the number of average addresses per experiment presented in Table V. Therefore, we contend that the frontend addresses used in our measurement probe all the of simultaneously working backend addresses at each experiment and the CDNs usually assign a limited number of backend addresses to handle the backend-origin connections.

VI. Performance and Security Implications of the Backend

The performance and security of CDNs are critical to CDN customers and end-users. In this section, we analyze the performance and security implications of the CDN platforms from the perspective of their backend interfaces.

A. Performance Implications

How the backend interfaces and origin servers are deployed significantly impacts the content fetching time when a cache miss happens at the CDN or when a non-cacheable content is requested. This is because a frontend is usually close to end-users so that the latency between the frontend and end-users is small, while the backend needs to fetch the content from the origin server, which could be located far away from it. Therefore, we assess the RTTs and AS paths between origin servers and backend PoPs to study the performance implications of the CDN backend.

Nowadays, with the rapid growth of the cloud and its elastic computing services, website owners usually host their web services in the cloud, such as Amazon Web Services (AWS). Therefore, to emulate the location of an origin server, we set up 15 vantage points at each of the regions of the AWS EC2. Then, we issue traceroutes from our vantage points to one backend address in each of the /24 prefixes of the CDN backend interfaces. We choose /24 prefixes because the traceroutes to the addresses in the same /24 prefixes typically traverse the same AS path and their RTTs are similar to one another. Finally, as a PoP may have multiple /24 prefixes, we group the traceroute results by PoPs and average them. For CloudFront, we issue traceroutes to the backend addresses at the second-level PoPs because, according to our observations in Section V-C, those addresses handle more than 97% of the backend connections. However, in the experiment, we notice that those backend addresses are configured not to respond to the traceroute queries. Nevertheless, since we know that the second-level PoPs of CloudFront are all located at the 11 EC2 regions, we then launch instances at the same regions as those backend addresses and issue traceroutes from our vantage points to these instances to obtain approximate RTTs and AS paths.

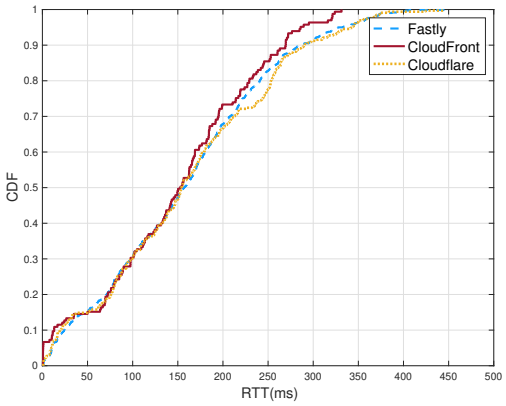

Fig. 7: CDF of RTTs from EC2 to Backend Interfaces

1) Round-Trip Time (RTT): We extract the RTTs from our traceroute dataset and present the results in Figure 7. Overall, the RTTs from our vantage points to a backend of the CDNs do not differ much among the three CDNs. This is because they all have the globally deployed backend PoPs and similar ratios of backend PoPs on each continent. The noticeable difference appears at the tail of the CDF, where Cloudflare and Fastly have a longer tail than that of CloudFront. In Figure 8, we further show the average RTTs from vantage points to all of the backend PoPs for each CDN. We can observe that the vantage points in North America produce the shortest average RTTs. The RTTs obtained from vantage points in Europe are slightly longer while the vantage points in Asia Pacific and South America generate the longest RTTs. Thus, for a website owner who aims to serve content to global users through these three CDNs, placing the origin server in North America is an optimal strategy to reduce latency from the origin server to the CDN backend PoPs.

2) AS Path: The traceroute dataset contains sequences of IP addresses from our vantage points to the CDN backend interfaces. To analyze the connections at the AS level, we convert the IP addresses into the AS numbers. To do so, we first remove the IP addresses with private and reserved IP prefixes, since those prefixes should not be announced by BGP routers. Then, we filter out the IP addresses within the prefixes[6] of Internet Exchange Points (IXPs) to obtain the

[6]We identify the IP prefixes of IXPs using the CAIDA's IXP dataset https://www.caida.org/data/ixps/.

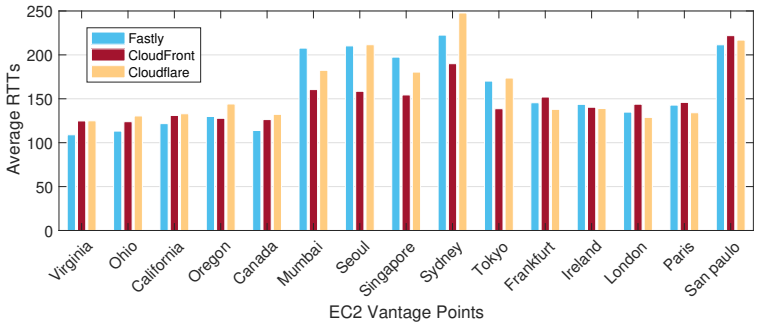

Fig. 8: Average RTTs from EC2 to Backend Interfaces

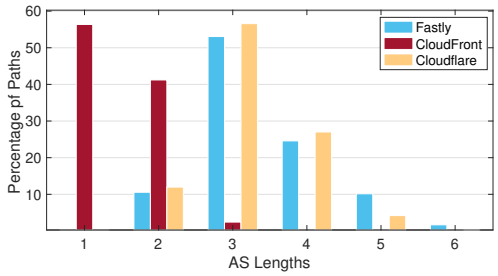

Fig. 9: AS Path Lengths from EC2 to Backend Interfaces

ASes that are directly connected at the IXP [27]. In addition, we remove the unresponsive hops, which account for 4.8% of the total hops. Finally, we map the remaining IP addresses to the AS numbers using the RouteView BGP dataset [28].

Figure 9 shows that the AS path length is shortest from the vantage points to CloudFront, and more than 95% of the path lengths are less than 3. Also, all ASes along the paths belong to Amazon itself. Note that such observation from CloudFront is largely due to the fact that we emulate the location of the origin servers at the EC2 while the backend PoPs of CloudFront are also located at EC2. Thus, even if the vantage points and CloudFront backend PoPs are both distributed across the world, Amazon still routes the traffic within its own network. Moreover, the distribution of AS path lengths of Cloudflare and Fastly are roughly the same, and they are mostly concentrated at 3 or 4. In comparison with AS path lengths from end-users to the frontend, which are usually 2 or 3 [29], the AS path lengths from the origin server to a CDN backend is 1 AS longer. The selection of AS paths between the origin server and a CDN backend is a key factor affecting RTTs. A shorter AS path would possibly shorten the RTT, which helps to improve the CDN performance. Our observation implies that CDN providers still have space to improve their backend performance.

B. Security Implications

Unlike the frontend connections, the backend communications are typically hidden from end-users. Here, we study the port management of backend addresses and analyze its security implications.

Given the purpose of fetching content from the origin, a backend server acts as a reverse proxy to initiate the TCP connection.[7] Therefore, a backend address does not need any port open to the public. The unnecessary open ports may increase the attack surface of the CDN infrastructure. In order to study the open ports of the backend addresses, we leverage Nmap to perform a TCP port scan on the 1,000 most common ports [30]. In particular, we send SYN packets to the scanned ports of all backend addresses we collected. If an SYN-ACK packet is received, we infer that the port is open; otherwise, an RST packet indicates that the port is closed. We then consider that the packets are dropped by the CDN's firewall if there is no response for two attempts, indicating that the port is also unreachable by the public.

TABLE VII: TCP Open Ports

Provider	TCP Open Ports	Backend Address	Percentage
Cloudflare	-	-	-
CloudFront	HTTP/80 HTTPS/443	365	42.3%
	RTMP/1935	29	3.7%
Fastly	HTTP/80 HTTPS/443	946	97.7%

1) Open Ports on Backend Addresses: As shown in Table VII, all probes sent to Cloudflare's backend addresses are dropped by the firewalls, so no open port is found. For CloudFront, its ports HTTP/80 and HTTPS/443 are open at 365 of the backend addresses, among which 29 of them are also opening port RTMP/1935. Interestingly, these 365 backend addresses are all from non-EC2 PoPs. In other words, all backend addresses from EC2 PoPs have no open ports. However, for Fastly, 97.7% of the backend addresses are opening ports HTTP/80 and HTTPS/443 to the public.

2) Backend Accessibility: Since we identify that the ports are opening on part of the backend addresses of CloudFront and Fastly, we test whether these backend addresses are allowed to be used as valid (frontend) addresses to process our HTTP requests. To do so, we send the HTTP requests directly to these backend addresses, asking for the landing pages of the CDN's customers. By doing so, we identify that all backend addresses are able to process our requests and provide the `200 OK` responses, which implies that these backend addresses perform the same function as that of frontend addresses.

In addition, we also directly send HTTP requests to these backend addresses to see how the CDNs would process these requests to reach our origin server. Similar to the frontend address cases, the majority of requests sent to CloudFront are forwarded to EC2 PoPs before arriving at our origin while requests sent to the backend addresses in TPE (Taipei)

[7]We note that the three CDN providers we studied are all with the *pull* mode and do not support the *push* mode, in which the origin server initiates the connections and actively uploads the content to the CDN. With the push mode enabled, CDNs need to assign specific domains/edge servers to receive the push connections from origins. We mainly focus on the pull mode because it has been dominant in today's CDN ecosystem.

and JAX (Jacksonville) are directly forwarded to us with the addresses in their own PoPs. Fastly forwards requests to our origin using a different backend address in the same prefix from the same PoP. Moreover, when we send multiple requests to the same backend address and examine the received requests at our origin server, we observe that the first server ID in the `Fastly-FF` headers of those requests is always the same, which indicates that all requests are received and processed by one particular edge server. This observation is completely different from the scenario of sending HTTP requests to Fastly's frontend address, where the first server ID in `Fastly-FF` headers is constantly changing due to the load balancing system. In other words, if we specify Fastly's backend addresses to fetch the content, it would completely bypass Fastly's load balancing system. This could be a potential security risk for Fastly's platform, since adversaries may intentionally increase the workload of a particular edge server.

VII. Related Work

A. CDN Deployment and Performance

Huang *et al.* [3] conducted extensive measurements to evaluate the performance of two large CDN platforms (Akamai and Limelight) and illustrated two design philosophies for CDNs: (1) enters deep into ISPs and (2) brings ISPs to home. They collected the IP addresses in DNS responses (i.e., the frontend addresses) and quantified their delay performance and server availability. Scott *et al.* [5] presented a toolchain, called Satellite, for studying the CDN platforms and measuring the web infrastructure deployment and accessibility. Pujol *et al.* [2] studied a commercial CDN's backend interface by analyzing logs from the CDN's edge servers. However, our approach provides an active examination on the backend interface of CDNs without accessing the logs.

Ager *et al.* [4] designed an automated approach based on DNS resolution and BGP routing information to discover and classify content hosting and delivery infrastructures and found that a large amount of content is exclusively served from particular geographic regions and ASes. Chiu *et al.* [29] investigated the length of the AS path between end-users and the frontend interface of the content providers, including a CDN and popular web services, and identified that many end-users are only one AS hop away from the frontend interface. Our work studies the AS path length between the CDN backend interface and the origin servers, showing that it is one hop longer than the AS path length between end-users and a CDN's frontend.

Calder *et al.* [31] examined the performance of an anycast-based CDN and identified that nearly 20% of the clients are being redirected to suboptimal PoPs. Chen *et al.* [32] studied the practical impact of the EDNS-Client-Subnet (ECS) extension in Akamai's end-user mapping system and showed that significant performance benefits can be achieved with ECS when selecting proximal edge servers. Chen *et al.* [33] investigated the effect of the deployment of edge servers on the user-perceived performance and found that the content fetching time between a CDN and origin servers is a dominant factor for end-to-end performance. Our approach allows a third party, such as CDN customers and researchers, to explore such a critical factor.

B. CDN Security

Liang *et al.* [34] systematically investigated the practical issues when composing HTTPS with CDN because CDN cuts the secure communication paths offered by HTTPS. Chen *et al.* [24] presented the forwarding-loop attacks in CDNs, which allow adversaries to create request loops within one CDN or across multiple CDNs to exhaust the source ports and bandwidth resources of CDNs. Gilad *et al.* [35] designed an on-demand CDN system in which website administrators can build their own CDNs on top of the widely deployed cloud infrastructure to defend against DDoS attacks.

Vissers *et al.* [7] explored eight attack vectors of origin exposure on the DPS offered by CDNs that could be exploited by adversaries to bypass a CDN's request rerouting and directly launch the DDoS attack on the origin server. Subsequently, Jin *et al.* [9] revealed the residual resolution vulnerability, in which adversaries could acquire the protected origin IP address from previously adopted DPS providers. Hao *et al.* [36] presented the redirection hijacking attack in CDNs, in which adversaries can maneuver a DNS mapping system to disrupt CDN operations.

VIII. Conclusion

In this paper, we designed an active approach to discovering the backend interface of CDNs. To the best of our knowledge, this is the first attempt to reverse-engineer and characterize the backend interface of CDNs without accessing the logs from edge servers of particular CDNs. Such an approach allows us to probe the backend interfaces of multiple CDN platforms from the outside. We then presented a comprehensive measurement study on the backend interfaces of three leading CDN platforms, including CloudFront, Cloudflare, and Fastly. Our measurement results show that the backend address space is relatively limited compared to the wide range of frontend address space. Also, we observed different operation patterns of backend addresses in the three CDNs. Moreover, by analyzing the backend addresses and their associated frontend addresses, we investigated the PoP association between the frontend and backend and visualized it for CloudFront's two-level PoP infrastructure. In addition, we explored the performance implications of CDN backends by assessing the RTTs and AS path lengths between origin servers and backend PoPs. Finally, we examined the port management of backend addresses, and then we analyzed the backend accessibility and its security implications.

Acknowledgment

We would like to thank the anonymous reviewers for their insightful and detailed comments, which helped us improve the quality of this paper. This work was supported in part by the U.S. NSF grants CNS-1618117 and DGE-1821744, as well as ARO grant W911NF-19-1-0049.

REFERENCES

[1] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2016-2021," 2017.

[2] E. Pujol, P. Richter, B. Chandrasekaran, G. Smaragdakis, A. Feldmann, B. M. Maggs, and K.-C. Ng, "Back-Office Web Traffic on The Internet," in *ACM Internet Measurement Conference (IMC)*, 2014.

[3] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and Evaluating Large-Scale CDNs (Withdrawn)," in *ACM Internet Measurement Conference (IMC)*, 2008.

[4] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, "Web Content Cartography," in *ACM Internet Measurement Conference (IMC)*, 2011.

[5] W. Scott, T. Anderson, T. Kohno, and A. Krishnamurthy, "Satellite: Joint Analysis of CDNs and Network-Level Interference," in *USENIX Annual Technical Conference (ATC)*, 2016.

[6] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "Open Connect Everywhere: A Glimpse at the Internet Ecosystem through the Lens of the Netflix CDN," in *ACM SIGCOMM Computer Communication Review (CCR)*, 2018.

[7] T. Vissers, T. van Goethem, W. Joosen, and N. Nikiforakis, "Maneuvering Around Clouds: Bypassing Cloud-based Security Providers," in *ACM Conference on Computer Communication Security (CCS)*, 2015.

[8] M. Jonker, A. Sperotto, R. van Rijswijk-Deij, R. Sadre, and A. Pras, "Measuring the Adoption of DDoS Protection Services," in *ACM Internet Measurement Conference (IMC)*, 2016.

[9] L. Jin, S. Hao, H. Wang, and C. Cotton, "Your Remnant Tells Secret: Residual Resolution in DDoS Protection Services," in *IEEE International Conference on Dependable Systems and Networks (DSN)*, 2018.

[10] A. Barbir, B. Cain, R. Nair, and O. Spatscheck, "Known Content Network (CN) Request-Routing Mechanisms," *IETF RFC 3568*, 2003.

[11] J. Abley, A. Canada, and K. Lindqvist, "Operation of Anycast Services," *IETF RFC 4786*, 2006.

[12] B. M. Maggs and R. K. Sitaraman, "Algorithmic Nuggets in Content Delivery," in *ACM SIGCOMM Computer Communication Review (CCR)*, 2015.

[13] Noction, "BGP and equal-cost multipath (ECMP)," https://www.noction.com/blog/equal-cost-multipath-ecmp.

[14] J. T. Araújo, L. Saino, L. Buytenhek, and R. Landa, "Balancing on the Edge: Transport Affinity without Network State," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[15] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A Search Engine Backed by Internet-Wide Scanning," in *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[16] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide Scanning and Its Security Applications," in *USENIX Security Symposium*, 2013.

[17] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, "Going Wild: Large-Scale Classification of Open DNS Resolvers," in *ACM Internet Measurement Conference (IMC)*, 2015.

[18] https://docs.aws.amazon.com/general/latest/gr/rande.html.

[19] https://www.datanyze.com/market-share/cdn/.

[20] https://www.maxmind.com.

[21] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye, "IP Geolocation Databases: Unreliable?" *ACM SIGCOMM Computer Communication Review (CCR)*, 2011.

[22] M. Gharaibeh, A. Shah, B. Huffaker, H. Zhang, R. Ensafi, and C. Papadopoulos, "A Look at Router Geolocation in Public and Commercial Databases," in *ACM Internet Measurement Conference (IMC)*, 2017.

[23] https://aws.amazon.com/cloudfront/details/.

[24] J. Chen, J. Jiang, X. Zheng, H. Duan, J. Liang, K. Li, T. Wan, and V. Paxson, "Forwarding-Loop Attacks in Content Delivery Networks," in *Network and Distributed System Security Symposium (NDSS)*, 2016.

[25] Cloudflare, "How does Cloudflare handle HTTP Request headers?" https://support.cloudflare.com/hc/en-us/articles/200170986-How-does-Cloudflare-handle-HTTP-Request-headers.

[26] Fastly, "Shielding," https://docs.fastly.com/guides/performance-tuning/shielding.html.

[27] K. Chen, D. Choffnes, R. Potharaju, Y. Chen, F. Bustamante, D. Pei, and Y. Zhao, "Where the Sidewalk Ends: Extending the Internet AS Graph Using Traceroutes from P2P Users," in *ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2009.

[28] http://archive.routeviews.org/bgpdata/.

[29] Y.-C. Chiu, B. Schlinker, A. B. Radhakrishnan, and R. G. Ethan Katz-Bassett, "Are We One Hop Away from a Better Internet?" in *ACM Internet Measurement Conference (IMC)*, 2015.

[30] Nmap, "Well Known Port List: nmap-services," https://nmap.org/book/nmap-services.html.

[31] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in *ACM Internet Measurement Conference (IMC)*, 2015.

[32] F. Chen, R. K. Sitaraman, and M. Torres, "End-User Mapping: Next Generation Request Routing for Content Delivery," in *ACM SIGCOMM Conference*, 2015.

[33] Y. Chen, S. Jain, V. K. Adhikari, and Z.-L. Zhang, "Characterizing Roles of Front-end Servers in End-to-End Performance of Dynamic Content Distribution," in *ACM Internet Measurement Conference (IMC)*, 2011.

[34] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS Meets CDN: A Case of Authentication in Delegated Service," in *IEEE Symposium on Security and Privacy (S&P)*, 2014.

[35] Y. Gilad, A. Herzberg, M. Sudkovitch, and M. Goberman, "CDN-on-Demand: An Affordable DDoS Defense via Untrusted Clouds," in *Network and Distributed System Security Symposium (NDSS)*, 2016.

[36] S. Hao, Y. Zhang, H. Wang, and A. Stavrou, "End-Users Get Maneuvered: Empirical Analysis of Redirection Hijacking in Content Delivery Networks," in *USENIX Security Symposium*, 2018.