FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Performance impact of the EDNS Client Subnet Extension
Student:	Vojtěch Šletr
Supervisor:	Ing. Martin Kolárik
Study program:	Informatics
Branch / specialization:	Computer Systems and Virtualization 2021
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2024/2025

Instructions

EDNS Client Subnet (ECS) is a DNS extension intended to improve DNS-based load balancing commonly used by content delivery networks and other global services. The goal of this work is to evaluate its real-world impact on the performance of existing web services.

Proceed in the following steps:
1. Familiarize yourself with the principles of ECS and the current state of its deployment in existing public DNS resolvers, and with the Globalping platform (https://globalping.io).
2. Propose a suitable metric for assessing the performance of DNS-based load balancing and a methodology for evaluating the impact of ECS on the proposed metric.
3. Implement a CLI application that can be used to perform experimental measurements based on the proposed methodology using the Globalping platform.
4. After consultation with the supervisor, select suitable targets for experimental measurements. Perform the experiments and discuss the results.

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 8 December 2023 in Prague.

Bachelor's thesis

PERFORMANCE IMPACT OF THE EDNS CLIENT SUBNET EXTENSION

Vojtěch Šletr

Faculty of Information Technology
Department of Computer Systems
Supervisor: Ing. Martin Kolárik
May 16, 2024

Citation of this thesis: Šletr Vojtěch. *Performance impact of the EDNS Client Subnet Extension.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Contents

List of Figures

List of Tables

List of code listings

I would like to thank my supervisor Ing. Martin Kolárik for sharing his experiences in this field with me, his assistance with the decisions made, and his contributions to the Globalping service that serves as a backbone to the application developed in this thesis.

Declaration

In Kladno on May 16, 2024

Abstract

The thesis focuses on comparing the results returned from load balancing DNS servers supporting ECS and those without this extension. Firstly, it describes what the DNS extension named ECS is, shows the intended implementation, and compares it to the actual practiced state. Then it introduces the Globalping platform, its functions and methods of data collection.

After introducing the main technologies used, it suggests suitable metrics for the comparison of the returned queries by DNS with and without the use of ECS. We then propose a methodology to collect and evaluate these metrics. According to the methodology, we develop a CLI application to automate this process with the use of Globalping API. This application is then used to perform measurements on a variety of public DNS services and CDN providers of different network sizes. The evaluated data suggest a statistically significant improvement when using ECS for some of the tested services, but without correlation to the size of the DNS resolver or the size of the CDN service network.

Keywords ECS, Globalping, DNS, load balancing, CDN, CLI, Python

Abstrakt

Práce se zaobírá porovnáváním rozhodnutí o přerozdělování zátěže na DNS serverech při použití ECS a naopak bez tohoto rozšíření. Zprvu proběhne seznámení se s DNS rozšířením nazývaným ECS, jeho doporučovanou implementací a porovnání s jeho aktuální podporou v praxi. Následně popíše službu Globalping, jí podporované funkce a způsoby sběru dat z jejích výstupů.

Po seznámení se s hlavními používanými technologiemi popíše možné metriky, které lze využít pro porovnání navrácených výsledků. Pro tyto metriky navrhne metodiku a implementujeme příslušnou CLI aplikaci určenou ke sběru dat dle navržené metodiky. Ta používá primárně služby Globalping ke sběru definovaných dat a následně tato data zpracovává pro vyhodnocení a případnou vizualizaci. Tato aplikace je následně použita na sběr ukázkových dat z testovaných CDN služeb za využití rozdílných DNS resolverů s různými počty serverů. Analyzovaná data naznačují statisticky významné zlepšení pozorovaných metrik při použití ECS pro některé z testovaných služeb, ovšem bez návaznosti na počet serverů CDN nebo DNS resolveru.

Klíčová slova ECS, Globalping, DNS, přerozdělování zátěže, CDN, CLI, Python

List of abbreviations

API	Application Programming Interface
CDN	Content Delivery Network
CLI	Command Line Interface
DNS	Domain Name System
ECS	EDNS Subnet Client
EDNS	Extension Mechanism for DNS
PoP	Point of Presence
REST	Representational State Transfer

Introduction

Since the rise of content delivery networks, there has been a need for a precise method to allocate servers near the client. Servers must be capable of providing content, but must also be close enough for efficient delivery. Today's DNS servers can generate dynamic responses based on the status of the servers inside a given cluster using load balancing features. But because the delivery time matters, the load balancing isn't based only on the load. For example, we should take into account the state of a network or the relative location of the server and the client.

In our ever-growing networks, the ability to approximate the client location only by the last known source address can be impaired. The DNS resolvers that have been assigned to us can be assigned to large areas and can deform our geographical or statistical estimations used for server assignment. Thats why ECS was created and implemented, it is a DNS extension that attaches a client subnet to the DNS query. This method primarily raised questions about the clients privacy, because techniques of approximating location by IP are used on a daily basis. But it is not the only rebuke, if the responses should differ based on the client subnet, it can affect the DNS server's caching abilities due to a larger amount of records. Because of this, the promised performance improvements come into question. Are the responses that were generated with added ECS performing significantly better? Are privacy concerns and caching challenges worth the performance improvement?

In this thesis, we will try to answer these questions. This question is mostly important to CDN providers, their goal is to globally deliver the content. They dispose of many data centers, and the task of assigning the server to the client is a crucial step for them. This task of assigning the server to the client is often delegated to a DNS resolver. But it is not the only possible solution. Some CDNs prefer load balancing through anycast network. As mentioned above, the DNS resolver can benefit from additional data about the original client, which could be lost during DNS query propagation. ECS provides to all resolvers that support ECS a subnet containing the client IP.

Since the implementation of ECS presents some challenges and compromises, not every DNS resolver supports the extension. We will describe related metrics that can get affected and present possibilities of their collection. One of the methodologies is implemented, set of CDN services to be tested will be prepared, and the collected data will be used to compare the differences caused by the use of ECS for these services.

Thesis goals

The main goal of this bachelor thesis is to describe the functionality of ECS and to measure the performance difference when it is used. We will define suitable metrics for measurement and propose how to collect and evaluate them. The thesis output is a CLI application designed to perform measurements and collect the data obtained. This application will be suitable for both occasional and periodic testing of the network state. Its main functionality will be linked to the Globalping API service. The application will parse the input and collect the data returned by the API. After data collection, the application will generate visualizations. This funcionality will be presented with exemplary input data as we discuss and evaluate the performance impact of ECS on the collected data with possible correlations to the sizes of tested services.

Chapter 1

EDNS Client Subnet

In the first chapters, we will look into the technologies around which our thesis will revolve. Now, let us describe the main technology, ECS. ECS is part of the DNS extensions family which provides additional information for better decision–making of the DNS resolvers. In this chapter, we will describe the Internet standard EDNS(0) and one of the extensions named ECS.

1.1 Extension Mechanisms for DNS

Extension mechanisms have been created and implemented as a reaction to the limits of the DNS protocol. As stated in the EDNS standard: "*The Domain Name System's wire protocol includes a number of fixed fields whose range has been or soon will be exhausted and does not allow requestors to advertise their capabilities to responders*" [1, Abstract]

```
+------------+--------------+------------------------------+
| Field Name | Field Type   | Description                  |
+------------+--------------+------------------------------+
| NAME       | domain name  | MUST be 0 (root domain)      |
| TYPE       | u_int16_t    | OPT (41)                     |
| CLASS      | u_int16_t    | requestor's UDP payload size |
| TTL        | u_int32_t    | extended RCODE and flags     |
| RDLEN      | u_int16_t    | length of all RDATA          |
| RDATA      | octet stream | {attribute,value} pairs      |
+------------+--------------+------------------------------+
```

Figure 1.1 Structure of fixed part of the OPT RR according to RFC 6891 [1, Chapter 6: The OPT Pseudo-RR]

Additional information is added into the additional data section of a DNS packet in the form of an OPT (Options) record, which is referred to as a pseudo-RR (Resource Record). It contains a structure of fixed parts, as shown in Figure 1.1, and a variable part containing the option code, length in bytes, and its data. [1, Chapter 6: The OPT Pseudo-RR]

By default, the UDP, packet size is set to 512 bytes, and if the UDP cannot be used for transmission, TCP on port 53 is used as a fallback mechanism. When EDNS is used, the size of DNS packets can often exceed 512 bytes, so a reconfiguration is needed to accommodate larger packets. This happens mainly when DNSSEC is used, which asynchronously ciphers the content. [2]

1.2 Intended implementation of ECS

The EDNS Subnet Client extension has not yet been standardized and is only documented for informational purposes, as stated in the RFC 7871 abstract [3]. This extension tries to solve the performance issues that arise with the spread of centralized resolvers. Their location can vastly vary from the original location of the client, and because of that, authoritative nameservers can create different, less precise responses. ECS in addition to DNS requests provides a scope of the originating network that can, but does not have to be used for creating a DNS response.

But it helps with more than just delivery performance. Geographical data can be used also for policy compliance, for instance the cloud security company Zscaler sees also benefits in using ECS values to identify the clients more accurately and enforce policies according to it or check the compliance with local regulations. This additional knowledge can help us with decision making, but raises questions about the security of client sensitive information [4].

```
                +0 (MSB)                            +1 (LSB)
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
   0: |                          OPTION-CODE                          |
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
   2: |                         OPTION-LENGTH                         |
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
   4: |                            FAMILY                             |
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
   6: |     SOURCE PREFIX-LENGTH      |     SCOPE PREFIX-LENGTH       |
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
   8: |                           ADDRESS...                          /
      +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 1.2 ECS extension format in DNS packet according to RFC 7871[3, Chapter 6: Option Format]

The format of the ECS option is stored, as mentioned in the definition of EDNS, in the additional data section of the DNS packet. The structure of the variable part of OPT RR is described in 1.2. The option code for ECS is 0x0008, family represents the address family code assigned by IANA. For us, the most important family codes are number 1: IPv4 and number 2: IPv6. The source prefix–length marks a usable prefix for lookup and is mirrored from query to response. The scope prefix–length is set to 0 in the query, and in the response it represents the prefix that was used for generating the response. The sent address must be truncated according to the source prefix and replaced by zeros; otherwise, the query should be rejected. [3, Chapter 6: Option Format]

The extension can be attached by the first resolver closest to the client, in most cases, it is represented by a stub resolver, but it doesn't have to be. It can be added along the way on a resolver that serves larger areas and where accuracy could be affected. Of course, for the successful usage of this information, every topologically succeeding resolver until the authoritative nameserver has to be ECS compatible.

"*If the triggering query included an ECS option itself, it must be examined for its source prefix-length. The Recursive Resolver's outgoing query must then set source prefix-length to the shorter of the incoming query's source prefix-length or the server's maximum cacheable prefix length.*" [3, Subsection 7.1.1: Recursive Resolvers] This means that the source prefix-length set to 0 effectively disables the extension. If any of the succeeding resolvers doesn't support ECS or DNS extensions at all, the additional data can be ignored and lost.

As documented in RFC 7871, when the authoritative nameserver receives a query with injected ECS, but the nameserver doesn't accept ECS, the extension should be ignored and the response shouldn't contain the ECS data. Some nameservers can blindly copy it from a query into a response. Because the scope prefix has to be set to 0 in the query, the response is still valid – prefix 0 means it applies to every address.

If the nameserver accepts ECS, it can use its provided information and set the scope prefix accordingly for correct caching. The response is propagated through the intermediate nameservers to the query author. The response gets propagated with additional ECS information only to the point where the ECS was injected. When caching, the address with the shortest prefixes of the source, scope, and locally configured maximal prefix gets stored. [3, Section 7.2: Generating a Response]

Even in the RFC 7871 [3, Subsection 7.3.1: Caching the Response], they mention the impact on the cache when ECS is supported. When intermediate nameserver supports ECS, the cache is filled up by entries duplicated for different subnets, and because of that, the number of unique entries cached is decreased, and the load on the server has to increase.

1.3 ECS usage in practice

Implementing ECS into a resolver's topology can impose many difficulties. From the impact on the cache performance to security obligations. For example, NS1 has seen a significant increase in queries when ECS is enabled: "*Typically, for a non-ECS-enabled record, 2-3% of queries come from Google and OpenDNS. For an ECS-enabled record, that can increase to between 10-50% of queries depending on the user base. In some extreme cases, NS1 has seen traffic inflate up to 2-3 times upon enabling ECS.*" [5] This corresponds to findings published at the Internet Measuring Conference in 2019 about cache–hit ratio drop: "*The results show a drop in hit rate due to ECS by more than half, for all client populations. For the full client population, the hit rate declines from around 76% to around 30%*" [6]

These results show that we must be careful about the prefix length that we are storing. The prefix length is important not only because of the cache performance, but also because of the security aspect. In RFC 7871 it is recommended to use a 24 or 20 bit prefix: "*To protect user's privacy, Recursive Resolvers are strongly encouraged to conceal part of the user's IP address by truncating IPv4 addresses to 24 bits. 56 bits are recommended for IPv6, based on [RFC6177].*

ISPs should have more detailed knowledge of their own networks. That is, they might know that all 24-bit prefixes in a /20 are in the same area. In those cases, for optimal cache utilization and improved privacy, the ISP's Recursive Resolver should truncate IP addresses in this /20 to just 20 bits, instead of 24 as recommended above." [3, Section 11.1: Privacy]

Table 1.1 Public DNS servers and their ECS support

Service name	ECS support	IP of controlled primary DNS resolver
AdGuard DNS	yes [7]	94.140.14.14
CleanBrowsing	no	185.228.168.168
Cloudflare	no [8]	1.1.1.1
Comodo Secure DNS	no	8.26.56.26
Control D	no	76.76.2.0
Dyn	yes[9]	216.146.36.36
Google	yes[10]	8.8.8.8
OpenDNS	yes[11]	208.67.222.222
Quad9	yes[12]	9.9.9.11
Yandex DNS	no	77.88.8.88

This shows that the integration of ECS into a standard isn't straightforward. Many public DNS recursive resolvers already support it, but we can still find nonnegligible outliers that slow down its full integration, as can be seen in the table 1.1.

Some of these DNS providers don't disclose their ECS support status. In those cases, we collected data from the *dig* utility. When the *subnet* option is used on the dig command, it specifies the ECS option in the DNS request, and we can analyze the responses to determine if the resolver utilized it.

1.4 ECS alternatives

The ECS isn't the only viable option for CDNs. Another popular method of effective load balancing can be achieved through anycast routing. Anycast-based CDNs delegate the load balancing to the routers. Their data centers share a common IP address, so the DNS resolver response is always the same. When a request is sent to the given IP, routers have to assign the server. This presents benefits as well as drawbacks. The load balancing can be done more dynamically – with every request, the assigned server can be changed according to the actual state of the network. That serves as an effective mitigation strategy for DDoS attacks. It is easier to dynamically change the active servers. When a server is added/removed, it just needs to be propagated to the routers, and no other action is needed. But, as mentioned, the network has to be adapted for anycast traffic. Routers have to be able to advertise the anycast groups, the routers have to monitor the state of the network to make load balancing decisions or, for example, there must be policies in place that supervise geopolitical compliance. [13]

Chapter 2

The Globalping service

There are multiple ways to test the state and quality of the network. In most cases, we can use many local-run techniques for basic network testing. But in some instances of worldwide accessible services, such as content delivery networks, tests have a higher informational value, when performed at different locations. These CDNs could potentially benefit the most from enabling the ECS option, but testing should become even more intensive to make the right load-balancing decisions. Since renting multiple server locations can become quite expensive for larger business models, there are services such as Globalping or Ripe Atlas that provide this feature.

Globalping is an open-source project developed by the jsDelivr organization and provides multiple testing features for data collection from many running probes around the world. Probes are hosted on servers and stations and can be expanded by anyone who runs a container provided by the developers. This chapter is based on the information provided by Globalping [14] and jsDelivr [15].

Ripe Atlas has more than 12 thousand testing probes around the world and, according to their website, they aim to be the largest testing network. But unlike Globalping, users cannot create their own tests above their infrastructure without participating in the project. They can become a sponsor or become a probe practitioner to gain credits. After that, they can create and run their own testing scenarios. [16]

2.1 JsDelivr

The jsDelivr organization's main product is a free CDN for open source files. With the main focus on fail-proof solutions, load balancing, and accessibility, the project is accessible without bandwidth limits for free, and the CDN loads are distributed along the sponsor's servers.

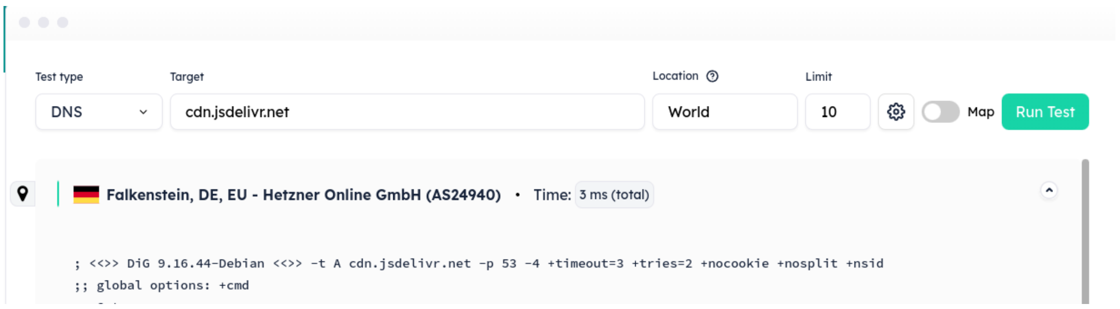
Currently, jsDelivr CDN is a multi-CDN that uses CDN networks provided by Cloudflare, Fastly, Bunny, and GCore, and uses 2 DNS providers with load balancing capabilities to evade the risk of a single point of failure.

It’s not a coincidence, that they decided to develop Globalping. As a CDN provider, they benefit from a worldwide testing tool for their network state. By allowing others to access this testing service and giving away benefits for maintaining their own probes, they gain a solid testing ground to evaluate the state of their network.

2.2 Available testing tools

To this day, five possible testing tools are provided. In this section, we look at them in more detail. Every tool can be targeted at a chosen URL/IP and the tester can specify a set of locations where to test from. If the tester wants randomly chosen locations, he leaves the location on “World” and then specifies the number of tests to run. According to the Globalping website, the location specification can be quite heterogeneous. Thanks to what is called the “magic API field”, they are able to parse many different location names of different granularity. For example, we can specify location from default *World* value, through continents, countries, to ASNs or ISP names. We can even specify datacenter tags or cloud region names. With a valid specification of the location, they will try to find the most suitable and active probes in the amount asked. If not enough probes fit the specification, the number of tests will be set to the available number of probes.

2.2.1 DNS


Test type: DNS
Target: cdn.jsdelivr.net
Location: World
Limit: 10

Falkenstein, DE, EU - Hetzner Online GmbH (AS24940) • Time: 3 ms (total)

```
; <<>> DiG 9.16.44-Debian <<>> -t A cdn.jsdelivr.net -p 53 -4 +timeout=3 +tries=2 +nocookie +nosplit +nsid
;; global options: +cmd
```

Figure 2.1 Exemplary usage of DNS test with default settings implemented above dig [14]

DNS testing is performed on probes using the *dig* command for DNS lookup. With default settings in use, the dig runs with some predefined arguments as shown in figure 2.1. In addition, we can set the port, protocol, record type, IP address of the desired resolver, and usage of the trace option for iterative queries.

2.2.2 My Traceroute (MTR)

MTR is a special combination of network tools used to test the network. “*My Traceroute (MTR) is a tool that combines traceroute and ping, which is another common method for testing network connectivity and speed. In addition to the hops along the network path, MTR constantly shows up-to-date information on the latency and packet loss along the route to the destination. This helps in troubleshooting network issues by allowing you to* *see what's happening along the path in real-time.*" [17] We can use options to specify the transport protocol (TCP, UDP, ICMP), with ICMP selected as a default for pinging, the number of test packets for each hop, and the port. Of course, Globalping provides both ping and traceroute as separate testing features, and these results could be replicated manually.

2.2.3 HTTP

For testing, we can also use the HTTP request and monitor the response metrics. Except for the received header, we gain time measurements about DNS query, TCP connection, time to first byte (TTFB) and download time of the contents. The request can be modified with a specified header consisting of hostname, path, query string, targeted port, protocol (HTTP, HTTP2, HTTPS), and method used (GET, HEAD). We can also select a specific resolver. This method of testing with consistent options can be periodically run for data collection and subsequently monitoring the state of the network.

2.3 Integration

The goal of the Globalping project is to provide easy testing tools so that they can be automatically deployed. This presents a challenge in creating many possible integrations to reach a larger audience. For one-time testing, the easiest is possibly the integration into their website. Although data can be collected directly from the website, it would be time-consuming and resource-consuming. The main computer-friendly integration is provided by the REST API, and the other integrations provided by Globalping are based on it as well. The API doesn't require authentication and the request–response model is JSON-encoded with standardized HTTP response codes.

In addition to API, we can use the CLI implementation for Debian / Red Hat-based operating systems or MacOS with support for all testing methods, as it is ideal for scripting. For better documentation of the tests and their shareability, it can generate an url for the recorded values.

We can also add Globalping functionality to our Slack chat, and it can be run directly by users or as a GitHub Bot. According to the Globalping website, more versions are to come. For example, integration for ChatGPT, Zapier, or GitHub Action. [18]

Chapter 3

Metrics for analysis

In this chapter, we will focus on accessible metrics for further analysis. We will describe some of the possibilities that can be periodically measured and compared against each other. Our main focus is on the server assigned by the DNS query response. The additional load generated by the ECS option on DNS resolvers and its effect on the DNS resolver response time are secondary to us. But even with this in mind, there are multiple ways to compare the responses.

3.1 Round-trip time

Even though we are not focused on the response time of the DNS resolvers, we can demand low response times and fast delivery of content from the assigned servers. One of the easiest scenarios for measuring these capabilities can be comparing round-trip times of packets between the client and the assigned server. These statistics provide important information on the state of the network. These metrics can be primary if the load is frequent, time-sensitive, and relatively small. With larger packets, this importance can decrease, even though it always depends on our ability to receive and process incoming data.

3.2 Geographical distance

Another way of comparing the assigned servers is to estimate the geological position and measure the distance from our client. Even though this type of measurement doesn't automatically relate to performance metrics, it can help us with geological policy enforcement and can give us a insight into the load balancing techniques used by the resolver.

The geographical distance can be estimated on its own using multiple techniques. One of the most used techniques is database geolocation. This database can map IPs to countries, cities, ISPs, and other similar locations.

However, the database has to be well maintained with a sufficient number of entries to be precise. The next option can be String matching geolocation that relies on information accessible from DNS names. We can track location by having many known nodes and finding location using the shortest RTT or topological order and other algorithms. [19]

3.3 Content delivery time

To complete our data collection, it is important to test the network throughput or the delivery time of the content. For periodic testing, we can prepare static data that correspond with our typical data delivery requests, or create multiple sets of these files and compare the download times from assigned servers. For example, this test can be performed as an HTTP request with time-measuring functions, as mentioned in Section 2.2.3.

For optimal testing, this measurement requires us to state our priorities and to research the most frequent file sizes that our service delivers. We should periodically inspect those decisions and change the tested files accordingly.

Chapter 4

Methodology proposal

We have already set the metrics of interest in the previous chapter. Now we can consider metrics collection techniques and propose a methodology for their evaluation. As stated before, we focus on the performance benefits of ECS for CDN providers. Because of that, the tests should be run globally and evaluated together.

Our goal is to decide whether the performance of content delivery is improved by using ECS. In addition, we want to test whether the performance impact is affected by the size of the CDN network or the DNS resolver. We expect that the change in network sizes could impact the ratio of distinctively and equally assigned CDN servers, and with this change in ratio, we could see the ECS perform differently. To test this, we will select the CDN services and resolvers to cover different company sizes and permutate these combinations.

4.1 Location of the tests

To run the tests from various locations around the world, we have to obtain access to the servers or end-user devices at those locations that are capable of running our tests. It could be done by renting many hosted servers or even buying and maintaining them. But it is very expensive and not nearly as flexible as our other option. In Chapter 2, we have described the Globalping service that provides us exactly with these features to run tests from hundreds of possible probes and, in this instance, even without any costs to run these tests. Globalping isn't the only testing service available, similarly, we mentioned RIPE Atlas with even more probes, but RIPE doesn't provide free testing utilities without participating on the project. Thats why we will propose the tests with the Globalping abilities in mind.

4.2 Targets of testing

As we are trying to find a relation between CDN size and the performance impact, we will choose our targets based on their availability to the public and the number of PoPs (points of presence) provided. We need to filter out CDNs that use anycasted routing, because then the DNS responses would be the same as they assign the server by routing in the moment of data request instead of DNS request. We won't also test private CDNs as they serve only content of a specific parent company and they can't be used by other customers.

4.3 IP assignment

As we are testing the ECS capabilities, we are coming to a bottleneck in our methodology proposal. For every location tested, we need a static resolver that will serve both our DNS requests — with and without the ECS. The selected testing utilities don't provide us with the option to set the subnet prefix provided by ECS to 0 length. Without setting the prefix to 0, we cannot ensure that an intermediate resolver will not add the ECS along the way. There is a special public DNS resolver Quad9 [20], which allows control of the ECS, but it would limit our choices and the results of our measurement could not be easily convertable to other DNS services.

With these limitations in mind, we have decided to perform the measurements using only DNS servers that do not support ECS. To simulate the ECS functionality, we will use the DNS trace option. When the trace flag is set, the command iteratively propagates the DNS query through the resolution hierarchy by sending the requests directly from the client to every intermediate resolver up to the authorative nameserver. This means that the nameserver has direct access to the client IP address. This will serve as a best-case scenario for the ECS function, because ECS should not provide a prefix longer than 24 bits [3, Section 11.1: Privacy] under normal circumstances and with the trace option, we will provide all 32 bits of our IPv4 address to the nameserver.

4.4 Resolvers

As described in previous section, we will be using resolvers that don't support ECS. Because of that we won't rely on default recursive resolvers assigned to the probes and we will be selecting public resolvers. We already listed a few of them in the section *ECS usage in practice* in the first chapter. We will consider the number of their servers, as it could impact the resolution.

4.5 Measurements

After we have selected our targets, the locations from which to run the tests and which resolvers to use, we can approach the measurements themselves. Let us say that we have been assigned with IP addresses of the service servers for every location and every service that we chose. With the first step completed, we can start collecting data about our defined metrics.

In some cases, the IP assigned by a non-traced DNS query will match the traced DNS query. We could run the test 2 times, but the differences coming from those results wouldn't reflect the change in DNS decision but only a instability of the network or change in cache. We don't consider these metrics relevant and because of that, the measurement will be conducted only once. We could even ignore locations with same assigned IPs, but since we are creating a global average of the difference, we don't want to exclude them as they are part of the DNS behavior.

4.5.1 Round-trip time

With already assigned IPs, the response time is fairly straightforward. We can use ICMP ping to measure it and run it multiple times just to get more representative average values. But average values are not everything, we can make use of its extremes for informational purposes. It is also a good idea to account for occasional packet loss by implementing outlier detection or to analyze the median of response times instead of mean.

4.5.2 Content delivery time

The measurement of the delivery time is not as straightforward as measuring response time. Before performing these tests, we need to set a representative response size for the tested service. Since we are conducting a general test of universal services, we can simulate a content delivery for a web page.

If we look into the last report on page weight by Web Almanac from the year 2022 [21], the websites are steadily growing each year and in 2022, the 50th percentile of the median total web weight was around 2MB. This isn't served as a single request, but is split into higher tenths or requests. The loaded images contribute to the page weight in about 1MB per page, and Javascript files can reach about 500kB. Even with compression and minification techniques, web weights grow consistently every year.

With these statistics in mind, we can prepare multiple files with different sizes and test their delivery times against each other. Since we already have the server's IP assigned, we can focus just on the delivery.

The most interesting metrics for us are the time until the first byte is received and the download time itself. To ensure the consistency of the tests, we need to run these tests multiple times, so that the CDN can cache the response at the nearest available location. This will help our further analysis as we don't have to consider the probabilities of a cache miss.

4.5.3 Geographical distance

To measure the approximated distance of the client from the server, one needs to obtain the longitude and latitude of both of them. Since we chose Globalping testing probes, we obtain its longitude and latitude data in every successful measurement. The location of the CDN server would then have to be obtained from a third-party geolocation database, and their precision could affect our measurements. Because of this, we will not gather this proposed metric.

4.6 Data evaluation

After collecting the data, we should have accessible data for every location in the permutation of the targeted service, the used resolver, and the downloaded file. This is an ideal premise, and during the real measurements, some locations will be lost as the test will be run continuously for hours. We can then group them into categories by the number of CDN PoPs and the number of DNS servers for each tested file. Then we can look if any of these groups performed better when the IP was assigned by trace DNS query and compare the groups between each other if the effects are canceling each other out with different number of servers or vice versa.

To confirm our observations, we can use Paired T-Test on the median response times and the download times. The differences between the measurements on the assigned servers should follow a normal distribution and will be independent between locations and services. By choosing the null and alternative hypotheses based on the mean of difference, we will be able to pick out subjects that will not fit into the set significance level and we will reject the null hypothesis in favor of the alternative hypothesis.

This evaluation can help us understand if the performance impact can become negligible with enough servers available by DNS and/or CDN service, or if the performance impact overcomes the stated privacy concerns and challenges with higher loads on DNS cache.

Chapter 5

Application for data collection

Since we have proposed a method that we could use for the ECS performance testing, we can develop an application, designated for data collection. In this thesis, we aim to develop a CLI application that will be able to collect data for metrics described in the previous chapter Metrics for analysis. Because our application is meant to test mostly CDN services that rely on DNS load balancing, we want to perform our tests from multiple locations. Because of that, we will be using testing utilities provided by Globalping. As mentioned in the chapter Globalping service, they dispose with hundreds of testing probes around the world, which can be accesed through their RESTful API. Eventhough the API doesn't require authetication, without it, there are hourly limits in place that we will have to respect.

For easier data collection, input processing, and mainly data aggregation, we chose to develop the application in Python 3.10. Python provides us with modules that ensure standardized CLI, API utilities, plenty of possibilities to store and aggregate the collected data, and also gives us solid ground for additional features like visualization of the data.

```
python3 ecs_collector -h
usage: ecs_collector [-h] {location,service,run,visualize} ...

ECS Collector is a CLI app designed for measuring differences returned by DNS
resolvers when EDNS Client Subnet is in use and when it is not.

options:
  -h, --help            show this help message and exit

Supported functions:
  {location,service,run,visualize}
    location            Manage the location sets from where the test should be
                        run
    service             Manage the service sets for which the test should be
                        run.
    run                 Run the tests.
    visualize           Run visualization. If no path is set, tries to load
                        last stored file from visualize/data.csv
```

Figure 5.1 Generated help-page for the first level of commands

5.1 CLI application

Since we chose Python, we could standardize the interface of our application with existing modules. After testing multiple of them, we build our interface on top of the Argparse module[1]. This provides us with support for nested commands, many types of options and arguments, and automatically generated help-pages.

As the commands are nested, firstly the commands are split between test locations management, management of the tested services, running the ping and http tests, and lastly running the visualizations. These commands are documented for help-page generation, which respects the command nesting as can be seen in figure 5.1

These commands continue to be divided by subcommands and other options similarly to the main run command in Figure 5.2. This command handles the main task of running the tests and therefore contains many possible options and flags to change the program behavior. From input specifications, suppressing error messages and enabling the visualization to the data export.

```
python3 ecs_collector run -h
usage: ecs_collector run [-h] [-i | -l] [-r RESOLVER] [-c [0-8]] [-e EXPORT] [-t [1-1024]]
                         [-k KEY] [-d] [-m] [-v] [-s] [-f [FILES ...]]
                         {ping,http} [{ping,http} ...] locationset serviceset

positional arguments:
  {ping,http}           Choose what tests to run.
  locationset           Location set name, path to CSV file when -l flag is used or id of
                        previous measurement when -i flag is used
  serviceset            Service set name or path to CSV file when -s flag is used

options:
  -h, --help            show this help message and exit
  -i, --id              Enter id of previous measurement instead of locationset
  -l, --locationImport  Import location set from CSV file consisting of columns named
                        "Location","Count"
  -r RESOLVER, --resolver RESOLVER
                        Resolver address
  -c [0-8], --cache [0-8]
                        Amount of rounds of cache warmup before http test is counted.
  -e EXPORT, --export EXPORT
                        Export the results into a CSV file
  -t [1-1024], --threads [1-1024]
                        Number of threads to be used, default=16
  -k KEY, --key KEY     Use key to unlock Globalping API limits.
  -d, --daemonize       Don't exit the script when Globalping API limit is reached and
                        wait till its reset. The limits reset hourly.
  -m, --mute            Supress warnings for skipped tests
  -v, --visualize       Visualize the results using Streamlit
  -s, --serviceImport   Import service set from CSV file consisting of columns named
                        "ServiceName","ServiceURL", "DownloadPath". DownloadPath is not
                        required for ping test
  -f [FILES ...], --files [FILES ...]
                        List of names to be appended to the DownloadPath for testing of
                        multiple files at once.
```

Figure 5.2 Generated help-page for the run subcommand

[1] `https://docs.python.org/3/library/argparse.html`

5.2 Inputs

The main data expected at the input are the locations from which to run the tests and the services that should be tested from them. Those inputs can be imported from external CSV files, or they can be stored internally in named sets with multiple ways of managing these sets.

For locations, our application doesn't parse the location names from input and we are handing them over to the Globalping API that provides special "magic" fields for location selection. This funcionality has been already described in the chapter Globalping. Thanks to this functionality, we only store the location specification and the number of probes that should be used in this location. When there are not enough test probes corresponding to the location specification, the number of tests is reduced to the number of corresponding probes. The number of locations is limited by the API and is set to 200 for every location up to 500 in total, or one location with up to 500 measurements.

For services, we have to store more information. Mainly we need the service URL, for more lucid data export, we optionally collect user-defined name for the service, and if the http test should be run, we collect additional URL from where a test object should be downloaded for advanced performance measurements. But if the user doesn't include them, the application generates both values from the service URL. If multiple files should be tested, the user can set only the base URL and append file names with option -f when starting the test.

Both sets can be saved internally with additions/deletions from the CLI, or they can be overwritten from an external CSV file. When the tests are run, they can accept more optional arguments like custom resolver for DNS requests, authentication keys to increase the API's limits, or IDs of previous measurements to reuse the test locations.

Both of these subcommands delegate file operation to functions available from *file_manager.py*. In this file, the adresses are parsed, and with the use of *pathlib* module[2], file operations are performed, including exception management. This can be seen in the code listing 5.1 where the line should be added to the internally stored set of locations or services. The function first generates an absolute path to the expected file to mitigate the path of the working directory from which the app was run. Then it checks if the file exists and if it is a writeable path. If the file or its parent directory does not exist, it will try to create them and write the initial line of CSV with column names. Only then is the line written, and possible exceptions are treated.

Other functions handle the files with a similar level of caution for errors. The File Manager takes care of file writing, deleting files and lines from them, loading Pandas Dataframes from CSV files, or listing stored files.

[2]https://docs.python.org/3/library/pathlib.html

Code listing 5.1 Function append_file from file_manager.py handling addition to internal files

```
def append_file(setname, line, location=False):
    filepath = pathlib.Path(local_addr(setname, location))
    dirpath = pathlib.Path(__file__).parent.absolute()
    dirpath = (dirpath / "location_sets"
               if location
               else dirpath / "service_sets")
    try:
        if not filepath.exists():
            if not dirpath.exists():
                dirpath.mkdir(parents=True)
            if not dirpath.is\_dir():
                print("Cannot create file, " +
                       dirpath.__str__() +
                      "is not a directory",
                       file=sys.stderr)
                return
            with open(filepath, "w+") as file:
                if location:
                    file.write("Location, Count\n")
                else:
                    file.write("ServiceName,ServiceURL, "
                                + "DownloadPath\n")
                file.write(line + "\n")
        elif filepath.is\_file():
            with open(filepath, "a") as file:
                file.write(line + "\n")
            return
        else:
            print("Cannot append file, " +
                  filepath.__str__() +
                   "is not a file", file=sys.stderr)
    except PermissionError as e:
        print("Permission denied", file=sys.stderr)
    except IOError as e:
        print("Unable to write into the file " +
              filepath.__str__(), file=sys.stderr)
```

5.3 Data collection

The app collects the data from the Globalping API. When input data are parsed, the app first uses DNS mode to receive the IPs of assigned servers. This is done twice for each location as we collect the IP assigned by the standard DNS request and also from the trace DNS request. The trace option ensures that the nameserver has our client's IP and simulates the best-case scenario for the ECS option. When the IPs are collected, we can continue with the selected test.

We must ensure that the locations do not change during our tests. Because Globalping probes don't have public IDs and we cannot choose them directly, Globalping returns ID of the measurement, which can later be used to run more tests from the same probes. We can use this instead of the separate probe ID and replace the location specification. When request ID is used, Globalping ensures that the same probes are used and results are returned in the same order. Although this solution seems flawless, it poses a challenge on the amount of the request sent. Since we have received the server's IP previously from the DNS request, we cannot use the URL as a target and we need to run the API request for every unique IP and location. This poses a challenge on the API limits and significantly slows down the data collection.

Following the guidelines of Globalping API [18], the application can request the results every 500 ms until measurements are collected. Because of that, there is a lot of waiting time that gives us an opportunity to parallelize the data collection. Firstly, we collectively query the DNS for each service. After getting the IPs assigned, we store each location as a set of country code, city name, and ASN. This can lead to losing some locations with the same attributes, but if we would query the DNS separately by every location, we would lose the automatical probe dispersion provided by the Globalping magic field.

After parsing the DNS queries, the following tests will be run for each service and location separately, once with IP assigned without the trace option and with the location defined by the country, city and ASN. Secondly, if IP assigned by the traced DNS differs, another test is run for the given IP, with the location specified using the ID of previous measurement.

All measurements are created, retrieved, and stored in *Pandas Dataframe* in functions defined in the file *api_interface.py* as they handle DNS queries and http or ping tests. The process of creating a measurement is done in a unified function *send_request* that sends POST requests using the module *Requests* [3] and handles possible response codes and exceptions that could arise, including the depletion of hourly API limits.

With the long waiting times for measurement data retrieval, parallelization is needed. For DNS queries, the worker manages both sending the request and retrieving the data, as it is required only twice per service. In follow-up testing scenarios, the measurement is created in the parent thread, and the worker threads serve only as consumers of the shared queue. The default number of running threads is set to 16 but we recommend running at least 64 as many of them will be waiting the defined 500ms. The user can change this value by using the -t option with values in the range of 1-1024.

If in any of the workers an exception arises that would make the continuing test unforessenable, the *thread_end* variable representing synchronized *threading.Event()* is set and the program is ended. In most cases, this action isn't needed as the failed measurement gets simply skipped. If all tasks are completed, the results get joined with existing dataframe and the collection ends.

[3]https://requests.readthedocs.io/en/latest/

5.4 Solution of Globalping API test limitations

Eventhough the service is accessible for free, there are limits to the number of tests to be performed per hour. To overcome this, the user can use the authentization key to gain higher limits or to be charged for the test in credits. When the hourly limit is depleted, API responses with code 429, the function *send_request* handles this behavior, as can be seen in the following code listing 5.2. The function reads the returned headers and they contain values about the limits and consumed credits if the authentication key was used. When the –daemonize flag is set, the application waits for the returned time and requests the results again. Otherwise, the application ends.

Code listing 5.2 Handling of depleted hourly API limits

```
if response.status_code == 429:
    till_reset = int(response
                     .headers['X-RateLimit-Reset'])
    if daemonize:
         time.sleep(till_reset)
         return send_request(data, headers, daemonize)
    else:
        raise requests.exceptions.RequestException(
        "Globalping API Limit Exceeded, wait " +
        str(till_reset) + " seconds and try again" +
        " or run the tests with -d flag. If you " +
        "have valid authentication key for Globalping"
        "API and sufficient credits, use parameter -k")
```

5.5 Data visualization

When running the tests, the flag –visualize can be set to display the data using *Streamlit* module[4]. Since the streamlit website app has to be started separately with *streamlit run [file.py]*, the application saves the generated data internally into the *visualize* folder and relaunches the application as can be seen in 5.3.

Code listing 5.3 Code to store data for visualization and relauch the application with Streamlit

```
if args.visualize:
    if data_for_visualization(results):
        print("To stop the web application, use Ctrl+C"+
              " or click the Close button on page.")
        sys.argv = ["streamlit", "run",
          pathlib.Path(__file__).parent.absolute()
                  .__str__() + "/visualizer.py"]
        sys.exit(stcli.main())
    else:
        print("Couldn't store data for visualization")
```

[4]https://streamlit.io/

The streamlit application opens up a loaded website in our browser and then can be terminated using Ctrl + C in the terminal, or we added an additional close button to its sidebar. The closing mechanism, seen in code listing 5.4, used by the button is also used when the application is unable to continue corectly like when the data at input couldn't be succesfully loaded. This process is based on a solution made by the Streamlit user Ahmadzam [22]

Code listing 5.4 Code for closing the Streamlit application

```
def close_app():
    time.sleep(0.5)
    # Terminate streamlit python process
    pid = os.getpid()
    p = psutil.Process(pid)
    p.terminate()
```

Another modification to the web page was to hide a *Deploy* button that is by default shown in the application header. This could be hidden with the Tony Kipkemboi code [23] with this modified block of code 5.5.

Code listing 5.5 Code for hiding the Streamlit deploy button

```
    st.markdown("""
        <style>
            .reportview-container {
                margin-top: -2em;
            }
            .stDeployButton {display:none;}
        </style>
    """, unsafe\_allow\_html=True)
```

When data are successfully loaded from the CSV file, the program checks for columns found and adds possible visualization modes accordingly. With selected visualizations, other control features can be loaded into the web sidebar. Those include filtering by a service, removing rows with the same IPs assigned by DNS query with and without the trace flag, filtering out the outliers or sampling the data, to make the graphs more lucid.

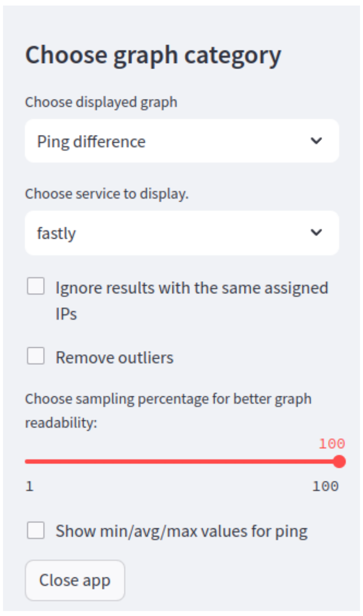
Outliers are filtered by the function *zscore* provided by *Scipy Stats* package. It calculates the mean of the values in the column and assign a z-value to them assuming the data follow a normal distribution and remove outliers with more than three times the deviation. [24]

5.5.1 Ratios of assigned IPs

First category of possible visualizations is the ratio of IPs assigned by DNS servers without trace flag that match the responses of the traced DNS. We can look into these numbers globally, per service and per location.

5.5.2 Response times

Another visualization is based on the time differences between the response times. The 16 collected pings are aggregated into the median and can be aggregated as a mean per service, or we can select a service and see the difference per location. In this detailed view, sampling comes handy so the location labels don't overlap each other. For ping, we offer the option to display the lowest and highest value along with the average. These controls are located in the sidebar, as can be seen in Figure 5.3


Choose graph category

Choose displayed graph: Ping difference

Choose service to display.: fastly

☐ Ignore results with the same assigned IPs

☐ Remove outliers

Choose sampling percentage for better graph readability: 100 (1 – 100)

☐ Show min/avg/max values for ping

Figure 5.3 Sidebar with controls for Ping difference visualization

5.5.3 Delivery times

Last set of visualizations uses data collected from the http test. There are two modes that display the difference between the total download times per service or per location and service, or there is a more detailed version that shows a dual-stacked bar graph that displays the actual time until the first byte was received and the download time afterward.

5.6 Application setup and requirements

The application comes with requirements.txt file through which the user can install all dependencies using the command *pip install -r requirements.txt*. After meeting the requirements, the application can be run with the command *python3 ecs_collector ...* The application comes with pregenerated data for visualization and comes with pre-filled location and service sets. To test the basic functions, the user can follow the available *ReadME.md* file or just use the -h flag that will show him the available subcommands.

Chapter 6

Performing the measurement

In this chapter we focus on input data selection, its parsing into the application, running the tests and generating data from them, exporting the data and lastly discussing the outcome.

6.1 Input data selection

As we mentioned in the methodology proposal, we will choose our input data with multiple criteria. For both public resolvers and CDN providers, we will try to find services with different numbers of servers. The CDNs will have to be publicly usable, and they have to use DNS to load balance their servers. The resolvers on the other hand can't support ECS to ensure the client's subnet won't be propagated to the nameserver.

6.1.1 CDNs for testing

We have chosen the services that met our criteria and were successful in storing our testfiles on their servers. The services and their number of locations can be seen in the table 6.1. Some other services were in play, but we were unable to get a free test account to be able to use them. To list some of them, we weren't able to get in contact with services like Akamai and CDNetworks. Some other known services come to our mind when selecting the services, like Microsoft Azure, Cloudflare, Google Cloud, Edgio, Wedos, Imperva, or GCore, but, based on our testing, they use anycast routing instead of DNS-based load balancing.

The data cached in these services were 3 files of different sizes with generated "lorem ipsum" content. The file sizes were 10KB, 100KB, and 1MB according to the methodology proposal. These files were uploaded to the services directly or a free Github-hosted page *ecstest1.github.io* was used to cache.

Table 6.1 Selected CDN providers

Service name	Number of servers / locations
Cloudfront	1200 PoPs in 300+ cities[25]
Bunny CDN	123 locations[26]
Fastly	78 locations[27]
KeyCDN	60 locations[28]
CDN77	54 locations[29]
Netlify	22 locations[30]

6.1.2 Public resolvers used

To test the difference that the size of public DNS can have on the assignment, we chose public resolvers listed in the table 6.2. This variance should provide us with enough information if the number of DNS locations affects the server assignment.

Table 6.2 Selected public DNS resolvers

Resolver name	Number of servers / locations	Resolver IP
Cloudflare	320 cities	1.1.1.1[31]
Quad9	220 locations	9.9.9.9[32]
Control D	137 locations	76.76.2.0[33]
CleanBrowsing	58 servers in 28 cities	94.140.14.14[34]
AdGuard	70 servers in 15 locations	208.67.222.222[35]

6.1.3 Locations of the tests

We didn't specify the locations for the testing and let Globalping choose them variably from the active pool of probes. We maximized the number of locations and ran a test with 500 locations specified as "World". We ran this request right before the data collection and obtained a measurement ID to reference this location list for every other succeeding test.

In addition, we slightly modified the application, as can be seen in the code listing 6.1, to record the returned locations for us, and the list of locations can be seen in the attachments.

Code listing 6.1 Modification of the app for collecting the locations

```
location_map = sim.StringIntMap()
print('index,country,city,asn')
for row in response['results']:
    if ('probe' in row and
        all(l in row['probe']
           for l in ['country', 'city', 'asn'])):
        print(str(location_map.string(
                      row["probe"]["country"],
                      row["probe"]["city"],
                      row["probe"]["asn"]))
             + "," + row["probe"]["country"]
             + "," + row["probe"]["city"]
             + "," + str(row["probe"]["asn"]))
        return location_map
```

6.2 Test outputs

After the measurement ID was obtained as mentioned in the previous section, we ran a script, which can be seen in the code listing 6.2, to collect and export data from the ping and http measurements for every mentioned service and resolver from the locations.

Code listing 6.2 Script for data collection

```
#!/bin/bash
declare -a res_ip=(
[0]=208.67.222.222
[1]=94.140.14.14
[2]=76.76.2.0
[3]=9.9.9.9
[4]=1.1.1.1
)
declare -a res_name=(
[0]=AdGuard
[1]=CleanBrowsing
[2]=ControlD
[3]=Quad9
[4]=Cloudflare
)
declare id="ThYo3LoyqqxcyBiL"
for i in {0..4}
do
time python3 ecs_collector run -i -c 1 -d -t 1024 \
 -e data/$(($i + 1))_${res_name[$i]}.csv \
 -r ${res_ip[$i]} ping http $id services \
-f 10 100 1000 2>&1 | tee -a \
 data/$(($i + 1))_${res_name[$i]}.log
```

This script runs the tests for every resolver with one round of cache warm-up to standardize the download times, uses the maximum of 1024 threads, exports the results into separate files and specifies the names of the 3 files to be downloaded.

In addition, it runs a *time* command to measure collection time and logs the output of the application into separate files. If the API limit should be reached, the -d flag is set to wait until reset of limits and continues in the measurements.

6.3 Exported data

The tests were completed successfully after 7 hours of continuous running. We have collected data from 437 locations, as some of them shared the same identifiers and some of them stopped responding during the testing phase. We haven't been able to collect response times from Netlify servers as it seems that the ICMP protocol for ping is blocked by their firewall.

For this exact collection, the testing time per resolver was, on average, 84 minutes, but when similar tests with the same measurement settings were run from another station, we were able to achieve times under 50 minutes. We suppose that the final times are dependent on the number of CPU cores and the abilities of the network interface.

All generated data can be found in attachments as they are five different CSV files consisting of around 7500 rows each. They are sorted per the service in the same order as on input, as multiple files were downloaded, the measurement rows are duplicated, and the file name is appended to them, the appended string is also stored in separate column for easier parsing later on. If a ping test was run, the measured data are represented only in the first of these duplicated rows. As 16 ping packets were sent to every assigned server, the response times are stored separately plus aggregated data such as mean, median, minimum, and maximum value. For the http test, the time to first byte, the download time, and their sum are stored. All these columns are duplicated and the prefix "t_" is used for the responses from the trace-assigned server. If same IP was resolved, the data are identical to the non-traced version.

6.4 Results discussion

After data collection, we have looked at the pregenerated visualizations by running a "*python3 ecs_collector visualize -p [file path]*", when we visually compared the results, we found that the ratio of identical and distinct assigned servers was consistent through our testing for every resolver, as the percentage of distinct IP addresses was around 65% as can be seen in Figure 6.1

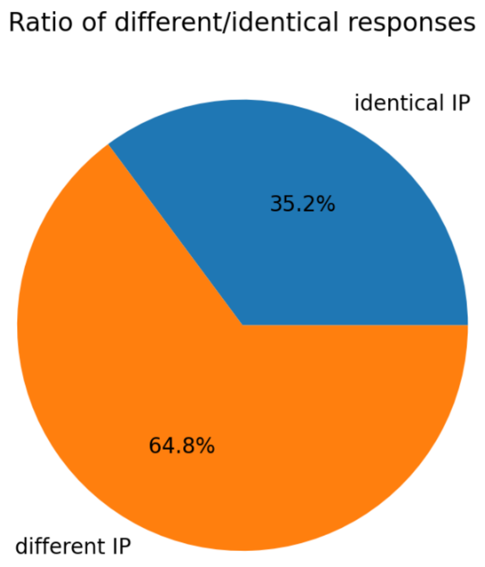


Figure 6.1 Ratio of the equal and distinct assigned addresses for AdGuard resolver

The ratios per service remained fairly similar throughout the tests as well, as some services responded differently more often than others, as can be seen in Figure 6.2

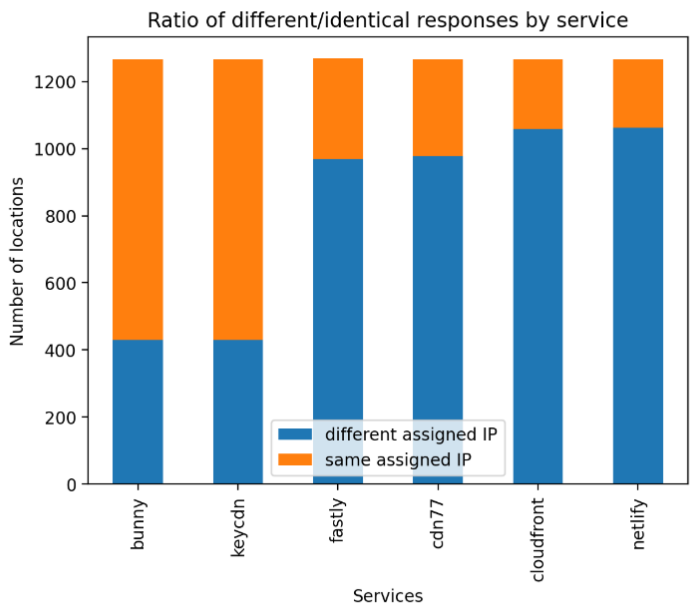


Figure 6.2 Ratio of the equal and distinct assigned addresses per service

We expected to see some differences between the used resolvers and we looked into the data itself. We have counted the total number of unique addresses, and later on we counted the addresses for each service also 6.3, and the data stays consistent throughout the testing.

Table 6.3 Number of unique IP addresses

Service name	AdGuard	CleanBrowsing	Control D	Quad9	Cloudflare
Cloudfront	257	262	247	247	249
Bunny CDN	120	119	122	120	117
Fastly	80	81	81	79	80
KeyCDN	45	46	45	46	45
CDN77	204	201	207	205	198
Netlify	28	28	28	28	28
total	734	737	730	725	717

The only outlier that didn't correspond with the expected number of servers was the CDN77, but after examining the IPs with the IPinfo tool [36], we were able to confirm that every location uses at least pair of IPs and often more, so the number of locations corresponds with our expectations in the end.

This data lead us to believe that, in our measurements, the resolvers in our tested range of tenths to lower hundreds of locations don't visibly affect the outcome of the response. After observing the responses from DNS, we arrive at evaluating the response times and download times of the services. When observed visually, we used primarily the absolute mean values per service next to each other, such as the ping measurement in Figure 6.3 or 6.4

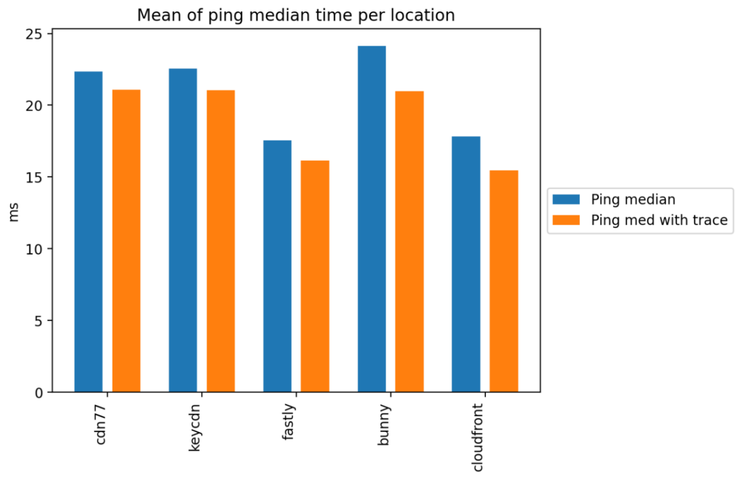


Figure 6.3 Mean of median response times per service with Control D resolver.

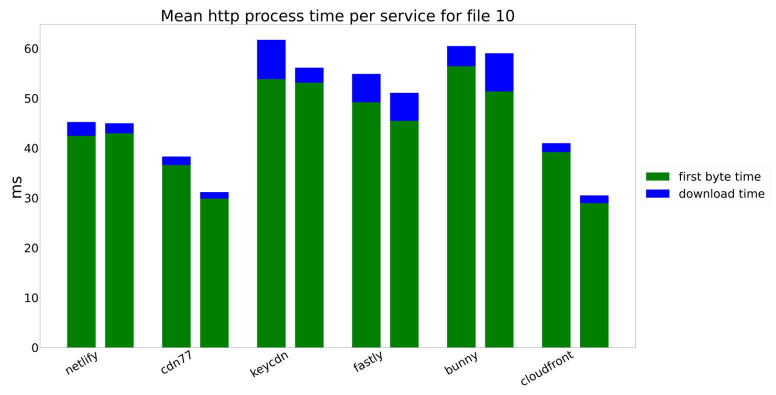


Figure 6.4 Mean of download times per service with Control D resolver

When compared to other visualizations from other data sets, some services kept similar results and some didn't, to evaluate the results more precisely, we have used the Paired T-Test on the median of the response time per location and the total download time per location. In both instances, the assigned servers should return data that are dependent on each other through location but are independent between locations or services and resolvers. Also, the differences between these values are normally distributed and so the Paired T-Test can be used.

To perform this test, we have used *Scipy Stats* module[1]. We have stated the null hypothesis such that the mean of the times measured on non-trace assigned servers will be equal or lower, with the alternative hypothesis that the mean times from trace assigned servers will be lower. When examined for the 5% significance level, we have rejected the null hypothesis in favor of the alternative hypothesis in some cases, which can be seen in the attachments.

When we examined the rejections, some services were present in almost all tested permutations of test type and resolver. With these data in mind, we can say that the Cloudfront servers assigned from DNS query with trace flag performed overall better and the mean measured times were lower. In terms of percentages, improvements in Cloudfront's performance ranged from 12.2% to 38.2% for download times and from 12.7% to 17.5% for the average response times. We have seen a statistically significant improvement for CDN77 download times in the range from 6% to 22.9% percentage wise, but the response times were not significantly better, as the improvement rate stayed between 5.6% and 11.2%. The download times were sometimes also better for Netlify and Bunny CDN, but they weren't consistent, so we will not conclude anything about the performance improvement. The Fastly service, on the other hand, performed often similarly or slightly worse when a trace DNS query was used.

[1]https://docs.scipy.org/doc/scipy/reference/stats.html

As the results vastly differ between the services, we don't find correlation between the CDN size and the results of measurements. Cloudfront as the biggest tested service performed overall better, but also the middle-sized CDN77 had seen significant performance improvements. On the other hand, Fastly, size-wise placed in between these CDN providers, doesn't profit from the client source address as well as the results from Bunny CDN, that don't show significant changes. The reason behind these differences is a good suggestion for future research. In this moment, we can only speculate that the service-specific setup of the load-balancing implementation may cause ECS to be inefective.

Even though we are unable to conclude the performance impact from the knowledge of client source address or part of it based on the size of CDN provider and the used resolver, we were able to confirm the possibility of performance improvement with the usage of ECS for some services. However, to achieve these improvements, more research is needed to understand why only some services benefit from ECS, and our application may be used to perform additional measurements.

Chapter 7
Conclusion

Our objective has been to map the EDNS Client Subnet extension and its performance impacts. We focus on its recommended implementation and use that is described in RFC 7871. We have looked into its drawbacks that are posed on ECS supporting DNS resolvers and outlined some reasons why it isn't supported by all public DNS resolvers including the biggest one of them – Cloudflare.

We wanted to introduce testing services that are used for testing the network state from many locations such as Globalping. We zoomed in on the CDN provider JsDelivr, which developed this testing utility as it gained its own worldwide testing service by providing the testing tools to others and rewarding users for running their own probes. We mentioned other testing service RIPE Atlas that provides more probes, but unlike Globalping, doesn't offer completely free testing without participation on the project. We dive into the offered testing utilities, their customizable options, and their implementations based on the REST API.

After introducing ECS and the need to review its performance impacts, which are most important for CDN providers that use DNS load balancing, we looked at possible metrics that could be compared against each other. Those metrics can be measured when the ECS enabled DNS query returns a different assigned IP than the ECS disabled query.

During the methodology proposal, we have substituted the use of ECS by the traced DNS query as it provides us with more flexibility and wider selection from possible DNS resolvers. We also specified the simulated content for delivery according to the global web statistics and the proposed methodology for evaluation.

Based on this methodology and the stated need for world-wide testing of the CDN load balancing capabilities, we have developed a CLI Python application. This application uses a standardized command-line interface, provides multiple possible ways of data import and export, and uses Globalping API to collect data about CDN services from multiple specified locations around the world.

After the application was implemented and tested, we gathered a list of CDNs, public DNS resolvers, set locations for the tests, and run our tests. Although the results were different from our expectations when the methodology was proposed, we were able to evaluate the data and confirm that some services perform overall better when ECS was simulated. We weren't able to confirm a correlation between the number of CDN servers, number of DNS servers and the outcomes. As this remains unanswered, more testing and research is required to better understand it. Based on our findings, possibly more complex methodology could be proposed to test the implications of different service sizes and perhaps find out the reason for these inconsistencies.

Our application can remain a solid testing ground for the ECS performance impacts, and it is possible to implement more modes of testing to reflect the possible shift in methodology proposal.

Bibliography

1. SILVA DAMAS, Joao da; GRAFF, Michael; VIXIE, Paul A. *Extension Mechanisms for DNS (EDNS(0))[online]* [RFC 6891]. RFC Editor, 2013. Request for Comments, no. 6891. Available from DOI: `10.17487/RFC6891`.
2. THE DNS INSTITUTE. *DNSSEC Guide : What's EDNS All About (And Why Should I Care)?* [Online]. ©2014-2017 [visited on 2024-02-24]. Available from: `https://dnsinstitute.com/documentation/dnssec-guide/ch03s05.html`.
3. CONTAVALLI, Carlo; GAAST, Wilmer van der; LAWRENCE, David C; KUMARI, Warren "Ace". *Client Subnet in DNS Queries[online]* [RFC 7871]. RFC Editor, 2016. Request for Comments, no. 7871. Available from DOI: `10.17487/RFC7871`.
4. ZSCALER INC. *About EDNS Client Subnet (ECS) Injection* [online]. ©2007-2024 [visited on 2024-03-03]. Available from: `https://help.zscaler.com/zia/about-edns-client-subnet-ecs-injection`.
5. NS1. *EDNS Client Subnet (ECS) extension* [online]. 2024-03 [visited on 2024-05-15]. Available from: `https://www.ibm.com/docs/en/ns1-connect?topic=overview-edns-client-subnet-ecs-extension`.
6. AL-DALKY, Rami; RABINOVICH, Michael; SCHOMP, Kyle. A Look at the ECS Behavior of DNS Resolvers [online]. In: *Proceedings of the Internet Measurement Conference*. Amsterdam, Netherlands: Association for Computing Machinery, 2019, pp. 116–129. IMC '19. ISBN 9781450369480. Available from DOI: `10.1145/3355369.3355586`.
7. ANDREY, Meshkov. *Privacy-friendly EDNS Client Subnet* [online]. 2024-01 [visited on 2024-04-12]. Available from: `https://adguard-dns.io/en/blog/privacy-friendly-edns-client-subnet.html`.
8. JACKSON, Kody. *FAQ - Cloudflare 1.1.1.1 docs* [online]. 2024-04 [visited on 2024-04-12]. Available from: `https://developers.cloudflare.com/1.1.1.1/faq/`.
9. ORACLE. *EDNS Client Subnet (ECS) FAQs & Information — Dyn Help Center* [online]. ©2024 [visited on 2024-04-17]. Available from: `https://help.dyn.com/edns-client-subnet-faq-info/`.

10. GOOGLE. *eDNS0 Client Subnet (ECS) - Interconnect Help* [online]. ©2024 [visited on 2024-04-17]. Available from: `https://support.google.com/interconnect/answer/7658602?hl=en`.
11. ALEXANDER, Harrison. *Umbrella and EDNS Client Subnet (ECS)* [online]. 2022-09 [visited on 2024-04-17]. Available from: `https://support.umbrella.com/hc/en-us/articles/360021857552-Umbrella-and-EDNS-Client-Subnet-ECS`.
12. QUAD9. *Frequently Asked Question* [online] [visited on 2024-04-17]. Available from: `https://www.quad9.net/support/faq/`.
13. IMPERVA INC. *What is Anycast Routing* [online]. © 2024 [visited on 2024-04-19]. Available from: `https://www.imperva.com/learn/performance/anycast`.
14. JSDELIVR. *Globalping - Internet and web infrastructure monitoring and benchmarking.* Available also from: `https://www.jsdelivr.com/globalping`.
15. JSDELIVR. *About - jsDelivr* [online]. ©2012-2024 [visited on 2024-03-15]. Available from: `https://www.jsdelivr.com/about`.
16. RIPE NCC. *What is RIPE Atlas? - RIPE Atlas — RIPE Network Coordination Centre* [online]. ©1992–2024 [visited on 2024-04-03]. Available from: `https://atlas.ripe.net/about`.
17. CLOUDFLARE, INC. *What is My Traceroute (MTR)?* [Online]. ©2024 [visited on 2024-03-20]. Available from: `https://www.cloudflare.com/learning/network-layer/what-is-mtr/`.
18. *Globalping API [online].* 1st ed. 2024. Available at `https://www.jsdelivr.com/docs/api.globalping.io`.
19. CHATZOPOULOU, Doxa; KOKKODIS, Marios. IP geolocation [online]. 2007. `https://www.researchgate.net/publication/228453630_IP_geolocation`.
20. QUAD9. *Service Addresses & Features* [online] [visited on 2024-05-14]. Available from: `https://quad9.net/service/service-addresses-and-features`.
21. JAMIE, Indigo; DAVE, Smart. *Page Weight 2022* [online]. 2022-10 [visited on 2024-05-08]. Available from: `https://almanac.httparchive.org/en/2022/page-weight`.
22. AHMADZAM. *Close Streamlit App with button click* [online]. 2023-07 [visited on 2024-05-07]. Available from: `https://discuss.streamlit.io/t/close-streamlit-app-with-button-click/35132/5`.
23. TONY, Kipkemboi. *Removing the deploy button* [online]. 2023-10 [visited on 2024-05-07]. Available from: `https://discuss.streamlit.io/t/removing-the-deploy-button/53621/2`.
24. PANDEY, Harsh. *How to Calculate z-score in Python* [online]. ©2024 [visited on 2024-05-07]. Available from: `https://flexiple.com/python/z-score-python`.
25. AMAZON WEB SERVICES, INC. *Key Features of a Content Delivery Network – Performance, Security – Amazon CloudFront* [online]. ©2024 [visited on 2024-05-09]. Available from: `https://aws.amazon.com/cloudfront/features/?nc=sn&loc=2&whats-new-cloudfront.sort-by=item.additionalFields.postDateTime&whats-new-cloudfront.sort-order=desc`.

26. BUNNYWAY D.O.O. *Global CDN Network — Low latency CDN with 114+ PoPs* [online]. ©2024 [visited on 2024-05-10]. Available from: `https://bunny.net/network/`.

27. FASTLY INC. *Fastly network map* [online]. 2024-03 [visited on 2024-05-09]. Available from: `https://www.fastly.com/network-map/`.

28. PROINITY LLC. *Network - KeyCDN* [online]. ©2024 [visited on 2024-05-09]. Available from: `https://www.keycdn.com/network`.

29. CDN77. *Status — CDN77 documentation* [online]. 2024-05 [visited on 2024-05-09]. Available from: `https://client.cdn77.com/support/status`.

30. CHRIS, McCraw. *Is there a list of where Netlify's CDN pops are located?* [Online]. 2024-01 [visited on 2024-05-10]. Available from: `https://answers.netlify.com/t/is-there-a-list-of-where-netlifys-cdn-pops-are-located/855/2`.

31. CLOUDFLARE INC. *Cloudflare Global Network — Data Center Locations* [online]. ©2024 [visited on 2024-05-10]. Available from: `https://www.cloudflare.com/network/`.

32. QUAD9. *Locations* [online] [visited on 2024-05-10]. Available from: `https://www.quad9.net/service/locations/`.

33. CONTROL D INC. *Network* [online] [visited on 2024-05-10]. Available from: `https://controld.com/network`.

34. CLEANBROWSING. *CleanBrowsing Network Status* [online]. 2024-05 [visited on 2024-05-10]. Available from: `https://cleanbrowsing.org/status/`.

35. ADGUARD DNS. *AdGuard DNS — ad-blocking DNS server* [online]. ©2016–2024 [visited on 2024-05-10]. Available from: `https://adguard-dns.io/en/welcome.html`.

36. IPINFO. *The trusted source for IP address data, leading IP data provider* [online]. ©2024 [visited on 2024-05-12]. Available from: `https://ipinfo.io/`.

Attachment structure

```
/
├── readme.md ........................................ Attachement structure
├── thesis ........................................ Thesis LaTeXsource code
│   └── sletrvoj_BP_2024.pdf ........................ Generated bachelor thesis
├── ecs_collector ................................................ Application
│   ├── service_sets ............................... Preloaded set of services
│   ├── location_sets ............................. Preloaded set of locations
│   └── visualize .................. Pregenerated data for the visualization demo
└── data ................................... Data and logs from data collection
    ├── inputs ................................. Input data for the measurements
    └── logs ........................................ Logs from the data collection
```