Network Data Scientist Challenge

1. CDN analysis

CDN Identification and Ranking

In order to identify when a domain uses a CDN infrastructure, it is important to define what use a CDN means.
Indeed, almost every website use at least a content from a CDN provider e.g., content from *googleanalytics*, a well-known tracking service by Google. For this reason, I define as domain based on CDN, a domain thta downloads most of the content on the main webpage from a CDN.

All the time we access a webpage with the browser, we download many objects like pictures, scripts, video, etc… To download them the browser needs the object URL, and the IP address of the server that host the object. By exploiting these two information and using different techniques, it is possible to identify whether a server belongs to a CDN or not. Therefore discovering if a domain is based on CDNs.

To map servers with their CDN provider, we analyzed the following techniques:

1. Network owner as returned by the *whois* database
2. CNAME of each domain as returned by the DNS
3. Subnet classifier

Before to discuss each technique, it is important to remark the notion of domain based on CDNs.
For our tool, we define a domain as based on a CDN if:
A domain downloads most of the content from a CDN. In particular, if the domain downloads at least 50% of the main page objects from a single CDN provider.
This approach may penalize pages hosted on a CDN provider that downloads a lot of content from many other providers. However, we decide to use this notion as it give us a subset of domains with strong relationship with CDNs.

Network owner as returned by the whois database

This first method rely on the information of the *whois* database. It is possible to query this database by using an IP address. Then, it returns several information about it such as: the network range, the network name, the origin AS number, the organization ID, etc....
About *whois* database, authors in [1] shown that this database is not always reliable. In addition, during the development of our tool we meet some cases where the *whois* database was suggesting as owner a different operator with respect to the one using the network. For instance, the IP address 193.206.135.153 according to *whois* belongs to Consortium "GARR, IT", however, it is used by Akamai. The proof is that many objects downloaded from this serve have in the URL "*akamaized.net*", which is a well know CDN signature of Akamai.
In addition, this solution require to known a priori which are the CDN providers, in order to highlight them form the network owner field.

CNAME

This methodology rely on the DNS protocol. When we send a DNS query, it is possible that the DNS server will reply with a Canonical name (CNAME) field. By following all the CNAME up to the Authoritative answer it is possible to find information about an alias that often contains keyword of a CDN provider [2][3].
For instance, Listing 1 shows an example of domain that apparently do not refer to a CDN, but following the CNAME chain hand up in an *Akamai* CDN.

```
host s-passets-cache-ak0.pinimg.com
s-passets-cache-ak0.pinimg.com is an alias for s.pinimg.com.
s.pinimg.com is an alias for s.pinimg.com.edgekey.net.
s.pinimg.com.edgekey.net is an alias for e4456.dscd.akamaiedge.net.
```

Listing 1: CNAME chain

As the previous solution, also this one requires the knowledge a priori of the possible CDN signatures. In our tool we leverage the CDN signatures found in [4].

Subnet Classifier

The network classification solution is driven by the idea that: if a domain uses a CDN to download its objects, its objects will be downloaded from a server belonging to a subnet that will be used by many other domains.
However, to prove this statement, it is required a first deep understating on how to define a subnet. A first option is to use a /24 as a default subnet, since it represent the smallest commonly used subnet.
To validate this approach, it is required a dataset composed by subnets and the respective ground truth to train our classifier. However, since we do not have such a dataset, and the creation of it require a long manual investigation, we did not exploit this solution.

Listing 2 presents an example of possible problem where both *ajax.aspnetcdn.com* and *mainpage.pixfs.net* refer to the same network 93.184.221.0/24. However, while the first domain has as CNAME *mscomajax.vo.msecnd.net,* which is a *Microsoft Azure* CDN signature, the second one has only as CNAME *cs57.wpc.epsiloncdn.net,* which is not a signature for *Microsoft Azure* CDN.

```
host ajax.aspnetcdn.com
ajax.aspnetcdn.com is an alias for mscomajax.vo.msecnd.net.
mscomajax.vo.msecnd.net is an alias for cs3.wpc.v0cdn.net.
cs3.wpc.v0cdn.net has address 93.184.221.200

host mainpage.pixfs.net
mainpage.pixfs.net is an alias for cs57.wpc.epsiloncdn.net.
cs57.wpc.epsiloncdn.net has address 93.184.221.79
```

Listing 2: Same /24 subnet vs different content provider

By taking into account the previous analysis, we decided to implement the solution based on the CNAME chain. In addition, our tool gives the possibility to the user to exploit partially the *Network Classifier* solution. Indeed, it is possible to enable a *cache* which in case it finds that a network belongs to any CDN provide, will assign the /24 network to that CDN provider. Therefore, if the tool will find again the same /24 network during the analysis, it will immediately assign the label without further investigations.

Algorithm workflow

In this section we briefly describe how our tool works, dividing the process in its main steps:

1. The process starts by reading top 500 websites as ranked by Alexa
2. It boots a proxy server namely *browsermob-proxy.* This proxy is required to retrieve the *har* file for each website. This file contains information about which objects are downloaded by the browser when a user accesses the main domain page. Among the information available, for each object we can find its URL, and the server IP address.
3. Then, for each domain, the tool uses *phantomJS* browser to visit the main domain page. The choice of use *phantomJS* is to keep the prototype as light as possible. Since *phantomJS* is a headless browser, it does not require opening any graphic interface i.e., making the process faster. One important parameter to set during the setup is the *set_page_load_timeout* parameter. It defines how long the page will be loaded before triggering a timeout and stop loading the page. Since some domains may require long time to load, we begin our analysis by using 15 seconds for all domains. In case of a domain will not download enough information in time the tool will store the domain into a vector for further investigation with longer timeout.
4. For each object of the selected domain we first check if the server IP address belongs to a well-known subnet (in case in the cache is enabled). If it does not belong to any well-known subnet, then, we investigate the FQDN itself to check whether it contains a well-known CDN signature or not. In case it does not have any signature, we proceed with the DNS analysis to recursively check all possible CNAME to search a signature. If any CNAME have a well-known signature, the signature owner is returned. Otherwise, we check all CNAMEs and in case no signature is find, we decide that the object does not belong to any CDN. Therefore, also the network is marked as not belonging to any CDN (if the cache is enabled). If the case is enabled, and in a second moment we find that the unknown subnet belongs to a CDN than we update the owner.
5. Finally, for each domain the process dumps in a file all the objects with the corresponding IP address and CDN Owner, or a "-" mark if the network does not belong to any CDN provider. If more than 50% of the main page objects are downloaded from a single CDN provider, then we save the domain in a file containing the information about CDN domains, which CDN provider they use, how many objects are download from that provider, and how many in total.

RESULTS WITHOUT CACHE

In this section, we describe the main result of this first experiment without using the caching techniques. Table 1 shows the breakdown of CDN providers and the number of domain for the experiment without cache.

CDN Provider	#Domains
Akamai	78
Google	69
Amazon	21
Fastly	11
EdgeCast	6
ChinaCache	5
Facebook	3
ChinaNetCenter	3
Twitter	2
Yahoo	2
Instart	2
Highwinds	2
WordPress	1
NetDNA	1
Level	1
CDNetworks	1
CDN77	1

Table 1: Result with the cache disabled

As we can see in total we identity 209 domains served by CDNs, with most of them hosted by Akamai CDN, followed by Google. Surprisingly, only 21 domains out of 209 use Amazon CDN. In general, this result shows more than a 40% of the domains use a CDN to offer their content.

RESULTS WITH CACHE

This result instead take into account the caching solution. Table 2 present the new breakdown, where 234 domains are present.

CDN Provider	#Domains
Akamai	83
Google	71
Amazon	23
Fastly	11
Taobao	9
Edgecast	8
ChinaNetCenter	6
Facebook	4
ChinaCache	4
Yahoo	3
Instart	2
Highwinds	2
Twitter	1

NetDNA	1
Microsoft	1
Limelight	1
Level	1
Incapsula	1
CDNetworks	1
CDN77	1

Table 2: Result with the cache enabled

The results of these two experiments show that nowadays CDNs are widely used as between 40 to 50% of TOP 500 domains use a CDN. However, a longer study would be fundamental to have deeper understanding of CDNs usage. In particular, we refer to the usage of unsupervised techniques to discover the CDN though the subnet classifier.

TIMINING PERFORMANCE

In order to evaluate the timing performance we exploited the python curl library[1], which offers direct access to the timing information we need[2]. To perform this experiment, we used as CDN list the NOCACHE version. Then, for each domain, we download the main page. To download this page we first try to contact the server with the *HTTPS* protocol, then, in case we fail, we use HTTP. We rely first on HTTPS to include the TLS setup time. However, in some cases this option does not work as for all domains belonging to the ChinaCache CDN.

Figure 1 reports the average response time for each CDN provider. As we can see most commonly used CDNs have all a similar average value for the Time To First Byte. Only Fastly and ChinaCache require a slightly longer time to return the first byte. By evaluating Figure 2 for ChinaCache CDN provider, we can see that most of the time to get the First Byte is spent waiting the content. Indeed, it represents 38% of overall time, as shown in Figure 3 and 58% of the Time To Fist Byte.

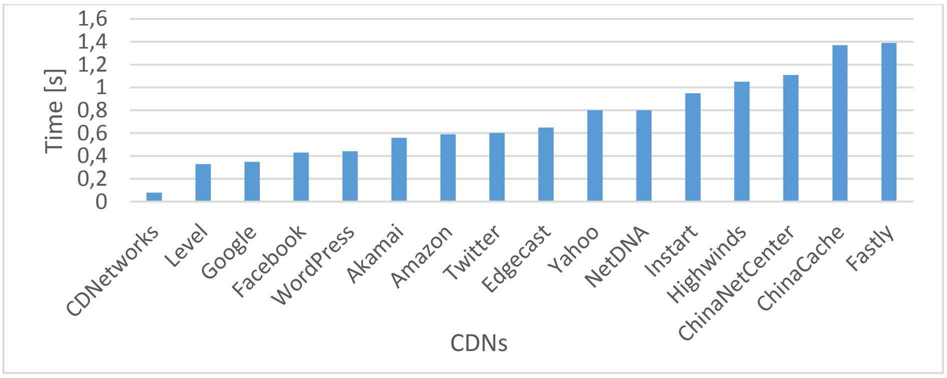

Figure 1: Time To First Byte for each CDN provider

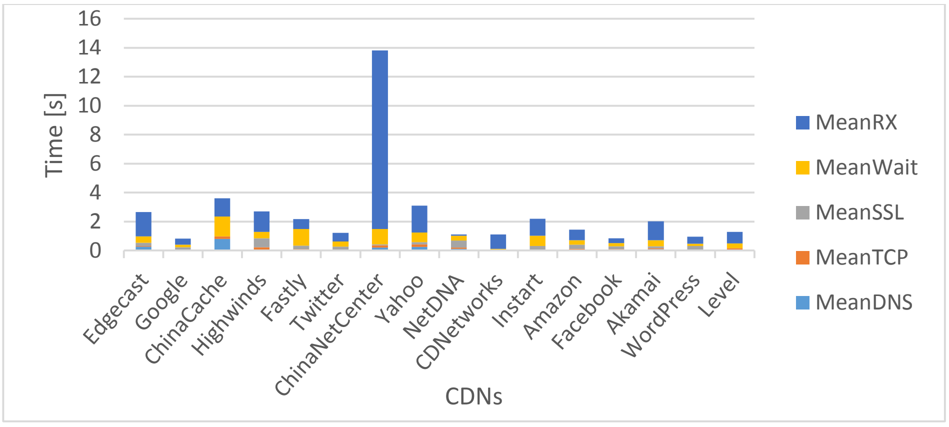

Figure 2: Breakdown of all timing

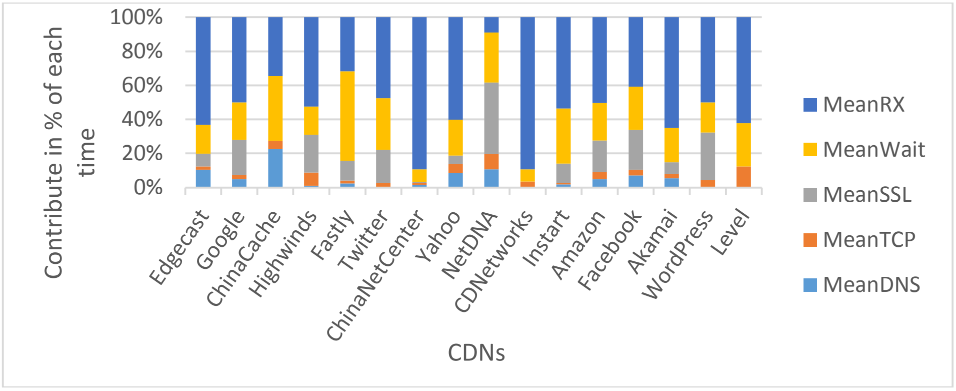

Figure 3: Contribute in % of each time per CDN

[1]http://pycurl.io/

[2]https://photodelphinus.com/Imagenes/carga/cURL/Manual/libcurl/curl_easy_getinfo.html

2. BGP Analysis

In order to perform the BGP analysis we decided to exploit *BGPStream*[3] a tool developed in CAIDA, UCSD, San Diego. We decided to use this tool, instead of rely on the Origin AS returned by *whois* database, because *BGPStream* has more than 30 collectors worldwide located giving us a very update picture of the BGP information.
Figure 4 shows the number of distinct domain for each Origin AS. As we can see most of the domains are hosted by a few origin AS number, in particular the TOP 10 ASNs are listed in Table 3. This table demonstrates that most of domains have an origin AS number owned by a CDN provider. Indeed, in the TOP 10 we find Google, CloudFlare, Amazon, Fastly, Akamai, and EdgeCast respectively. However, differently from what our tool highlighted according to this analysis CloudFlare hosts many domains. This result suggests that CloudFlare hosts also CDNs of other providers.

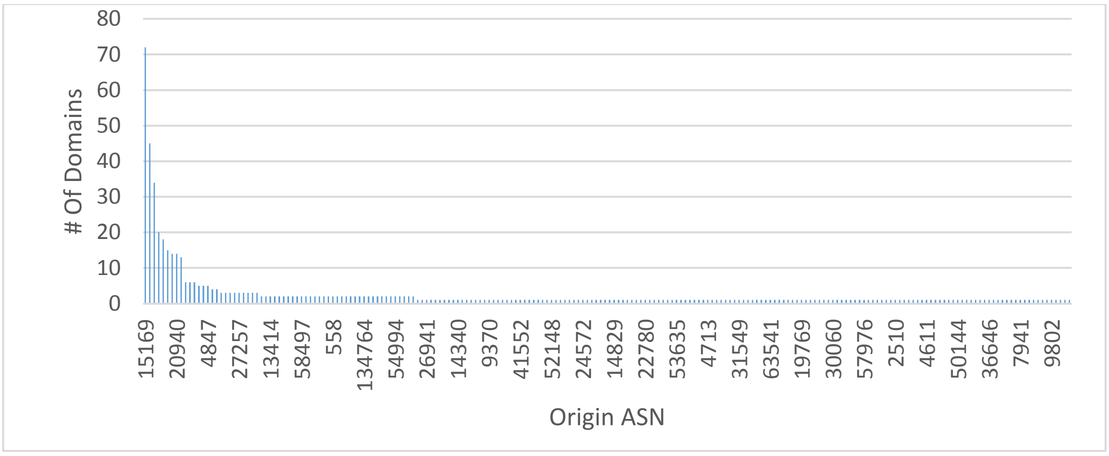

Figure 4: Time to First Byte for each CDN provider

Origin ASN	#Owner	#Domains
15169	Google	72
13335	CloudFlare	45
16509	Amazon	34
54113	Fastly	20
4808	China Unicom Beijing Province Network, CN	18
23724	China Telecommunications Corporation, CN	15
14618	Amazon	14
20940	Akamai	14
37963	Alibaba	13
15133	EdgeCast	6

Table 3: TOP 10 Origin AS Number

[3] http://bgpstream.caida.org/

Article Analysis

A Guide to DNS Record Type

https://blog.thousandeyes.com/guide-to-dns-record-types/

This recent technical article speaks about the different DNS records available. We found this article interesting for different reasons. Firstly, the article gives an overview of what is the purpose of the common DNS records, and what is the purpose of each one. In addition, the author wrote the text without dig too much in detail giving a simple and clear overview of the records, and how to get them, making the article easy to follow. The final reason is that the author gave a clear explanation about the CNAME field and how it one of its application nowadays. In particular, I refer about the usage of the CNAME record to provide an alias to a CDN infrastructure. This last aspect on of the key of our CDN tool.

To conclude we think that this article is interesting most for its content and the utility to support our choice while developing our tool and because it gave general and clear overview of some DNS records.

Pokémon GO Experiences Network and Application Outages

https://blog.thousandeyes.com/pokemon-go-network-application-outages/

The authors of this article analyzed two important outages of the famous Pokémon GO application.
This application released during 2016 quickly became very popular with millions of users worldwide located. For this reason study these outages is very important. First because it is important to understand why the outages happened, secondly because each one may affect millions of customers. The authors employed different applications to achieve their goal, in particular Charles proxy and ThousandEyes HTTP server Test to monitor the application infrastructure. By using these tools and monitoring Pokémon GO infrastructure from different points, they discovered that this application is centralized in a single place, located in the US. This first finding is very important, since it shows that in case of failure of this node, Pokémon GO will not be able to provide any service. A second interesting aspect is how smart hackers exploited this single point of failure. Indeed, during the first outage a group of hackers performed a DDoS attack while Pokémon GO was released in 26 European country; compromising the European customers’ trust. This aspect is very important as it show how nowadays redundancy in location has to be mandatory goal in order to deploy a robust infrastructure. The authors further prove this statement by analyzing the second outages caused by a problem with an application update.

To wrap up this article is interesting because it highlights how to study the infrastructure behind Pokémon GO and the reasons behind these outages.

Reference

[1] http://ieeexplore.ieee.org/document/7039137/metrics
[2] http://www.cloudvps.com/customer-service/knowledge-base/how-does-a-cdn-work/
[3] https://blog.thousandeyes.com/guide-to-dns-record-types/
[4] https://github.com/WPO-Foundation/webpagetest/blob/master/agent/wpthook/cdn.h
[5] http://bgpstream.caida.org/