TECHNISCHE UNIVERSITÄT BERLIN

FAKULTÄT IV
FACHGEBIET INTERNET NETWORK ARCHITECTURES

Diplomarbeit in Informatik

Measuring EDNS-Client-Subnet Extension

Florian Streibelt

September 6, 2013

Aufgabenstellerin:	Prof. Anja Feldmann, Ph. D.
Zweitgutachter:	Prof. Dr. Odej Kao
Betreuer/innen:	Georgios Smaragdakis, Ph. D.
	Jan Böttger, Dipl.-Inf.
	Prof. Anja Feldmann, Ph. D.
Abgabedatum:	6. September 2013

Submitted and Joint Work

Parts of this thesis were written in collaboration and submitted as a conference paper:

Florian Streibelt, Jan Boettger, Nikolaos Chatzis, Georgios Smaragdakis and Anja Feldmann
Exploring EDNS-Client-Subnet Adopters in your Free Time (short paper)
ACM Internet Measurement Conference 2013.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 6. September 2013

Unterschrift

Abstract

Large Content Distribution Networks (CDNs) and Content Providers (CPs) rely on inference of the location of their users to assign them to appropriate servers. The client location is considered to be close to the DNS resolver utilized by the client. Recent studies have shown that this assumption does not always hold and can lead to degradation of the end-user end-to-end performance. Moreover end-users are more and more using public resolvers not operated by their ISP but at a global scale by companies like Google or OpenDNS, typically further increasing the distance between client and resolver.

The recently proposed Enhanced DNS (EDNS) Client Subnet Extension (ECS) is addressing this problem by providing (partial) information on the clients IP addresses to the authoritative nameservers of the CDNs and CPs and has been quickly adopted by major Internet companies such as Google.

Recent studies have focused on the penetration and performance implications of the Client Subnet Extension (ECS), this thesis shows that ECS also offers easy to use opportunities for active measurements. A key observation is, that ECS allows to resolve domain names of ECS adopters on behalf of any arbitrary IP/prefix in the Internet. Utilizing only a single residential vantage point and publicly available information, we are able to (i) uncover the global footprint of ECS adopters, (ii) infer the DNS response cacheability and end-user clustering of ECS adopters for arbitrary networks in the Internet, and (iii) snapshot the mapping of users to server locations as practiced by major ECS adopters.

While pointing out such new measurement opportunities, our work is also intended to make current and future ECS adopters aware of which operational information gets exposed when utilizing this recent DNS extension.

Zusammenfassung

Die großen Content Distribution Netzwerke (CDN) und Content Provider (CP) benötigen zur Verteilung der Anfragen auf geeignete Server Informationen über den ungefähren Standort der Clients. Es wird nun davon ausgegangen, dass die Clients sich in der Nähe der von ihnen gbenutzten DNS-Server befinden. Aktuelle Studien zeigen jedoch, dass diese Annahme nicht mehr länger zutrifft und sogar zu einer Verschlechterung der Leistung für den Endnutzer führen kann. Hinzu kommt, dass immer mehr Nutzer nicht mehr die Nameserver ihres Serviceproviders (ISP) sondern von Anbietern wie Google oder OpenDNS nutzen, die hierzu globale Infrastrukturen betreiben, welche die Entfernung zwischen Nutzer und Nameserver noch weiter erhöht.

Die vorgeschlagene Erweiterung des DNS, Client Subnet Extension (ECS), soll hier Abhilfe schaffen, indem (Teile) der IP-Adresse des anfragenden Clients an den authoritativen Nameserver des CDN oder CP übertragen werden.

Während kürzlich durchgeführte Studien sich hauptsächlich der Verbreitung und Effizienzsteigerung des Protokolls widmen, zeigen wir, dass ECS auch einige einfach zu nutzenden Möglichkeiten bietet, um aktive Messungen durchzuführen. Eine Hauptbeobachtung dabei ist, dass wir in der Lage sind Hostnamen der ECS-Anbieter so aufzulösen als würde diese Anfrage aus einem beliebigen Netz im Internet stammen. In unserer Arbeit zeigen wir, dass wir damit von einem einzigen Standort aus und nur unter Nutzung öffentlich verfügbarer Daten in der Lage sind, (i) die globale Ausbreitung eines ECS-Anbieters zu messen, (ii) Aussagen zum Caching der DNS-Antworten und dem Nutzer-Clustering für beliebige Netze zu treffen, und (iii) Momentaufnahmen der Zuordnung von Nutzern und Netzen zu den Server-Standorten der ECS-Anbieter zu erstellen.

Wir möchten mit dieser Arbeit nicht nur diese neuen Methoden für aktive Messungen vorstellen, sondern auch das Bewusstsein aktueller und zukünftiger ECS-Anwender darauf lenken, welche möglicherweise internen Informationen sich bei der Verwendung dieser Erweiterung durch Dritte ermitteln lassen.

Contents

List of Figures

List of Tables

Glossary

API Application Program Interface.

AS Autonomous System.

ASN AS Number.

BGP Border Gateway Protocol.

CDN Content Distribution Network.

CIDR Classless Interdomain Routing.

CP Content Provider.

DNS Domain Name System.

DNSSEC Domain Name System Security Extensions.

ECS Client Subnet Extension.

EDNS Enhanced DNS.

FQDN Full Qualified Domain Name.

GGC Google Global Cache.

IETF Internet Engineering TaskForce, informal standards organization.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

ISP Internet Service Provider.

JSON JavaScript Object Notation.

PI Space Provider Independent Address Space.

RTT Round Trip Time.

SMTP Simple Mail Transfer Protocol.

SPF Sender Policy Framework.

SSH Secure Shell.

TCP Transmission Control Protocol.

TLD Top Level Domain.

TTL Time To Live.

UDP User Datagram Protocol.

XMPP Extensible Messaging and Presence Protocol.

Chapter 1.

Introduction

In the current Internet a growing amount of inter-domain traffic is generated by Content Distribution Networks (CDNs) and Content Providers (CPs) [32]. From their globally distributed infrastructures content like images, videos or other web resources is served to clients. This architecture can have several advantages ranging from load distribution across servers or data centers, robustness of services by enabling automatic failover to other clusters or elastic services for customers with heavily varying workloads.

In this work we focus on a problem that the operators face when mapping clients requesting a resource, e. g., a video file, to the servers capable of serving the requested data. Besides other factors, like the load on the servers in question, one main metric is the latency the request is served with. To minimize latency, knowledge of the *distance* between a client requesting a resource and the servers suitable for serving these requests is required. When speaking of distance in connection with computer networks we usually refer to a metric like the hopcount, that is the number of routers between adjacent points in the network.

A common abstraction is made by using the network-centric definition of locations, instead of the geographic location of the physical machines, as geolocation databases are easy to access and publicly available[1].

An approach often used by CDNs and CPs is to utilize the Domain Name System to infer the client location and map the client to servers [35, 50]. The general concept works by the premise that the nameserver used by the client usually is operated by the client's corresponding Internet Service Provider [49] and thus the client is located relatively close to this nameserver. Because the IP address of the nameserver is known from the request it sends to the CDN, mapping of the location utilizing a geoIP database is possible and a response can be created that sends all clients of that resolver to a specific cluster or server.

This method for inferring the client location is doomed to failure as soon as clients and corresponding resolvers are not located *nearby* any more. Several studies [13, 33] have shown that these assumptions on locality in fact do not always hold which, in turn, can lead to degraded end-user experience.

Notably, the debut of third party resolvers like Google Public DNS [4] and OpenDNS [7] has increased this trend as end-users, taking advantage of these as resolvers, may experience poor performance[13, 37, 43].

[1] e.g. `http://maxmind.com`

This is a result of the mis-mapping of clients to servers located near the nameservers of the public DNS providers but not near the client.

To address this problem the EDNS Client Subnet Extension (ECS) has been proposed to the IETF [21] to reintroduce a relation between the requesting client and the source of a DNS request an authoritative nameserver sees. This is achieved by adding information about the client subnet to the DNS request forwarded to the authoritative nameserver.

Thus, it is not surprising that some major Internet companies (e. g., Google, Edgecast, and OpenDNS) have already adopted ECS and have established the consortium "a faster Internet" [1]. The fact that ECS can indeed help improve end-user performance is highlighted by extensive active measurement studies [37, 43].

This thesis shows that indeed the information submitted via ECS is already being used by major companies that have adopted the ECS extension. We also show that by crafting packets with arbitrary client IP information it is possible to map the infrastructure of these early ECS adopters by collecting the returned server addresses for a number of pretended client locations. By comparing these snapshots of client-to-server mapping it is also possible to observe the growth of providers or changes in their operational practice, as we demonstrate on datasets obtained for Google and YouTube.

We emphasize that ECS allows everyone to resolve domain names of ECS adopters on behalf of any arbitrary IP/prefix in the Internet and thus offers unique, but likely unintended, opportunities to uncover details about these companies' operational practices at almost no cost. For our measurements we mostly utilized only a single residential vantage point with an ordinary Personal Computer.

Thus, ECS queries can help uncovering the sophisticated techniques that CDNs and CPs use for mapping users to servers (e.g, see [31, 30, 33]). Indeed, this currently hard-to-extract information can now be collected using only a single vantage point and relying on publicly available information. In the past, to obtain similar information, network researchers had to find and use open or mis-configured resolvers [12, 29, 47], have access to a multitude of vantage points in edge networks [37, 43], rely on volunteers [13, 14], analyze proprietary data [32, 40], or resort to searching the Web [46].

In chapter 2 we will explain the technical background of the protocols and systems used in this work and explain some of the challenges involved. In chapter 3 we take a closer look at the datasets used in our experiments and how we obtain them while chapter 4 explains the methodology of our experiments and the framework we developed for conducting our measurements. This chapter also contains a section on problems we encountered during the work. We will then progress by looking at the results in chapter 5 before we discuss the outcomes in chapter 6.

Chapter 2.

Background

2.1. The Domain Name System

The DNS is a globally distributed, hierarchical database. It was first used in 1983, initially to map hostnames to addresses and vice versa. Until then a simple textfile, called HOSTS.TXT, was used[34] that contained all hostnames in use and the addresses associated to them. With the growing number of nodes connected to the developing Internet, not only the number of entries in the file increased but also the rate at which the contents changed.

The solution to this problem was the invention of the DNS.

By designing DNS as a distributed system, each domain can be operated independently. Also this allows for scaling of the whole system. Key element is the subdivision of the namespace in zones building a tree with the rootzone, called '.', on top. The first level of subdivision would be the top level domains, like .edu, .net or .com - each forming a tree again with sub-entries like example.org[1]. The last dot for the rootzone is usually omitted but in a strict representation would be part of the Full Qualified Domain Name (FQDN) of a host. This hierarchy allows to delegate the responsibility of a subtree, like .edu, to distinct nameservers that only handle entries within the .edu subtree. Because every entry here again can be seen as a subtree with its root being .edu, another delegation can, and usually will, be in place. Thus it is possible to operate designated nameservers for domains like example.org.

At the root of the DNS there are the so-called root-servers, which delegate the Top Level Domains (TLDs), such as .com, .net, .org and .de to the nameservers of the respective registries managing the respective TLDs.

The nameservers of a TLD now hold a list of all Internet domains registered below the TLD. The domain name *example.org* consists of the TLD *.org* and the domain part *example*. In this case the nameservers of the *.org*-TLD will delegate further request for names within the example-domain to nameservers operated by example.org.

The steps involved in resolving a hostname to an IP-address can be seen in Figure 2.1. Here we see in step 1 how a client queries the local resolver at the ISP for the IP address of the hostname www.example.org. This nameserver, acting as recursive resolver, will send the query to one of the rootservers, seen in step 2. The IP addresses of these rootservers are configured locally on each recursive resolver, as they are the entry point to the hierarchy of the DNS.

[1] Per [10] this is a reserved second level domain name for documentation.

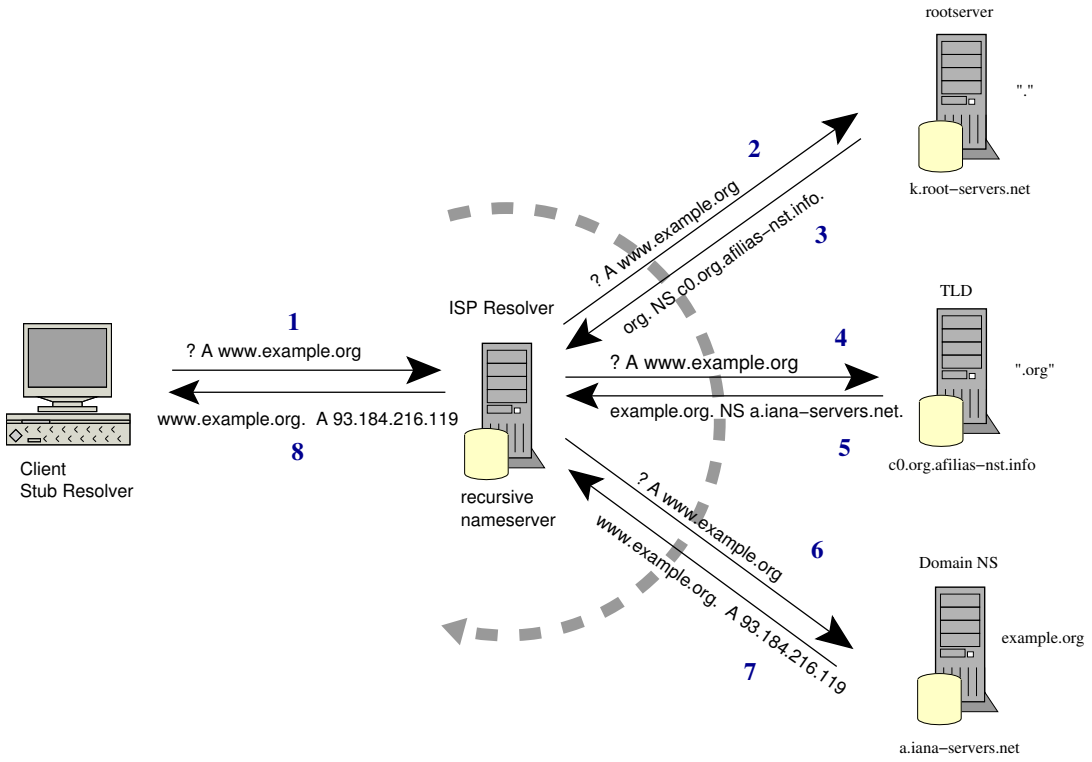


Figure 2.1.: Delegation in the Domain Name System

In step 3 the rootserver informs the nameserver at the ISP that it does not know the A record in question, but it delegates the question to the nameserver of the .org Toplevel Domain. Step 4 subsequently shows the nameserver at the ISP to follow that delegation and ask the nameserver of the .org TLD for the A-record in question. Yet again a delegation is returned in step 5 that will finally lead to the nameserver operated by the domain in question. In steps 6 and 7 we see how the nameserver of the ISP is finally able to retrieve the result from the authoritative nameserver of the example.org domain and return it to the client in step 8.

2.1.1. DNS Message Format

By default DNS messages are transferred via UDP for several reasons. First of all, the overhead of the TCP three-way-handshake would increase the latency of name resolution and thus the setup time for new data connections. Secondly, the amount of data transferred in each request/response is very small and fits in a single packet - considering the three-way-handshake and teardown signalling this again shows the amount of overhead TCP introduces in this application. Thirdly due to the delegation concept of the DNS, a single nameserver will query many different servers, very often only for one or two records, such that the setup of an TCP connection would not have any benefits here as well, even though the number of queries sent and received can be quite high.

There is, of course, one exception to this rule: TCP is used whenever the amount of data does not fit into a single data packet any more, for example when nameservers of a domain

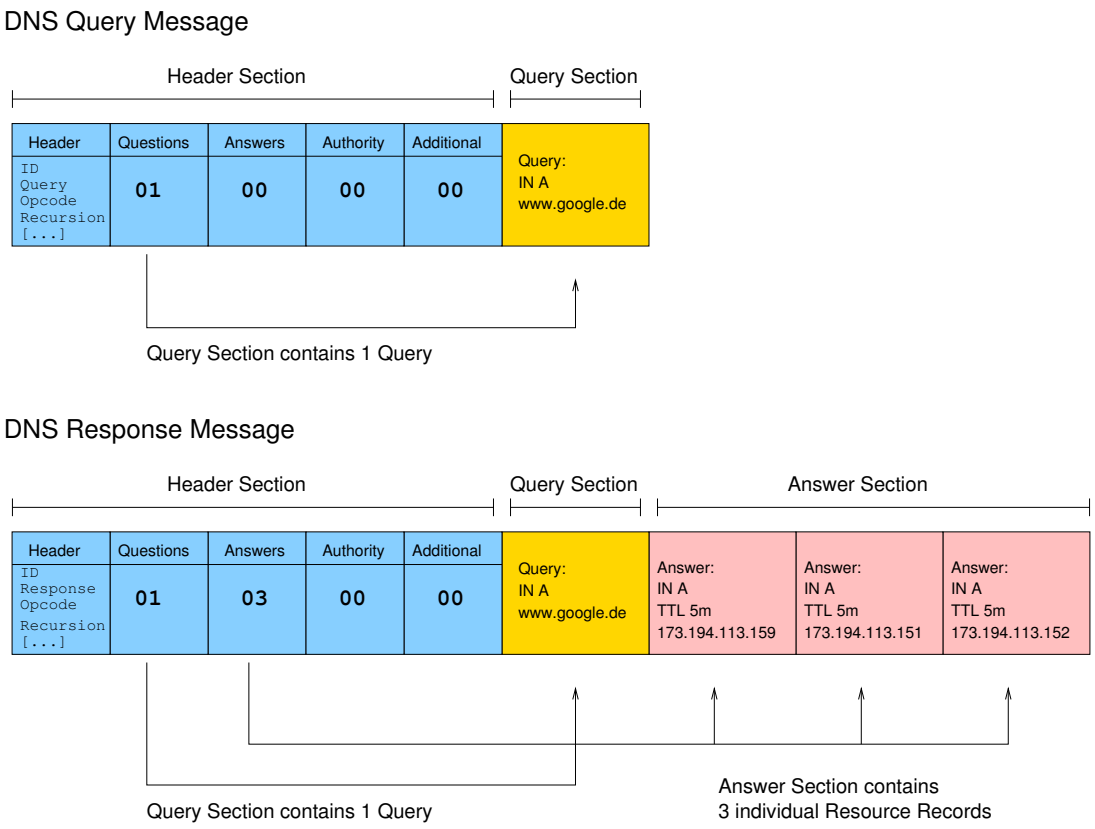


Figure 2.2.: Sample DNS query and response message

synchronize a whole zone (zone transfer) at once. A nameserver can signal a client to re-issue a request using TCP for this purpose.

Both the DNS request sent to a nameserver and the response sent to the client are constructed in the same way. Each DNS message consists of several sections, of which some are mandatory. Each of these sections can contain several Resource Records, explained below, and the header section states the count of these.

Figure 2.2 shows an example request and response. It is easy to see that the response message simply consists of a copy of the query with the answer section added and some minor changes to the header section, e. g., the count of sections. This is done on purpose as it allows for efficient implementations of server software. The query can be copied in a buffer and needs only minimal changes before it can be send back to the client.

2.1.2. Resource Records: DNS Beyond Address and Name Resolution

The data in the sections of a DNS message are organized in Resource Records, representing the various types of information.

The record types PTR, A and AAAA are related to address resolution. While A and AAAA are used for mapping of names to IPv4 and IPv6 addresses respectively, a PTR-record is used to map an IP-address (both IPv4 and IPv6) to hostnames. Lookups for addresses are called forward-lookups while a reverse-lookup maps an address to a hostname.

Nameserver Records (NS) are used to delegate a zone to an (authoritative) nameserver. When a client queries a nameserver asking for a record and an NS record is returned, this is called a delegation. The client should now resend the query to the nameserver pointed to in the NS record.

Besides address resolution the DNS is also used to accomplish other tasks, for example service discovery. The most prominent service is probably the discovery of a mail server that accepts e-mail messages for a certain domain. This works by sending a query for the MX Resource Record of a domain to the nameservers and will result in a list of hostnames that act as mail exchangers, thus the name, for the domain. This mechanism was generalized by the introduction of the SRV Resource Record that is used e. g., by services like the Extensible Messaging and Presence Protocol (XMPP) instant messaging protocol.

Another Resource Record we like to mention is the TXT record. It allows for free-form text and can be used as an example of how protocols can be used for other than the intended purposes.

As explained in the last section, the DNS protocol was designed with performance in mind, thus it is no surprise that an extension of the protocol is not easy as there are almost no spare bits in the message headers.

This lead to the use of the TXT-Record for various purposes in the aim of extending the ECS. One is the publication of e-mail server policies in the Sender Policy Framework (SPF) [51], while the TXT record type is also used to confirm the ownership of a website with the Google webmaster tools[2].

While address resolution is the functionality we focus on in this thesis, it is worth noting that the DNS gained attraction for distribution of other data as well. Further examples are publication of Secure Shell (SSH) key fingerprints [44] or SSL-Certificates [28].

2.2. Content Distribution Networks and Content Providers

With the growing use of multimedia content on websites and the appearance of video streaming services, the traditional way of publishing content in the Internet lead to various problems. Serving customers world wide from a single location makes bottle necks in the available bandwidth visible, e. g., between continents or regions. Also over-provisioning of infrastructure to handle peak-loads was found to be expensive.

A natural development thus is the appearance of globally distributed service providers specialized in providing data distribution on the application layer.

[2] `https://support.google.com/webmasters/answer/176792?hl=en`

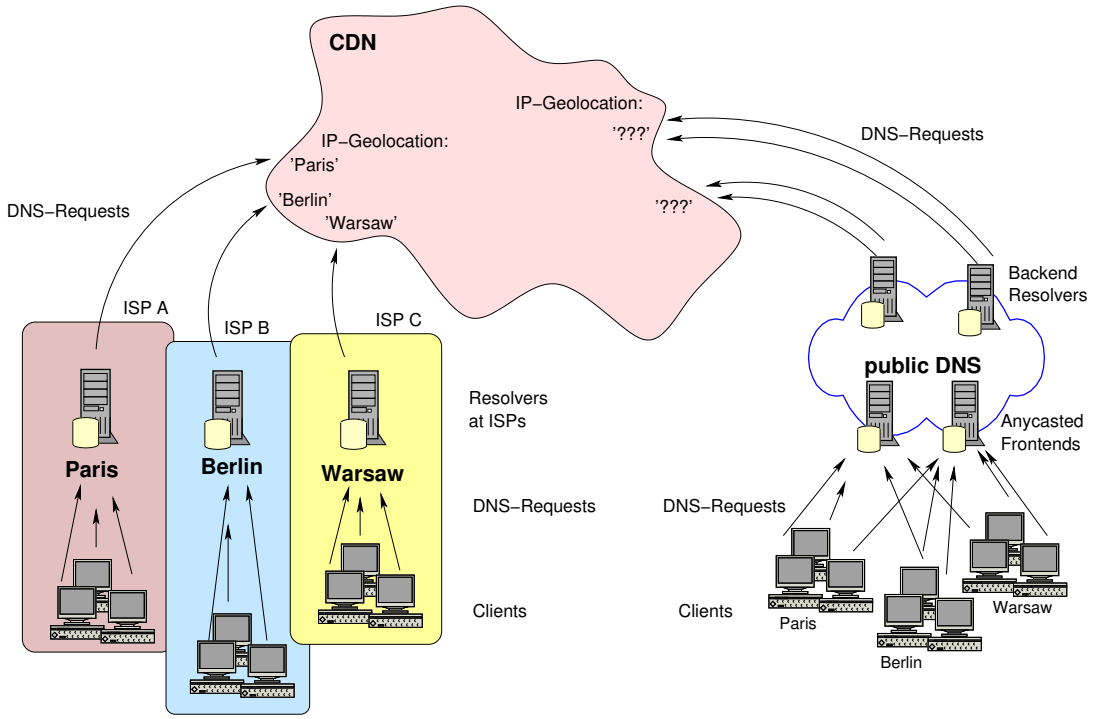


Figure 2.3.: Problems in client-location mappings

One example is the distribution of media files to a worldwide audience. This could be handled by a CDN, providing caches in various locations worldwide. When an increase of requests from a certain region is detected, the requested files are copied to a cache near that region and all clients from that region requesting the file are redirected to that location. If the origin of the data was North America and the clients are in Europe, this could mean that the data is copied only once to a cache in Europe and all clients are then served from that location without the need of per-client transatlantic data transfers. It is obvious that this increases the overall end-to-end performance for the clients requesting the file.

By combining the data of several customers within the caches the CDN can also handle peak loads with less overprovisioning, compared to the number of customers. Assuming that not all customer data will be requested at once, the CDN does not need much more overprovisioning than a customer would need on his own.

2.2.1. DNS for Client to Server Mapping

One of the main interests of CDNs and CPs is to assign a client requesting a resource to a server that can fulfill the request as quickly as possible, including the time for data transfer. Usually this is a server that (i) is not overloaded and (ii) is near the client from a network point of view. This redirection of clients should also happen as early as possible and with as little overhead for the CDNs provider as possible. The natural choice here is to use the DNS for this purpose, as this is the first contact between client and CDN. Also DNS responses are usually cached within the network by recursive nameservers, not operated by the CDN providers.

Figure 2.3 shows on the left side how this mapping is done. Clients in Paris, Berlin and Warsaw are using their respective ISP resolver to resolve a hostname that points to the CDN. The idea is that the authoritative nameserver at the CDN can use the source-IP address of the DNS request to geolocate the recursive resolver of the ISP and thus can infer the geographical client location. These steps of indirection lead to inaccuracy, for several reasons.

First of all, from a geo-IP database only approximate geographic locations can be found, while the network-location would be of interest. Secondly, not the client is located, but the resolver used by the client, because the IP address of the client is unknown. The general assumption is that clients are located near the resolvers they utilize, and that the accuracy is still sufficient for mapping clients to servers.

These assumptions seem to be proven wrong [13, 33] and the problem gets even worse in the case of clients using public DNS resolvers like Google or OpenDNS.

On the right figure 2.3 shows how, seen from the CDN provider's perspective, the clients "hide" behind the public DNS provider. Often not even the country the client is located in can be found. In this scenario it is obvious that the long held assumption of client and resolver being located nearby has to be dropped.

Even though in the case of Google a list of nameservers and their location on city level is published[3], it is obvious that this approach cannot scale.

2.3. The ECS EDNS Extension

The Client Subnet Extension (ECS) [21] was introduced to tackle the problem of mis-locating the end-system which originates the DNS request. The problem is that the end-system's IP information is typically hidden from the authoritative name server. With ECS, client IP information is forwarded by all ECS enabled resolvers to the authoritative name server in the form of network prefixes.

2.3.1. EDNS

To understand the modifications of the DNS protocol that come along by using ECS, we first need to look at the underlying extension mechanism EDNS, often referenced as EDNS0 due to a specific header field set to the value zero.

The DNS protocol was developed with performance in mind. As the speed of name resolution is crucial for the setup-time of new connections, UDP was chosen as transport protocol, eliminating the need of the RTTs involved with a TCP handshake. This is also reflected in the effort of making the DNS packets as small as possible, e.g. by using pointers within the packet when labels[4] are used more than once, removing redundancy.

This lead to the problem that the message headers had no room to add new signaling when the protocol needed to be extended – all header fields had been in use. To address this issue and

[3] `https://developers.google.com/speed/public-dns/faq#locations`

[4] labels are all strings separated by dots, e.g. example in example.org

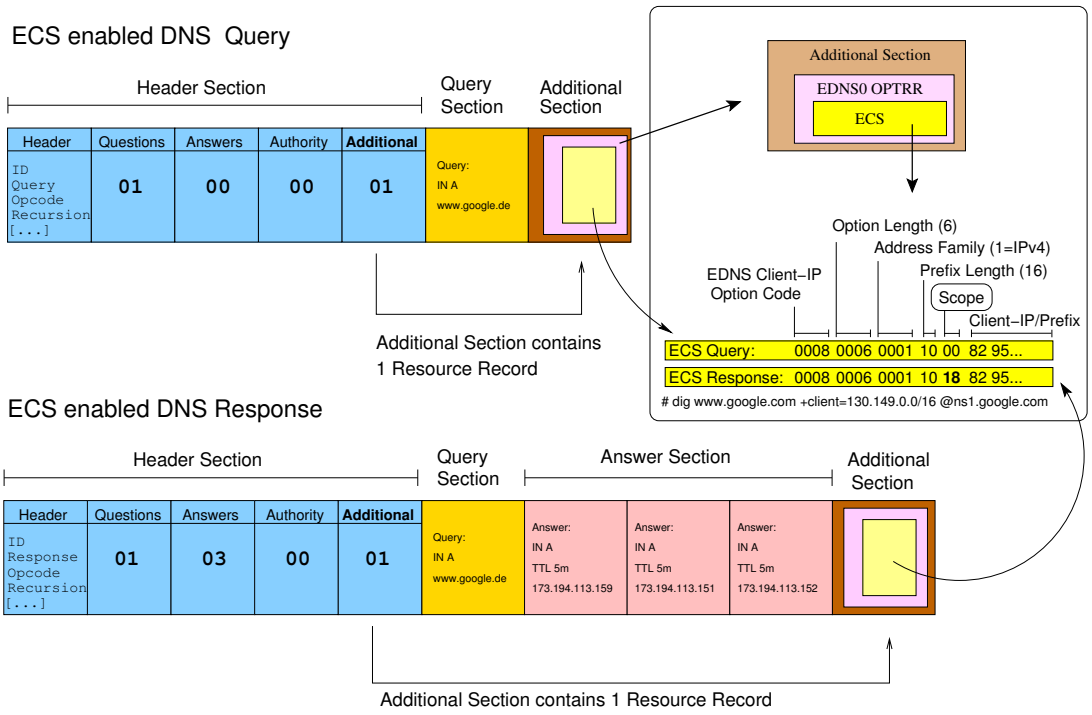


Figure 2.4.: Sample ECS query and response message

allow flexible extension of the DNS protocol in the future, EDNS0 [48] was proposed by the IETF DNS Extensions Working Group.

In contrast to normal DNS messages (see fig. 2.2) EDNS0 adds an ADDITIONAL section containing a pseudo Resource Record (OPTRR) to the query message. Under normal circumstances the ADDITIONAL section is only used to inform a client on additional data that it would most likely request next, e. g., the IP address for an MX record, that is only allowed to contain hostnames. By following the general rules of using a Resource Record, this extension is compatible with traditional DNS, as name servers that do not support EDNS0 either strip the EDNS0 OPTRR in the ADDITIONAL section or forward it unmodified as they detect a legal data structure in the section.

Since all sections from a DNS query are usually also present in the DNS response, bidirectional data transfer is enabled as the DNS server can modify this pseudo Resource Record in the answer sent to the client.

2.3.2. ECS Protocol Specification

An example ECS enabled query and response is shown in figure 2.4. The ADDITIONAL section contains the EDNS Resource Record (OPTRR) with our ECS data.

The ECS payload, as shown on the upper right, consists of the EDNS option code, which is 0x008, a length field describing the length of the ECS data, the address family used for the client IP, i. e., IPv4 or IPv6, a prefix length, scope and finally the significant bits of the client

prefix. If the prefix does not consist of a full IP address (32 bits for IPv4, 128 for IPv6) the trailing octets which would be zeroes can be omitted, resulting in a shorter payload.

To protect a client's privacy, [21] recommends to use prefixes less specific than 32 for IPv4. In each query the scope field must be zero and is a placeholder for the scope returned by the server.

The response from an ECS enabled DNS server only differs in a single byte, namely the scope, which is needed for DNS caching: The answer can be cached and used for any client requesting the same resource with a client prefix that is a more specific or equal to the prefix as specified by the scope.

We note that the response may contain a different scope than the query network mask, and we have indeed observed larger as well as shorter scopes than prefix length in our measurements. In the example, the query prefix length is 16 while the returned scope is 24. The scope is the essential element that allows us to infer operational practices of ECS adopters, besides the A records returned in the DNS answer.

It is also worth noting, that by using ECS the nameserver being queried cannot simply re-use the buffer the request has been received in, as the answer section has to be inserted between header and additional section, and the size is dynamic. Also we would like to note that in figure 2.4 the answer from an authoritative server is shown although no authority section is present. This is not a mistake but an optimization than can be observed e. g., on ns1.google.com which was queried in this case, to decrease packet size, as the information in that section is usually not used by the expected clients, e. g., stub resolvers.

2.3.3. Challenges in Enabling ECS

While ECS is transparent to the end-user, it requires significant efforts by the operators of DNS servers. Here we have to distinguish between the different ways nameservers are being used.

First of all the authoritative nameservers that are holding the address information need to be ECS enabled. That means these servers have to understand the ECS extension and return an ECS compatible answer. For this they have to at least return the additional section without modifications in the answer, resulting in the scope to be zero, indicating that ECS was not used to generate the answer. This also means that larger DNS-packets have to be accepted and the generic EDNS extension has to be enabled. The current operational practice of the "a faster Internet" [1] group then requires an ECS adopter to send an e-mail to the mailing-list of the consortium which will cause an engineer, e. g., from Google, to manually check the authoritative name servers and white-list them as ECS compliant.

If the authoritative nameservers are not reachable directly, all forwarders between the clients and these nameservers need to be upgraded as well and at least forward the ECS information unmodified to their upstream resolvers. If these servers act as caching server this also means that they have to actively understand the ECS extension and cache according to the scope returned in the answers.

Adding appropriate cache support to the DNS resolvers can be quite a challenge, as ECS with its notion of scope introduces another factor to decide on a cache hit. For each query,

the resolver has to check if the IP address of the client lies within the scope of any cached result. If not, the query has to be relayed with the appropriate prefix information. Just imagine the extreme scenario—scope of 32. Then, the resolver should keep a separate entry per client making caching largely ineffective.

Overall, handling ECS is quite complicated as the draft requires DNS forwarders to forward the ECS information sent by the client. It may modify the prefix mask to a less specific. If no ECS information is present in the DNS request, the forwarder may add an OPTRR record based on information from the socket. This is the rule followed, e. g., by Google's public DNS servers. Note, that until Google enabled EDNS for DNSSEC[16] support, it stripped the ECS records. Now, this information seems to be forwarded unmodified.

Among the major obstacles identified so far are:

- ECS support in server software is not widely available. We find that only PowerDNS supports ECS as authoritative nameserver but not as resolver. Moreover, only for some clients e. g., dig (part of the BIND nameserver) and dnspython patches are available.
- All involved DNS servers need to be upgraded. CDNs may internally use multiple resolution levels, e. g., Akamai [35], requiring large scale redesigns,
- Third-party resolvers are not necessarily sending ECS queries by default, manual whitelisting may be needed.

2.4. Internet Routing

The Internet is a global network of interconnected networks. Each of these can be operated separately and by different entities, although global coordination for e. g., address allocation is needed.

Often multiple connections between networks exist and routers decide which route the data packets take. These routers use routing protocols to determine the best path and ensure, that each network can reach all other connected networks in the Internet and also provides resistance against failures of single links between networks by choosing a different route on link failure.

2.4.1. Autonomous Systems, Peering and Transit

The concept of an Autonomous System (AS) plays a key role when it comes to routing. An Autonomous System consists of one or more networks operated by a single entity. Each AS has a unique AS Number (ASN) that is used in the Border Gateway Protocol (BGP), explained below, to identify the AS.

Operators of ASes that want to directly exchange traffic can setup a peering, that means a direct data connection between routers in their network and the exchange of routing information, usually via the BGP, among them. Often this happens on mutual agreements without fees, as both parties involved profit from exchanging traffic with each other. To reach other ASes without direct peerings, so called Transit Providers, explained in the next section, are needed.

Smaller networks may not be an AS on their own but part of the AS of an ISP that allows customers to use parts of the address space allocated by the ISP. These networks then do not use Provider Independent Address Space (PI Space) and do not need to set up routing sessions as usually static routing is used for the connection to the ISP over dedicated network links.

2.4.2. Transit Providers

Looking at the size of the Internet it becomes clear that it is not feasible to directly connect with all the other ASes in the Internet. Yet we know that it is possible to reach all networks in the Internet, so other techniques must exist and one is the use of Transit Providers.

Transit Providers are usually large Internet Service Providers with a global infrastructure that have peerings with many different ASes – for a finer grained classification see [22].

These providers allow their customers to send and receive traffic to and from other ASes through their infrastructure. This is associated with a fee for that service. Also they will publicly announce (via BGP, see below) the reacheability of their customer's ASes/networks. This means that not only the other customers but also peering partners of these providers get to know what networks are reachable through this Transit Provider and thus will send traffic for these networks to this provider.

2.4.3. Network Prefixes, CIDR and Netmasks

Traditionally the IPv4 address space was split into subnets of fixed sizes, named network classes A to D. With classful routing the first bits of an address indicate the class of the network and each network of a certain class has a fixed size. Thus from the network base address it is possible to infer the size of the network being addressed.

This is achieved by splitting the address space using the binary representation of the IP addresses. In a class A network the first 8 bits are used as the network address. This represents also the first octet in the decimal dot-notation of addresses. A common way to express this is the notation 4.0.0.0/8 which represents a traditional class A network with the first usable IP address 4.0.0.1 - the network address cannot be used for host addresses. The IP network written as 130.149.0.0/16 represents a class B network where 16 of the 32 bits of the IPv4 addresses are used as network address and the rest as hostpart within the network. In this network 65534 addresses can be assigned to hosts, two addresses are reserved - the already mentioned network address and the broadcast address, addressing all hosts within a network at once. This address is defined as all bits of the host part set to one: 130.149.255.255. Similar a class C network uses 24 bits as network address and only 8 bit as host part allowing for 254 hosts. Class D networks are a special case as they have been reserved for multicast, addressing more than one host at a time.

Over time it was found that this solution is too inflexible and in 1993 the Classless Interdomain Routing (CIDR) [26, 25] was introduced, obsoleting network classes. Now the prefix length, that is the number of bits used as network ID, can be set without restrictions.

It is common to speak of a prefix instead of an address range, especially in the routing context. For example the prefix 130.149.0.0/16 represents the network with the base address 130.149.0.0 and a netmask of 16 (bits) referred to as prefix length of 16.

2.4.4. The Border Gateway Protocol

The Border Gateway Protocol (BGP) is used in the Internet to exchange routing information between routers of ASes. While within a network any routing protocol, or static routing, can be used, in the Internet BGP is in practice the only protocol used to establish peerings with other ASes. This means that every network that should be reachable in the Internet has to be announced via this protocol at some point, either directly or in aggregated form as part of a larger network.

Operators of an AS will use the BGP to exchange routing information with all of their peering partners by configuring a *BGP-session* between their own router(s) and the routers of the peering partners.

Important for the understanding of this work is that these BGP sessions can also be established without exchanging other data or announcing networks, for example to collect the information about the ASes being announced and the network prefixes reachable via these ASes. In section 3.1 we discuss some of the limitations with regards to this approach of collecting routing information.

A detailed description of the protocol itself can be found in the corresponding RFC [42].

2.4.5. Anycast

While usually a network prefix is usually only announced from one AS in one location, it is also possible to announce the same network prefix from different locations and thus allow several hosts to answer requests to the same IP address. This practice is called anycast[38] in contrast to unicast, where only one host is addressed – here any host that got assigned the IP address is allowed to answer.

Today this is for example used to increase the stability of the root nameservers[20]. By using anycast, requests to the anycast-enabled root nameservers are always sent to the nearest server available, as the routing table prefers the shortest path when multiple paths are available to a destination. Thus load can be distributed over more root nameservers without the need to reconfigure all recursive nameservers. Also this allows for the root-zone to be unchanged although the number of servers can be increased. Another important improvement is the robustness against denial of service attacks - because now the traffic of these attacks will only affect a limited number of anycasted instances of the rootservers. If these anycasted nodes are placed in “regions with poor external connectivity” [11] this can even prevent the connection to be congested if attacks are ongoing only within that region and otherwise improve the speed of the service.

Another example are the IP addresses of the public DNS service of Google, 8.8.8.8 and 8.8.4.4, both being anycasted for similar reasons.

Using anycast is especially easy for DNS, as it is a stateless protocol using UDP and no long standing connections are established that would be interrupted when the underlying routing changes and another host is answering requests to the anycasted IP address.

2.5. Related Work on ECS

During our research we found two other groups also conducting experiments on ECS.

One experiment is described by an anonymous author with the nickname b4ldr in a blog[5], where scripts for the nmap portscanner are provided to scan an ECS enabled nameserver. The scripts are categorized as 'dirty hacks' by the author and provide a proof-of-concept implementation to map IP address locations against the MaxMind database, a publicly available database of IP address geolocations.

We also found a study by Calder et al. [18] that investigates the growth of the global infrastructure run by Google by comparing the results found for ECS enabled DNS queries over time that will be referenced in the Evaluation (Chapter 5).

Chapter 3.

Datasets

In this chapter we explain the datasets that lay the foundations to our analysis, described later in Chapter 5 and how these are obtained.

We are using two different groups of datasets as inputs for our experiments. The first is a list of prefixes to be used as pretended "client location" for the ECS queries, derived from various datasources. The second is a list of popular ECS adopters we found using different techniques explained below.

3.1. Network Prefixes

To better understand the mapping of end-users to CDN servers and analyze the cacheability of DNS answers, we are interested in using as many different client IP-addresses as possible in our requests to cover as much address space as possible. On the other hand we want to avoid unnecessary requests to speed up our measurements and not to trigger automated alarms on the systems being queried. Also we want to make sure to use client-IP addresses that are in use by real clients.

While in principle one list of prefixes may be considered sufficient, we also want to know if smaller datasets are sufficient to gain comparable results. Another question regards possible IP-space profiling done by the CDNs and CPs which we suspect can be seen in deviations in the results for special ranges of client IPs.

We use both private and public sets of network prefixes in our analysis, also to explore if the usage of publicly available data is sufficient for our purpose.

The datasets used as client prefixes are:

Academic Network (UNI). This network includes a very diverse set of clients, ranging from office working spaces to student dorms and research facilities which complicates usage profiling. This network uses two /16 blocks that are not located next to each other. The networks do not have an AS on their own, and are localized to a single city in Europe.

Large ISP (ISP). The dataset includes more than 400 prefixes, ranging from /10 to /24, announced by a European tier-1 ISP. This ISP offers services to residential users as well as enterprise networks and also hosts CDN servers. This dataset is obtained via queries to the whois system/database.

Large ISP, de-aggregated prefixes (ISP24). We also use the de-aggregated announced prefixes of our large ISP at the granularity of /24 blocks to investigate if the finer granularity lets us uncover additional operational details.

Popular Resolvers (PRES). This proprietary dataset consists of the 280K most popular resolver IPs that contacted a large commercial CDN. These resolvers are distributed across 21K ASes, 74K prefixes, and 230 countries. The resolvers are ranked based on the number of requests they sent to the CDN. This set of top 20K resolver IPs is quite stable throughout our experiment. This is consistent with recent observations [36].

RIPE RIS (RV). This is a publicly available dataset of full BGP routing tables, collected in 17 locations [8]. The unique number of announced prefixes is about 500K, originating from all (around 43K) active ASes. Notice that the collection of prefixes takes places in different network aggregation level points and geographical locations, thus there is a degree of prefix overlapping. This overlapping is beneficial for the purpose of our study as we will show in Section 5. While the RIPE dataset is updated every five minutes, we are using a daily summary available for download as compressed input files for the *bgpdump* utility.

Routeviews (RV). This is another public BGP routing table source, offered by the University of Oregon [9] using other vantage points for collection of the data.

In contrast to the work presented by Calder et. al [18] we are not splitting the IPv4 address space at an aggregation level of /24. One reason is that this will lead to much more client prefixes to probe in the experiments and because in out approach we wanted to compare many different ECS adopters this would have prevented efficient scaling of our experiment. We also think that the list of publicly announced networks, as collected by RIPE and Route Views, to a large extent covers the used address space, while we are aware of certain limitations. Recent studies [15] show, that the data collected by RIPE and Route Views indeed is not complete. Only a fraction of peerings and not all aggregation levels of the networks can be found, as some operators seem to aggregate subnets into larger prefixes before announcing them publicly.

3.2. Content Provider Datasets

To compile the list of ECS adopters we considered different approaches. Some approaches have been discarded as unusable due to technical, legal and scalability issues and are described in Section 4.5.8 (scanning of the nameservers of all registered .com, .net and .de-domains) and Section 4.5.7 (using collected sFlow data samples).

For our experiments, we need to identify ECS adopters and the corresponding hostnames. The nameservers to query can easily be derived from the DNS by querying the nameservers of the corresponding TLD.

While a first starting point was the website of the “A Faster Internet Consortium” [1] we looked for a more systematic approach than extracting the information by hand.

3.2.1. Analyzing the Alexa Dataset

Because we were interested in finding popular adopters of ECS, we looked into using the Alexa List [2] of the top 1 million second-level domains for our purpose, as was done by Otto et al.[37].

We use the publicly available database (April 20, 2013) as source to scan for possible ECS adopters.

Since one of the limitations of the ECS extension is that the protocol specifies no way to query a nameserver for ECS support we have to use an heuristic approach.

We select a random client IP, in our case the real subnet of our vantage point to avoid biased results, and re-send this ECS enabled DNS query with three different prefix lengths to the first authoritative nameserver of the domain, asking for the A records of the entry in the Alexa list. If the returned scope is non-zero for at least one of the replies, we annotate the server and hostname as ECS enabled.

After application of this algorithm to the list of second level domains we obtain two disjoint groups of (domain names, name-servers) pairs. The first group fully supports ECS and accounts for 3% of the second-level domain names. The second group, about 10%, is ECS enabled according to the IETF draft [21] but does not appear to use the additional section information for the tested domains, it may well just return a copy of the additional section with the scope always set to zero.

Thus, roughly 13% of the top 1 million domains may be ECS enabled. This number is only slightly larger than what was reported in a 2012 study [37]. However, some of the big players are among the ECS adopters, including Google (and YouTube), Edgecast, CacheFly, HiCloud, and applications hosted in the cloud such as MySqueezebox.

3.2.2. Estimated Impact of ECS: Analyzing a Packetlevel Trace

A second dataset is used to find adopters but also to estimate the potential traffic affected by ECS.

We use a 24 hour anonymized packet-level trace from a large European ISP. The trace is from a residential network with more than 10K active end-users and was gathered using Endace cards and analyzed with Bro [39]. It contains 20.3 million DNS requests for more than 450K unique hostnames and 83 million connections. While Alexa only includes the second-level domain names, this dataset allows us to identify full hostnames, which we use in a similar manner as above. In total, we find that roughly 30% of the traffic involves ECS adopters. This highlights that while the number of ECS adopters is relatively small (less than 3% of the authoritative name servers), it includes some of the relevant players responsible for a significant fraction of traffic.

3.2.3. Selected ECS Adopters

From the sets of identified ECS adopters, we select a large CDN, two smaller CDNs, and an application deployed in the cloud to explore their operational practices.

We found these to give interesting results after manual investigation of several candidates and in the rest of this thesis we will hence focus on:

Google as a founding member of the consortium "A Faster Internet" and one of the main supporters of ECS. It has adopted ECS in all their resolvers and name servers. Moreover, Google uses a sophisticated backend with many data centers, edge-servers, and Google Global Cache (GGC) servers located inside ISPs [3, 17, 23]. It is known that there can be anywhere between tens to thousands of Google servers behind a single Google IP [45] which makes an analysis of the user-to-server mapping and clustering of end-user networks especially compelling. Also Google makes heavy use of Anycast in the frontend as well as the backend.

Edgecast, a large CDN that also offers streaming solutions. It is also one of the participants in the "A Faster Internet" consortium.

CacheFly, another CDN that has adopted ECS. It is one of the first companies to offer CDN services and rely solely on TCP anycast for routing – rather than DNS based global load balancing.

MySqueezebox, a Logitech product that runs on top of Amazon's Web cloud Service EC2, confirmed by both the mapping to AS and the reverse DNS (amazonaws.com).

TorrentFreak, a blog/website dedicated to reporting the latest news and trends on the BitTorrent protocol and file sharing.

Chapter 4.

Methodology

In this chapter we are going to explain the framework we developed to execute our experiments on the ECS adopters. After an overview of the framework the different parts are shown and the tasks carried out by each module will be explained in detail.

The software presented is also available for download on our project website at:
`http://projects.net.t-labs.tu-berlin.de/projects/ecs-adopters/`

In Figure 4.1 we see our general setup. The framework in the center uses instances of workers to send ECS enabled DNS queries to the authoritative nameservers of the ECS adopters we explore. These workers can also be called on remote machines over an ssh connection using a wrapper script. The results of all DNS-queries made, including timestamps, are stored in an SQL-database. This database also holds all imported network prefixes, including the source and timestamp of the import. For evaluation the framework is capable of exporting various types of csv files, e. g., a list of all A-Records returned for all queries resulting from a specific import for a given worker.

4.1. Framework Overview

When we started the work on ECS and conducted the first evaluations, it got clear that for our measurements we would send many DNS-requests to a high count of servers using various different sources for both client prefixes and servers to query. Furthermore it was clear at a very early stage that we needed to repeat the measurements several times to compare changes and see developments and that these experiments would take several hours to complete.

This was the motivation to plan a framework that is able to track all of the various data sources and DNS-queries in an easy way and to automate as much as possible, including notification via e-mail in case an experiment stops unexpectedly or when the data is ready to be analyzed after a successful run.

Because of the number of queries and imports we estimated it was quite obvious that any approach utilizing flat files instead of a database would not scale.

The number of 50,953,750 DNS-responses, or 30.74GB, stored in only one of our database tables, illustrates this.

The following sections will first give an overview of the different parts of the framework and then explain their use in more detail. The different experiments we conducted, most of them using this framework, will then be explained in section 4.4.

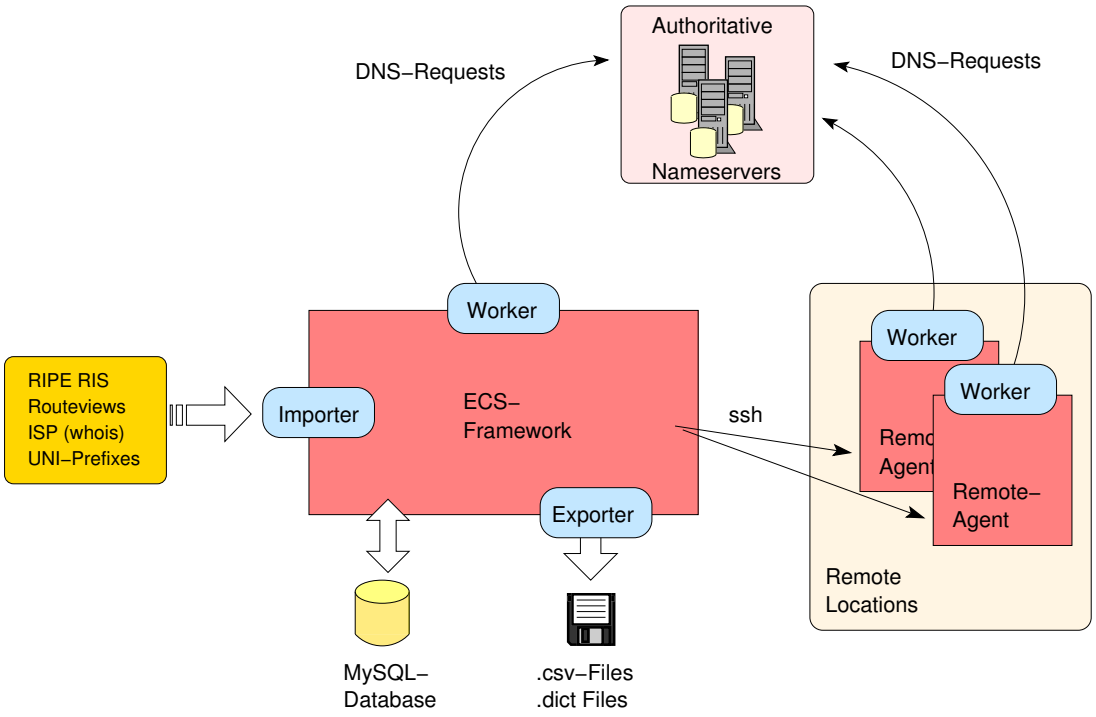


Figure 4.1.: Overview of the experiment setup.

4.2. Framework Elements

The framework, written in Python, consist of these different logical entities:

Database: Storage of imported network prefixes, client IPs and results of all DNS-request. Also holds most of the meta-information of the experiments like start and end time and a short name.

Worker: Execution of DNS-queries and storing of the results in the database.

Remote Worker: Execution of DNS-queries at a remote location, controlled via SSH.

4.2.1. Database

The Database, we used mySQL running on the measurement server itself, is used to store all imported datasets and all results from our experiments. The database scheme can be seen in the appendix in figure A.1.

The most important tables are the imports, rawdata, clientip and results tables.

While the imports table keeps track of where an prefix, stored as IP address and mask, stems from, the rawdata table contains these prefixes itself.

The clientip table is used during the experiments to keep track of which queries have to be made, which queries need to be repeated because they failed and which worker is assigned to execute the queries.

Finally the results table stores the results of all DNS queries we made during the experiments. The data we are interested in most, like the returned scope, is directly accessible in a database field, but the full results of the query are also stored as human readable text in a blob field for later reference.

After each experiment run the database was exported into an SQL file using the mysqldump command for backup.

4.2.2. Worker

The main task, executing DNS queries and saving them to the database, was performed by worker instances.

Each worker is configured with the IP address of a nameserver to query and a hostname to query for. At the beginning of each experiment a list of client IP prefixes to use was compiled for each worker and saved to the database. A worker would then go through this list and execute these queries. If a query fails this will be logged and the query will be repeated up to three times until it succeeds.

The workers are defined in the configuration file. Parts of this configuration is stored to the database on first use of the worker. This is done to ensure that a dump file of the database contains all information necessary to understand the experiments stored within.

A worker configuration consists of a descriptive name, the nameserver to query, the hostname to resolve and a description:

```
1 [worker_squeezebox]
2 id: 4
3 resolver: 79.125.21.137
4 hostname: www.mysqueezebox.com
5 description: Logitech Squeezebox
```

Several workers are defined in the configuration file, during an experiment these workers are executed in parallel, according to the experiment setup.

4.2.3. Remote Vantage Points

To estimate the possible bias of different vantage points on the results, we needed to find a way of executing the same ECS queries from different vantage points at once. The question we asked was if the IP-address of the connection also influences the results and not only the client-IP information submitted by means of ECS.

We found that it is not sufficient to run the same experiment without synchronization from other vantage points and combine the results and that the overhead of a full installation with mySQL and the required packages is not feasible on remote locations operated by third parties. Thus we implemented a solution with minimal requirements, especially without the need for root access to these vantage points.

The configuration of such an ssh-enabled vantage point looks as follows:

```
[vpoint_ssh_xxx.xx.edu]
id: 10
description: xx.edu worker on xxx.xx.edu
type: ssh
host: xxx.xx.edu
user: fls
pass: xxxxxxx
runcmd: /home/guest/fls/fls-edns/remoteclient.py
```

The framework can be extended easily to support other ways of connecting to vantage points, for example via web-technologies or unencrypted telnet connections offering more throughput.

Another type of 'remote' vantage point we implemented is the 'local' vantage point that will execute on the measurement machine but internally deliver results in JavaScript Object Notation (JSON) format as well. This made it possible for the synchronized experiment scheduler (see section 4.4.4) to use a common API internally to access the workers.

4.3. Framework Operation

After a high-level view on the different functional units of our framework we will now describe the different submodules implemented in detail to understand how these can be orchestrated to execute our experiments.

We decided to automate as much as possible in our experiments to be able to reproduce the experiments with minimal manual intervention, which guarantees us comparable results from run to run.

The different submodules share one common configuration file. Thus it is possible to set, e. g., the loglevel for the logging facility for all modules at once. An example configuration can be found in the appendix A.4.

The framework consist of these submodules:

Data Import: Create entries in the database for client-IPs to use in queries, based on various inputs.

Data Export: Export data, mostly A-Records and ECS scopes as csv-Files and generate dict files.

Database Access: Provide information on the database currently in use, e. g., usage statistics, or initialize tables of an empty database.

Global Configuration: Access to configuration settings, setting of internal parameters, e. g., logging, based on the configuration file.

E-mail Notification: Notification of errors and progress of the experiments via e-mail.

Analyzed Output: Generation of output files for histogram plots of scopes and prefix lengths.

Random Sampling: Balanced selection of imported data for down-scaled experiments.

Worker Preparation: Preparation of unique lists of client-IPs for a worker at experiment start and worker execution.

Back to Back: Execution of DNS queries by the workers with a modified schedule.

Synchronized Workers: Synchronized execution of workers on different servers/vantage points.

4.3.1. Data Import

The framework is able to import various data as source for the DNS-queries/client IP Information used.

The compressed files provided by RIPE/Routeviews are read by the bgpdump utility, the output is parsed and all prefixes are recorded in our database together with the source of the prefix - usually the input file name for bgpdump. Also the timestamp is recorded. The whole process of downloading the files from the respective websites and importing them with correct timestamps and source id was automated using a shellscript.

The import module also allows individual IP addresses to be imported, e.g. the list of popular DNS resolvers or the creation of individual addresses by expanding networks by prefix and netmask.

4.3.2. Data Export

The framework allows to export the data collected during the experiments based on the workers used, the experiment and the datasources (imports).

The output files contain the A records or the scope returned for each client-IP/prefix used in csv format. In addition, for each exported .csv-File a dict-file is written, explaining the arguments used for the export, the inputs filtered on, etc.

This functionality was used to further analyze our data with statistic tools like R/Splus and to generate the various statistics using custom Python and shellscripts.

4.3.3. Experiment Preparation

An important role plays the phase between the import of new data and the start of the workers. In this preparation phase, which can take up to several hours, the client-IP prefixes that will be used in the queries are prepared for each worker.

In this phase all Client-IP-Prefixes from the various data sources are copied into a new table, but only if the same prefix is not yet present in the table. This way a unique set of prefixes to ask is built decreasing the number of queries dramatically. Also counters are set up for each prefix to allow efficient retries of queries that time out.

4.3.4. E-mail Notification

Because we knew that the experiments will run for several hours and we do not want to lose time if the execution stops due to errors, e. g., a filled harddrive or because we got blacklisted on remote servers, we implemented a notification mechanism using e-mail to get informed on the status of the experiments.

At each step of the experiments, that is data import, preparation of the client IP table and start of the workers, the framework sends an e-mail directly using the Simple Mail Transfer Protocol

(SMTP). This approach was chosen because we wanted to make sure that the framework can operate independently of the configuration of the underlying server, and without the need for root access.

In addition the framework sends an e-mail whenever an exception in the code is thrown that is not caught, unlike a failing DNS query that resulted in a retry.

4.3.5. Analysis

The analysis module is able to generate data files for heatmap creation, specifically of the prefix lengths versus ECS scope returned.

All other analysis was made using external applications, based on the exported csv-files.

It was planned to integrate the functionality, including plot creation, into the framework but we decided to drop these features as it would have meant to run these while experiments are collecting new data.

4.3.6. Random Sample Creation

To take snapshots of the user-to-server mappings of the various CDNs and to gather time series it can be desirable to be able to run experiments with a fixed number of client IP addresses, that means with a pre-defined number of DNS requests.

Our framework makes sure that the distribution of datasources in the samples is the same compared to the original data. If 10%of client prefixes comes from dataset A in the random sample again 10%will stem from that datasource.

The implementation of this functionality was needed because the size of the different datasources varies heavily and a purely random selection would be heavily biased by this.

This module creates a new 'imported' dataset based on existing imports, making it possible to run down-sized experiments very easily and scale the size of the experiments if desired just by using the designated import ID on experiment execution. The datasource can easily be found using SQL by mapping the IP-address to the according tables.

4.4. Experiments

The following sections give an overview on the execution of the experiments, after the last sections explained the framework and the various submodules.

4.4.1. Experiment Layout

Each experiment conducted was given a name in the database and timestamps were recorded at experiment start and termination.

A full experiment consists of these steps:

- Creation of a new experiment entry in the database
- Import of data (Routeviews, RIPE, ISP, . . .)
- Preparation of the experiment
- Parallel execution of workers
- Backup of the database
- Final e-mail notification

These steps are executed by a shell script written specifically for each experiment run, orchestrating the call of the various submodules of the framework.

This makes it possible to react very flexibly to changing requirements and also serves as a documentation on the exact steps involved with each experiment conducted.

4.4.2. Experiment Execution

During execution phase, each worker selects the client IP information and prefixlength from his own list of distinct prefixes, where the retry count is below a set limit, until all prefixes have been queried successfully or the retry count is above the threshold.

The worker will send a DNS request to the specific nameserver configured for the worker and query for the name set in the configuration. All returned data is then parsed and stored in the database, including a timestamp.

The parsed data specifically consists of the network prefix, mask, ECS scope, query, A records returned. All data,also including additional records, DNS-query id, etc, are stored in a text blob for further analysis and as a fallback.

4.4.3. Back to Back Tests

This module uses a different schedule in the workers when executing queries. Specifically it repeats every query several times within a few seconds to detect different answers for the same query within the Time To Live (TTL) of a record. As this dramatically increases the number of total queries and thus the time needed to query all prefixes, only a fixed number of prefixes are selected using the Random Sample module described in the last paragraph.

4.4.4. Remote Workers: Synchronizing Vantage Points

This submodule again uses a different schedule for the queries by executing them in parallel from different servers to better understand the impact of different vantage points on the results.

For this the concept of a remote-worker is introduced. This worker will execute all DNS requests synchronized on each configured vantage point.

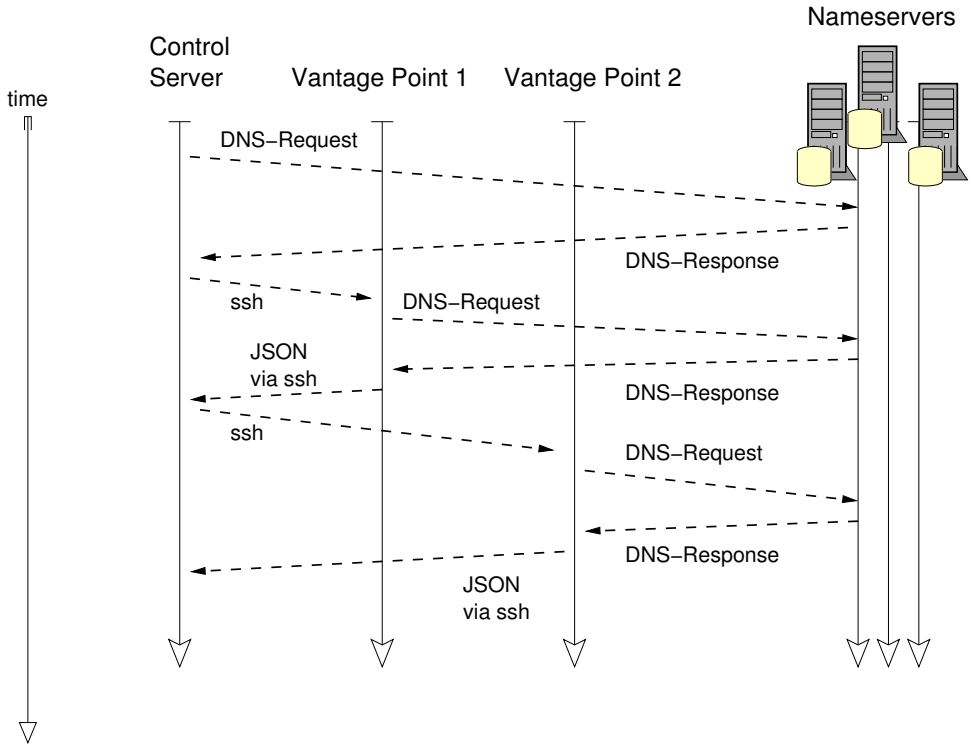


Figure 4.2.: Remote Execution: Time Sequence Diagram

A diagram showing the timing can be seen in figure 4.2. The framework on the controlling machine first opens an SSH connection to each vantage point and starts a wrapper script with the nameserver and the specific hostname to query as command line arguments.

For each prefix in the prepared list of queries the controller will send the tuple of prefix and prefixlength to one vantage point after the other. Upon receiving the data the remote script will immediately execute the DNS-request and report the result of the query in JSON format on stdout. There the data is captured over the SSH connection and stored in the database.

If all vantage points report a successful query the prefix is marked as completed and the next prefix is selected from the database, thus only complete datasets will be marked as done. Again we retry each prefix three times. We observed that all queries will be usually executed within the same second, on average we see 3.5 prefixes queried per second.

4.5. Lessons Learned

During our work we encountered various difficulties, of which only some are of technical nature. We want to take the chance documenting some of these challenges and provide insight into some of the design decisions made and give hints for further improvements of our framework. While starting off with some of the expected technical issues we want to emphasize that we had to also deal with legal and political issues.

4.5.1. Nameservers Blocking Responses

Besides all automation and failure tolerance implemented, we missed one obvious case. The case when one of the resolvers queried gets completely unavailable or is blocking our requests because the measurements are detected as denial of service attack.

Because we knew that DNS queries via UDP would be lost, we implemented a retry up to three times for each request, which would not happen immediately but around 1000 queries later, and because we also knew that these errors can occur in clusters, we did not enable e-mail notification for this case. But we missed detection of a complete server outage or block, so each query was still repeated three times, which led to unnecessary traffic towards that server and some delay in the overall experiment until we noticed that behaviour. Also we missed the implementation of a back-off time, pausing requests to a server after an amount of failed requests before disabling it completely.

We observed such behaviour at one of the ECS adopters in two experiments, so we had to remove that one from the final result-set.

4.5.2. Backend Changes

One thing to keep in mind when doing active measurements on infrastructure controlled by third parties is that this infrastructure can change, especially if the operators of that infrastructure are not aware of these measurements taking place, or want to prevent them. This meant that the object being observed can change while we measure - in contrast to a simulation. And it is not always clear what causes such change - an operational intervention or the measurements conducted?

When we compared the results from some of the synchronized queries we made from different vantage points, we found one phenomenon that we, at first, could not explain. While we had manually checked that the results are not biased from using different vantage points, the results of the checks run when using the full datasets revealed something else. After around 100k queries that have by chance been made in order of the IP prefix, the results from the vantage point in the US started to differ from the results in Germany.

We have two possible explanations. The first, and most likely is, that the databases the nameservers used in the backend had been updated only in one region, e. g., the US, and not in Europe. The other could be that after we presented some preliminary findings publicly at a conference, the engineers of that specific CDN implemented countermeasures that to some extent stop our experiments to unveil too much information about the backend databases.

On the other hand sometimes less “harmful” changes that even give more measurement opportunities can occur as well - which is why we would anybody advise to analyze such changes carefully:

We noticed a change how EDNS packets sent to the public DNS service provided by Google are handled from the day DNSSEC support was enabled. We noticed that the response packets suddenly also contained EDNS data and we found that the response messages had a scope set and the returned A records for Google servers were identical when sending these ECS enabled queries to the authoritative servers.

That means that the activation of Domain Name System Security Extensions (DNSSEC) also enabled the forwarding of our ECS information to third party resolvers via the Google public nameservers - and we are able to use this to query other CDNs that for example only accept ECS queries from Google.

4.5.3. Efficient Caching of Cymru Data

While using the mapping service of Cymru[1] to map all client-IP prefixes and the addresses seen in the A-records to AS numbers and geographic location we experienced yet another challenge.

We had various sets of A-records and client prefixes from the various experiments conducted, most of them overlapping as the experiments had been repeated with updated data.

To reduce query time and the number of total queries to sent to Cymru (it's a free service and we neither want to overload the servers nor get banned for too many queries) we decided to build a set of unique IPs/prefixes. This set had in total over one million entries which of course also resulted in an output file of over one million results from their service, obtained via the telnet interface provided on port 43.

We then used the standard grep utility from our operating system and supplied a file with a list of input data we are interested in, using the command line switch -f – which reads patterns from a file, line by line.

We observed that a filter run for around 20k lines of patterns took more than 120 minutes. It was clear that this was not a usable solution - as we had to repeat this with every dataset we are interested in.

The solution here again was not using the simple approach using standard command line tools but to write two Python scripts and a small database scheme.

Here we again used search indices on the IP address field and interpreted the IPv4 addresses in the datasets as Integers - which resulted in a huge performance gain.

The import of all results took around 13 seconds, one filter run now took around 8 seconds instead of over 120 minutes.

4.5.4. Robust Design: Redundant Data in the Database

One bug that was present until the end of the experiment nearly ruined the effort completely. Because of a missing assignment, the A records received in the DNS requests from the queried nameservers was never written to the designated datafield. Luckily this extra datafield was only present for easier access to the returned data and the same data was also present in a second datafield in the database, where the whole DNS response was stored as human readable text in a blob-field. This made it necessary to parse this field in all our analysis, but we did not notice a loss in performance. Of course we fixed this bug during the ongoing experiment but we never used the designated field, because parsing all results and writing back the results took longer

[1] `http://whois.cymru.com`

than expected due to hardware limitations. The process was not finished although running for several days and on the other hand we did not want to risk biased results due to newly introduced bugs when now using the other datafield.

4.5.5. Database: Drawbacks

While the last sections explained the benefits of using a database, there are also some hidden pitfalls that can come along when using a database naively.

The most crucial part is the design of the database scheme. In the appendix (see Fig. A.1) the schema we implemented can be seen. The choice of datatypes is important. For example we chose to store IP-addresses as integers and network prefixes as one integer containing the network base address and a short integer storing the mask to apply. This was possible because we only considered IPv4 addresses in out experiments but it enabled very fast searches on the datasets.

Besides organizing the data efficiently in the database another important role, when it comes to speed, is the correct use of indexes in the database tables.

We knew that for our experiments we would need to search for IP-addresses and network prefixes, thus several indices have been put in place to allow for fast table joins when filtering the results based on the different imports we used.

To give an example: When we are only interested in the results of the RIPE datasets we need to filter all imports by the datasource id to match RIPE, then join the client IP prefixes with the results table and extract the A records.

With the correct indices in place this is a matter of about half an hour - without, the database has to do linear searches.

We had the chance to observe the effect when we started our first experiment and forgot one index.

4.5.6. Monitoring of Experiments

We like to emphasize that however sophisticated a test framework can be, sometimes the most basic piece needs monitoring and that a system that is meant as proof-of-concept implementation in a scientific environment cannot be compared to the requirements of an industry application with multiple levels of quality assurance and full time operators responsible for the systems and monitoring.

We had two different occasions where side effects of the actual experiments produced enormous amounts of data.

The first incident happened when we ran the synchronized queries on remote vantage points. From the test runs the configuration file still enabled file logging at the finest level. The result was a single logfile nearly filling up the disk partition which would have ended in an incomplete experiment. We had to stop the experiment, change the configuration and continue the experiment at the point where it stopped. Luckily our design allowed us to do so.

The second incident resulted in more than 130 GB of disk space blocked unnecessarily by mySQL. The script mentioned in section 4.5.4 should have enabled autocommit on the SQL-connection to the database, but for an unknown reason failed to do so – although the command was present in the code it had no effect. This resulted in all database write operations being only temporarily written to the database, waiting for a commit command by the application. When the script was manually stopped things got worse, because i) a rollback operation started, using most of the IO bandwidth of the harddisks and ii) the disk space blocked by the `ibdata1`-file to store the uncommitted requests was not freed, not even after the rollback was complete. We learned that this behaviour is by design and can only be changed by exporting all databases from the server, activating the setting `innodb_file_per_table` and re-importing all databases from dumpfiles.

We estimated this to last several days from the experience we had with importing our database on other machines to do analysis and thus kept the setting disabled, after deleting all local backups to free diskspace.

4.5.7. Packetlevel Traces: Drawbacks of Limited Sample Size

Another limitation was found with a dataset, from which we had hoped to extract usage information on the ECS extension and ECS adopters. We learned that both is not feasible due to technical limitations.

The dataset consists of anonymized sFlow samples. This technique uses packet sampling and is mostly used in switched high bandwidth networks where capturing all traffic is not feasible due to the amount of data. Moreover each sFlow sample from the dataset only contains a limited amount of the original data packet's payload. We found two problems in using this kind of datasets, rendering them useless for our specific purpose of analyzing ECS:

(i) The sFlow data only contains statistical samples and not all packets seen on the wire. This means all results can also only be interpreted with statistical means.

(ii) Because only a limited number of bytes at the start of the payload is available the Additional Section, which holds the data we are interested in and is located at the end of the DNS packets (see Fig. 2.4), is not part of the samples.

4.5.8. Abandoned Approach: Full TLD Scan

A question we strived to answer was how many domains in the Internet are ECS enabled, or how many nameservers respond to such queries. For this we first considered a straight forward approach: just ask, using the DNS.

While access to the full Zonefile of the .com and .net TLD is possible under a non disclosure agreement, this is not possible for the .de-Zone managed by denic for legal/political reasons.

With this first limitation in mind we evaluated the complexity of a full scan of all .net and .com nameservers and discarded this approach quickly, also because the results would not have been meaningful while a lot of resources would have been bound. We can see this at the Alexa

experiment, explained in section 3.2.1 - because there is no definite way of determining ECS support in a nameserver, the results would have been very vague.

It's been an interesting take away to see that not only technical limitations limit the ability to do research but often organizational or legal issues as well, or the combination of both.

Chapter 5.

Evaluation

To evaluate the capabilities of ECS as a measurement tool, we explore how difficult it is to

(i) uncover the footprint of ECS adopters,

(ii) assess the effect of ECS on cacheability of DNS records,

(iii) capture snapshots of how ECS adopters assign users to server locations.

For each of the ECS adopters we analyze, we select a fixed pair of: hostname to resolve and nameserver to query. For Google these are the hostname `www.google.com` and one of the authoritative name servers e. g., `ns1.google.com`. See appendix A.3 for a full list of adopters, hostnames and nameservers selected.

While for the RIPE, RV, ISP and ISP24 dataset we use the prefixes as announced, for the UNI dataset an ECS prefix length of 32 is chosen, as this dataset contains individual IP addresses.

We periodically collect traces until mid of August 2013. Some of the datasets are incomplete and thus are omitted in the evaluation.

5.1. Uncovering Infrastructure Footprints

We first report on our experiences using ECS to uncover the footprint of four selected ECS adopters. The operational community apparently also did some investigation [5] in enumerating CDN servers using ECS.

For this purpose we analyze the A-records that are returned in the DNS answers we receive from the authoritative nameservers of the ECS adopters. These A-records contain the IP addresses of the servers delivering the service of the ECS adopters, in our case most of the time (at least) serving a website. We then use the whois service of Cymru[1] to map these IP-addresses to announced subnets and the corresponding ASes.

Table 5.1 summarizes the number of unique server IPs, subnets, ASes, and locations. The footprint of Google is by far the most interesting, with more than 6,300 server IPs across 166 ASes in 47 countries.

For geolocation we use the service of MaxMind [6]. We are aware that geolocation of CDN servers can be inaccurate, e. g., MaxMind maps the IPs of the main Google AS (AS15169) to

[1] `http://whois.cymru.com`

	Prefix set	Server IPs	Sub nets	ASes	Countries
Google (03/26/13)	RIPE	6,340	329	166	47
	RV	6,308	328	166	47
	PRES	6,088	313	159	46
	ISP	207	28	1	1
	ISP24	535	44	2	2
	UNI	123	13	1	1
MySqueezebox (03/26/13)	ALL \ UNI	10	7	2	2
	UNI	6	4	1	1
Edgecast (04/21/13)	RIPE/RV/PRES	4	4	1	2
	ISP/ISP24/UNI	1	1	1	1
CacheFly (04/21/13)	RIPE/RV	18	18	10	10
	PRES	21	21	11	11
	ISP	6	6	5	5
	ISP24	5	5	4	4
	UNI	1	1	1	1

Table 5.1.: Selected ECS adopters: Uncovered footprint.

California, but it is accurate on the country level for IPs that belong to ISPs [41] and thus good enough for the purpose of this study. A more sophisticated approach of geolocating Google server IPs in the Google AS is presented in the work of Calder et al. [18].

We also notice that GGCs are typically located in ASes that are "categorized" as enterprise customers and small transit providers [22] in both developed and developing countries. In March 2013, Google servers are found in 81 enterprise customers, 62 small transit providers, 14 content/access/hosting providers, and only 4 large transit providers (a small number of ASes that host Google server IPs do not belong to any of these categories). While illustrative and also informative (e. g., we uncover more locations than previously reported [14, 45]), the main and more surprising finding is the simplicity with which we can uncover this infrastructure using ECS from a single vantage point in less than 4 hours.

For validation purposes, we check each server IP—all of them serve us the Google search main page. In addition, the reverse look-up reveals that while all servers inside the official Google AS use the second level domain 1e100.net [27], those deployed in third-party ASes use different hostnames (e. g., cache.google.com, or names containing the strings ggc or googlevideo.com). In some cases we observe legacy names that indicate the prior use of the IP range by the ISP.

We want to emphasize that therefore the presence of a GGC cannot be inferred by looking at the reverse DNS zones without performing active measurements.

5.1.1. Comparing Results Using Different Prefix Sets

Both the RIPE as well as the RV prefix sets are sufficiently complete to yield the same results with respect to the uncovered footprints of our ECS adopters.

We attribute this to the fact that the publicly advertised address space collected in both datasets overlaps significantly. We see around 500K announced prefixes in the data sets at various aggregation levels. Using only the most specifics without overlap this number reduces to about 130K prefixes. For our experiments we decided to use the prefixes as announced. We think this corresponds to the distribution to be seen at an ECS enabled nameserver[2] and reflects the public IP-address space being used[3].

Next we compare our results (RIPE only) to a study of Calder et al.[18], where queries were made using /24 prefixes. When we compare the raw data, a list of IP addresses found on August 8th, we see a 94% overlap in the discovered Google server IP-addresses while issuing significantly less DNS queries in our approach. This indicates that the limitations coming from using only publicly announced networks are irrelevant regarding our experiments.

PRES however is not sufficient to uncover the full set of Google Web servers, but still yields a major fraction of them in only 55 minutes per experiment. Alternatively, one can use a subset of the RIPE/RV prefix sets. Using a random prefix from each AS reduces the number of RIPE/RV prefixes to 43,400 (8.8% of RIPE prefixes) and results in 4,120 server IPs in 130 ASes and 40 countries in 18 minutes (with 40 requests/second). By doubling the number of selected prefixes to two per AS, we uncover 4,580 server IPs in 143 ASes, and 44 countries.

When relying on the ISP, ISP24, and UNI data sets, we see the effect of mapping end-users to server IPs using ECS, in the case of Google we uncover a much smaller number of servers.

However, by using the de-aggregated prefix set of the ISP (i. e., ISP24), we are able to expand the coverage from 200 to more than 500 server IPs. More than 95% of them are in the Google AS while the rest is located in a neighbor AS to that ISP. A more careful investigation reveals that the client prefixes served from the neighbor AS are from a customer of this ISP whose prefix is not announced separately but only in aggregated form (i. e., together with other prefixes of the ISP). Our conjecture is that this is due to the BGP feed sent to the GGC by the ISP [17].

Of the ASes uncovered by using the RIPE prefix set, only 845 and 96 server IPs are in the ASes of Google and YouTube, respectively. All the others IPs are in ASes not associated with Google. This shows the profound effect of GGCs which have been deployed to many ASes. We repeat the experiments by using the Google Public DNS server and observe that the returned answers are almost always identical (99%). This is not necessarily the case when using Google's Public DNS server for other lookups. However, we find Google's Public DNS server forwarding our ECS queries unmodified to white-listed authoritative DNS servers of other ECS adopters. Therefore, we can even (ab)use Google's public DNS server as intermediary for measurement queries and thus (i) hide from discovery or (ii) explore if these ECS adopters use a different clustering for Google customers.

[2] A study on this is currently being performed.
[3] See Section 3.1 for known limitations.

	Google				Youtube			
Date (RIPE)	IPs	Sub nets	ASes	Countries	IPs	Sub nets	ASes	Countries
2013-03-26	6340	329	166	47				
2013-03-30	6495	332	167	47				
2013-04-13	6821	331	167	46				
2013-04-21	7162	346	169	46				
2013-05-16	9762	485	287	55	8124	393	220	49
2013-05-26	9465	471	281	52	8003	385	221	47
2013-06-18	14418	703	454	91	11527	546	337	85
2013-07-13	21321	1040	714	91	17638	842	558	122
2013-08-08	21862	1083	761	123	18069	875	598	112

Table 5.2.: Google (YouTube) growth within five (three) months.

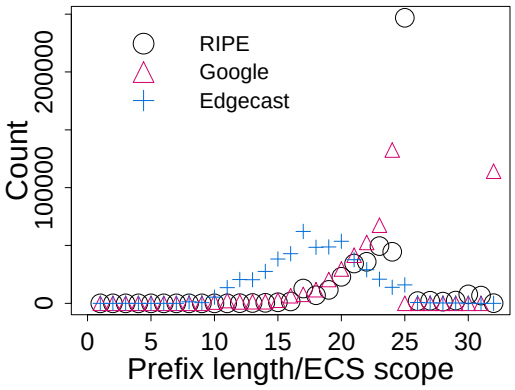
Table 5.1 also shows that the footprints of the other ECS adopters are "less" interesting, mainly because their footprint is not as widely distributed compared to Google. Nevertheless, we see in principle similar results. Most of the infrastructure can be uncovered with the RIPE/RV/PRES prefix sets. The ECS adopters again use clustering such that the ISP, ISP24, and UNI prefixes are all mapped to a single server IP. Note that Edgecast may use HTTP-based redirection which cannot be uncovered using only DNS. While Edgecast uses a single AS, CacheFly, and MySqueezebox are utilizing infrastructures across multiple ASes. We also observe that both players map the UNI and ISP/ISP24 prefixes to infrastructures in Europe (e. g., MySqueezebox maps them to the European facility of Amazon EC2).

5.1.2. Tracking the Expansion of CDNs Footprints

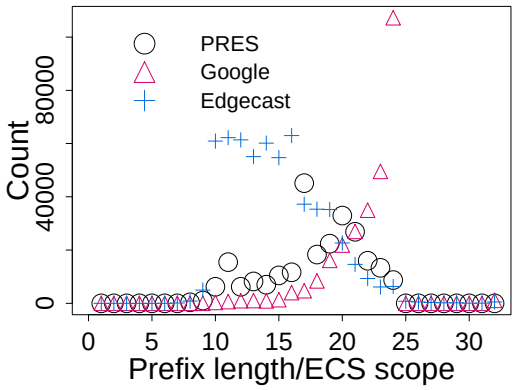
Our method allows us to track the expansion of ECS adopters' footprints over time. This becomes increasingly important as many CDNs continuously deploy servers at the network edges or within ISPs. Thus, one can not infer the operator of a cache by simply looking at the IP address or AS number [19].

As we show above, RIPE and RV public prefix sets uncover by far more IPs than the other prefix sets. We use the RIPE prefix set to track the expansion of ECS adopters as it is updated more frequently than RV. In Table 5.2 we report the rapid increase of discovered Google server IPs over a four month period (March-August 2013) and the growth of the YouTube frontend IPs over a three month period (May-August 2013). We observe that the number of Google server IPs at least triples (345%), the number of ASes hosting Google infrastructure increases by 595 (458%) and the global presence at least doubles (261%).

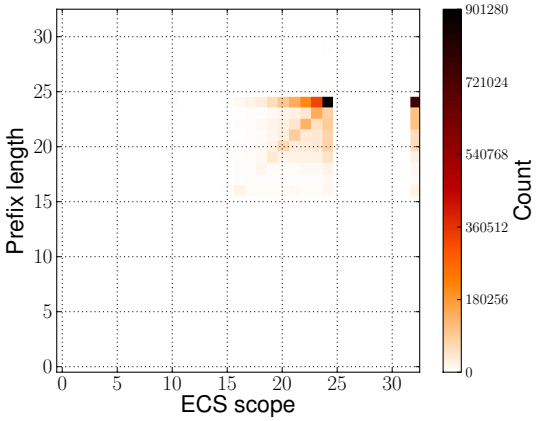
In August 2013, Google servers are found in 372 enterprise networks, 224 small transit providers, 102 content/access/hosting providers, and 11 large transit providers. Starting mid



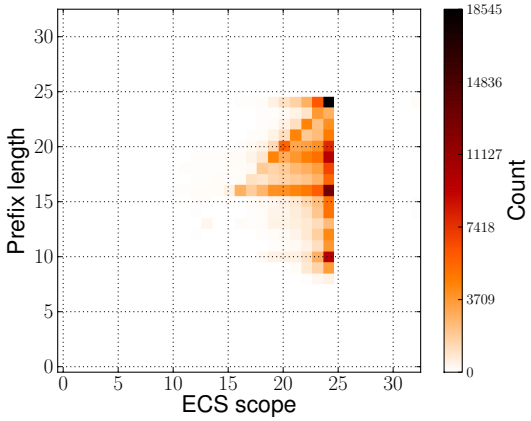
(a) RIPE



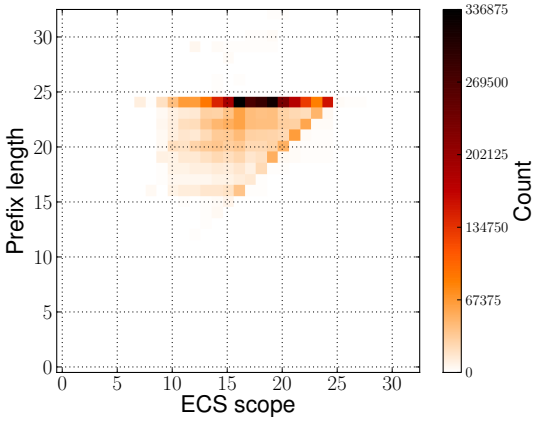
(b) PRES



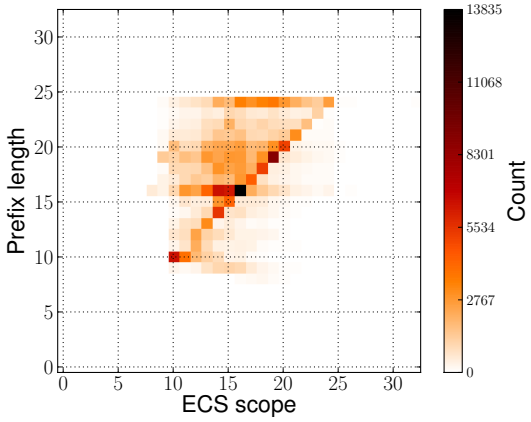
(c) Google (RIPE)



(d) Google (PRES)



(e) Edgecast (RIPE)



(f) Edgecast (PRES)

Figure 5.1.: Prefix length vs. ECS scope for RIPE and PRES (Google: March 2013, Edgecast: May 2013).

May we include the YouTube website in our measurements and notice that while the number of ASes with YouTube servers almost triples (271%), these ASes overlap with already uncovered Google infrastructure. We figure this being a result of incorporating the YouTube infrastructure into Google's global platform. By merging the sets of IP addresses for Google and YouTube, the count only increases to 24048. For an in-depth study of the expansion of Google infrastructure since November 2012 we refer to Calder et al. [18]. Our study did not uncover significant growth of serving infrastructure for the other ECS adopters.

5.2. Uncovering DNS Cacheability

Next, we examine the ECS specific information included in the DNS response: The scope. In principle, if the ECS information corresponds to a publicly announced prefix, one may expect that the returned scope is equal to the prefix length. However, this is not necessarily the case. Content providers often return either coarser- or finer-grained scopes e. g., they respond either with aggregated or de-aggregated prefixes. This indicates that they perform the end-user clustering for client-to-server assignment on a different granularity than the routing announcements.

We note that the scope may have a major effect on the re-usability of the DNS response i. e., the cacheability of DNS responses. While most of the responses have a non-zero TTL, a surprisingly large number has a /32 scope. An ECS scope of /32 implies that the answer is valid only for the specific client IP which issued the DNS request. In this section, we explore DNS cacheability for two ECS adopters in detail: Google and Edgecast. The others are less interesting as CacheFly always uses a /24 scope and MySqueezebox is similar to Edgecast.

Figure 5.1(a) shows the RIPE prefix length distribution (circles). In addition, it includes the returned scopes from using the RIPE prefixes to query our ECS adopters. We note that the distributions vary significantly. There is massive de-aggregation by Google but massive aggregation by Edgecast which operates a smaller infrastructure.

When executing back-to-back measurements for Google (e. g., 4 queries within a second), we find that typically both the answer and scopes are consistent within the duration of the TTL (300 seconds for Google). The answer as well as the scope can change in some cases within seconds[4]. A detailed study of the temporal changes of the returned scope is part of our future work. Overall, as seen in Figure 5.1(a), for almost a quarter of the queries, the returned scope is 32. This indicates that currently Google severely restricts the cacheability of ECS responses or may want to restrict reuse of the answers to single client IPs. For approximately 27% of the queries prefix length and scope are identical. For 41% of the queries we see de-aggregation while there is aggregation for 31%. The difference between the announced prefix and the returned scope may also be due to the the BGP feed sent to the GGC by the ISP [17] that is not necessarily the same that is publicly announced and collected by RIPE or Routeviews. We again note, that the returned scopes for the RIPE and RV prefixes are almost identical.

Exploring the scopes returned by Edgecast may at first glance appear useless, because they only returned a single IP with a TTL of 180 seconds. However, Edgecast is using significant

[4]This can be attributed to the fact that authoritative nameservers may use anycast or load balancing [13] or because the rapid increase of DNS queries (e. g., from our measurements) triggers a change on IP/prefix to server mapping.

aggregation for all prefix lengths across all prefix sets. For example, when using the RIPE prefixes, the return scope is identical for 10.5% but less specific for 87%.

When using the ISP prefix set, the overall picture is similar even though the specific numbers vary. An initial study of the prefixes with scope 32 indicates that Google performs profiling, e. g., Google returns scope /32 for all CDN servers of a large CDN provider inside the ISP. In future work, we plan to explore if there exists a natural clustering for those responses with scope /32.

Given that the size of the UNI prefix set is limited, we issue queries from all IP addresses with prefix length 32. The returned scopes vary heavily from /32 to /15, even for neighboring IP addresses. In the August measurements this behaviour is not found and we consider this a side effect of e. g., a loadbalancing mechanism that distributes the queries to different backend servers with different database revisions.

For the PRES prefixes, Figure 5.1(b) shows extreme de-aggregation. For more than 74% of the prefixes, the scope is more restrictive than the prefix length, and in 17% they are identical. Only few returned scopes are /32s. This may indicate that Google treats popular resolvers differently than random IP addresses. Google may already be aware of the problem regarding caching DNS answers as discussed in Section 2.3.3. For Edgecast we see significant aggregation.

To highlight the relationship of prefix length in the query to scope in the reply, Figures 5.1(c) and 5.1(d) show heatmaps of the corresponding two-dimensional histograms. For the RIPE dataset we notice the two extreme points at scopes /24 and /32. For the PRES dataset, the heatmap highlights the de-aggregation. Figures 5.1(e) and 5.1(f) show the heatmaps for Edgecast. While for the RIPE dataset we see the effect of the extreme prefix de-aggregation for Google very clearly, the picture for Edgecast is more complicated as there is mainly aggregation. For the PRES dataset, the heatmap shows even more diversity as there is de-aggregation as well as aggregation. This results in a blob in the middle of the heatmap.

5.3. User To Server Mapping Snapshots

So far we have not yet taken advantage of the Web server IP addresses in the DNS replies in correlation with the client prefixes or client IPs. Establishing this relationship allows us to capture snapshots of the user-to-server mapping employed by an ECS enabled CDN or CP that can be used to shed light on CDN mapping strategies. In the following, we illustrate the measurement capabilities offered by ECS. We explore snapshots of Google's user-to-server mappings (based on the RIPE data set) and examine how stable this mapping is.

Google returns 5 to 16 different IP addresses in each reply. Almost all responses (>90%) include either 5 or 6 different IP addresses. We do not find any correlation between the ECS prefix length or the returned scope and the number of returned IP addresses. All IP addresses from a single response always belong to the same /24 subnet (the returned IPs are not necessarily in close geographic distance to each other [24]). We also notice that typically the announced prefix length of subnets that host Google servers is /24. Thus, based on a single ECS lookup per prefix, we always find a unique mapping between query prefix and the server subnet from the DNS reply.

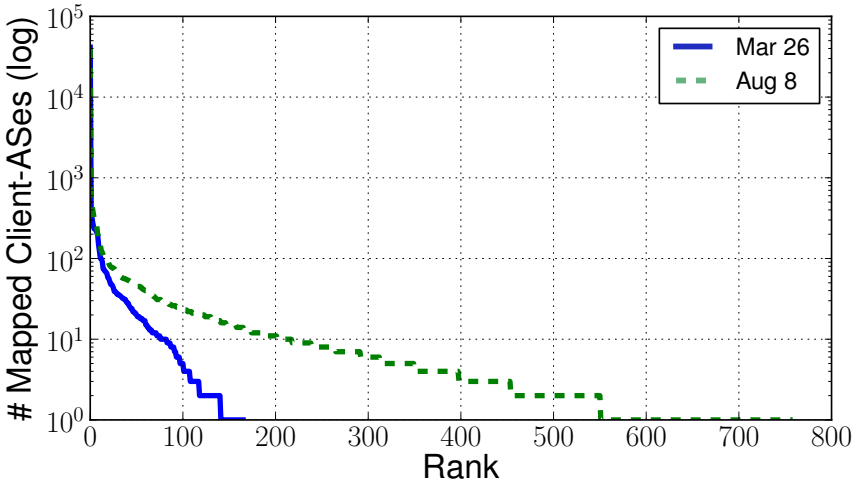


Figure 5.2.: Google: User to server mapping (RIPE).

Next we assess the mapping consistency at AS-level.

First we map all prefixes used in the ECS queries to their corresponding AS. Then, by looking at the returned A records, we find the corresponding server ASes for each client AS.

On March 26 2013, the majority of client ASes, around 41K, was served exclusively by Google servers from a single AS. About 2K ASes were served by servers in 2 ASes, and less than 100 ASes were served by servers from more than 5 ASes. On August 8, 2013 the number of ASes that served by a single AS dropped to around 38.5K and around 5K served by 2 ASes. ASes served by a large number of server ASes typically have a global footprint. We find that client prefixes of ASes that host GGC are also served by servers in other ASes. This is to be expected as GGC capacity may not always be sufficient to handle demand and also because different prefixes within an AS, e. g., those that host the GGC servers, may be handled differently.

As illustrated in Figure 5.2 a small number of ASes hosts servers that serve a large number of ASes. By far the most popular AS is the official Google AS (AS15169) that served more than 41.5K ASes in March and around 40.5K in August 2013. In the top-10 we find the YouTube AS, as well as small and large transit providers that serve their customers. There is also a small number of ASes that exclusively serve their client subnets from GGC servers they host.

From our analysis we derive some important observations. First, the Google content is not any more exclusively served by servers in Google ASes, as was reported in a study of Ager et al. [14]. Second, GGCs have been enabled in a significant number of ASes over a five months period. Third, within these five months the number of ASes that are served by GGC servers in other ASes has increased significantly.

This trend has significant implications for caching as Google content can be available in the same or a neighboring ASes. It also has implication for peering as the presence of GGCs reduces the inter-domain Google traffic and it is now possible for smaller networks to reduce their transit cost by either install GGCs or peer with ASes that host GGCs.

To assess the stability of user-server mapping over time, we analyze the returned IP addresses when asking back-to-back queries over two days (May 3-4, 2013). We found that around 35% of the prefixes are always served by a single /24 block over the 48 hours period. Given the highly distributed infrastructure of Google one may have expected larger churn. 44% of the query prefixes are mapped to two /24 and a very small percentage to more than five /24s. A possible explanation for this stable mapping is that Google uses local load-balancers [31, 45]. We leave the study of temporal dynamics in user-to-server mapping over longer periods and during flash crowds or other events as future work. Our future research agenda also includes the study of the temporal changes in user-to-server mapping not just for Google but also for other ECS adopters.

Chapter 6.

Conclusion

6.1. Summary

In this thesis we show, that the introduction of the protocol extension Client Subnet Extension (ECS) for DNS is very well suitable to conduct active measurements, revealing details on the infrastructure of the adopters of this new protocol extension.

Large CDNs and CPs rely on inference of the location of their users to assign them to appropriate servers, often relying on the source of the DNS-requests seen at their authoritative resolvers. The basic approach here is that the location of the requesting client can be inferred from the location of the recursive nameserver used by the client, assuming that they are located close by. This neglects various facts of today's Internet, as proven by recent studies showing mis-location of clients and the resulting degradation of end-to-end performance. Besides others, for example multi level DNS infrastructure, more and more end-users are moving away from using their ISP nameservers and switch to public DNS providers like Google or OpenDNS.

These companies operate global scale infrastructures for name resolution with multiple levels of anycast routing that make it impossible to find a stable mapping between clients and nameserver location, and thus allowing no statement on the distance between client and resolver. Also the resolver sending the request to the authoritative server is usually not the front-end server contacted by the client.

Studies show that ECS is indeed solving the problem of mis-locating clients for CDNs and CPs when relying on DNS to infer the client location by forwarding parts of the client IP-address to the authoritative nameserver within each request.

This is achieved by using the mechanism of EDNS to transmit an ECS payload containing the client IP-subnet in the additional section of the DNS query forwarded from the nameserver contacted by the client for name resolution towards the authoritative nameserver of the CDN or CP.

A key observation is that this information can be arbitrarily set by our experimental framework to forge any desired network location towards the CDN or CP being analyzed.

Using early ECS adopters like Google, Edgecast, and CacheFly as examples, our experimental study shows how simple it is by using a single vantage point, as simple as a commodity PC, to

(i) uncover the footprint of these CDN/CP companies,

(ii) observe them clustering clients,

(iii) take snapshots of the user-to-server mappings.

The analysis of our data collected over a period of several months reveals a significant growth of the infrastructure from Google and YouTube with respect to the number of IP addresses serving content, the number of ASes and countries we find. We point out that our approach allows for easy detection of GGCs within the networks of various providers. Further results point out potential implications of ECS on the cacheability of DNS responses by major DNS resolvers, such as ISPs.

We want to emphasize that our results can be reproduced by using only publicly available data and commodity hardware.

We believe that the tools developed and the traces collected in this work, made available to the research community, shed light on the deployment and operation of CDNs given the central role they play in today's Internet. This work also highlights the need to increase the awareness among current and future ECS adopters about the consequences of enabling ECS.

6.2. Discussion

The ECS extension to DNS tackles an actual problem that CDNs and CPs are experiencing every day: Bad performance for end-users using public DNS providers due to mis-locating these clients. While the approach of using the anyway necessary DNS requests for in-band signalling of the client sending the request in the first place is an efficient way of handling the submission of this additional data, it does come at a cost and leads to other challenges.

First of all by using an EDNS extension it is necessary that all nameservers involved need to support this underlying protocol. With Google we see how this can lead to other unexpected behaviour, e. g., forwarding of all EDNS queries to backend servers. It is possible that this is not always desired. Also by using EDNS the size of both the DNS requests and the responses will increase and appliances like firewalls and loadbalancers, that some people seem to use in front of nameservers, although generally disregarded in the operational community, need to support the forwarding in both directions.

The effect of caching of the answers has also to be studied. In our experiments we recorded many DNS responses that are to be cached only for a single host - which should happen at the host anyway and makes caching of these responses highly ineffective with regards to other clients requesting that resource. What we can say is that with ECS a nameserver will need more memory for caching of the answers depending on the scope returned and this amount will not only depend on the number and size of the client networks served but also on the answers generated by the operators of the authoritative nameservers - which to some extend is already true with regards to sending a very small TTL.

One major flaw we see in the protocol when it comes to automated deployment and transparency as the protocol has no mechanism to query a nameserver for support of the extension. This leads to various problems in the operational practice, one being the need of manual intervention in enabling ECS queries from e. g., Google and OpenDNS. The only way to find out if a nameserver is ECS enabled currently is to apply heuristics.

We suggest a concrete extension of ECS: The ability to query a resolver via ECS to respond with a set of preferences and capabilities supported. The answer should also include a version number of the protocol implemented.

This would allow for automated discovery of ECS on authoritative nameservers. The list of capabilities may further include flags to indicate if the nameserver should be automatically whitelisted, if all resolvers of the domain queried are ECS aware and if all zones the server is authoritative for are capable of handling ECS queries.

6.3. Future Work

We see several open questions that we think are worth exploring, using both passive and active measurements. During our work we got in contact with a CDN provider that is providing us with anonymized packet level traces of all DNS traffic from the ECS enabled authoritative nameservers of that CDN. Using this data we are able to

(i) analyze the distribution of the submitted client prefix length,

(ii) estimate the total amount of ECS enabled non-authoritative nameservers,

(iii) compare the improvement of accuracy seen by geolocating the resolver IP and the client IP transmitted,

(iv) analyze the share of ECS enabled queries versus traditional queries.

Together with other traces we have access to, we can estimate the possible benefits of enabling ECS for an ISP. It could either be beneficial to share information about the client networks or in the worst case this could overload single upstream links of that ISP. For example if all of the ISP's clients using a CDNs would suddenly be redirected to a cache reachable only through a link with limited bandwidth. Here a comparison to other techniques we have experience with, like PaDIS [40], would be interesting. An open question for ISPs is whether it is worth to support ECS now, although only a limited number of CDNs is actively supporting it. The key question here is: How big is the amount of traffic that would benefit from enabling ECS today?

We also see that an even more automated system with a limited number of queries performed could be interesting to run as a long time experiment to further analyze the infrastructural changes in certain CDNs, e. g., Google and YouTube.

It would also be interesting if the timescale for the experiments could be further decreased, for example by taking an approach backup software uses: Executing a full scale scan every month and only an incremental scan with a limited number from client IPs we dynamically extract from the full scan that we know are sufficient to detect all CDN servers.

Appendix A.

Technical Specifications

A.1. Database Schema

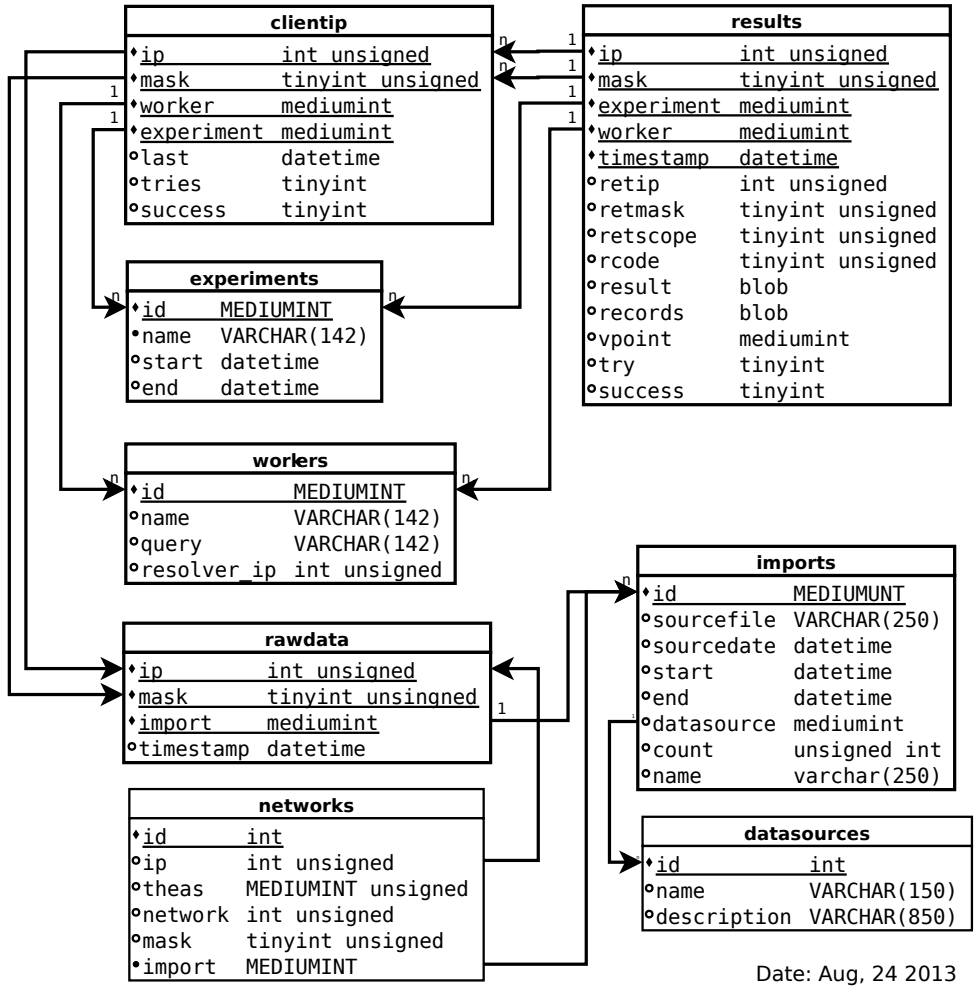


Figure A.1.: SQL-Schema of the used database

A.2. Third Party Software Required

A.2.1. Main Vantage Point / Coordinator

To prepare a Debian based system these issues install the needed distribution packages:

```
# apt-get install python-virtualenv python-pip python-mysqldb
# apt-get build-dep python-mysqldb
```

The following python packages are required:

- MySQL-python==1.2.4
- argparse==1.2.1
- distribute==0.6.35
- dnspython==1.10.0
- ipaddr==2.1.10
- ipython==0.13.1
- paramiko==1.11.0
- pycrypto==2.6
- pydns==2.3.6

Furthermore the bgpdump utility is required when the RIPE/Routviews data are going to be imported. For a source see `https://bitbucket.org/ripencc/bgpdump/wiki/Home`.

In addition a mySQL server has to be installed and running or reachable via network.

Details can be found in the Documentation available with the source code at:
`http://projects.inet.tu-berlin.de/projects/ecs-adopters/wiki`

A.2.2. Remote Vantage Point

The remote client was implemented to have as little external dependencies as possible for easy deployment on machines operated by third parties.

The only two python packages that need to be present in addition to a default install are simplejson and dnspython, we used these versions with success:

- simplejson==2.0.9
- dnspython==1.10.0

A.3. List of Hostnames and Nameservers Used

worker	resolver	hostname	description
googleauth	216.239.32.10	www.google.com	Google Authoritative
googleanycast	8.8.8.8	www.google.com	Google Anycast
youtube	216.239.32.10	www.youtube.com	YouTube Website
torrentfreak	85.195.84.42	www.torrentfreak.com	Torrentfreak
squeezebox	79.125.21.137	www.mysqueezebox.com	Logitech Squeezebox
edgecast	72.21.80.6	wac.67ef.edgecastcdn.net	Efgecast instance
edgcast-website	72.21.80.5	www.edgecast.com	Edgecast Website
cachefly	205.234.175.2	sedo.cachefly.net	Cachefly
hicloud	221.204.186.7	query.hicloud.com	Hi-cloud
ga-akamai	8.8.8.8	www.akamai.com	Akamai Website
ga-cdn77	8.8.8.8	www.cdn77.com	CDN77 Website
ga-limelight	8.8.8.8	www.limelight.com	Limelight Website
ga-cloudflare]	8.8.8.8	www.cloudflare.com	Cloudflare Website
ga-azure	8.8.8.8	az15570.vo.msecnd.net	Microsoft Azure Host
ga-amazon-inet	8.8.8.8	(witheld).cloudfront.net	Amazon CloudFront INET

A.4. Used/Example Configuration

```
1
2  [base]
3  backupdir: /shared/eDNS/backups
4  basedir: /home/florian/fls-edns/
5  scriptdir: /home/florian/fls-edns/helperscripts
6
7  [logging]
8  logdir: /shared/eDNS/logs
9  basename: edns
10 filelevel: info
11 consolelevel: info
12
13 [scripts]
14 #these live in the scriptdir set in base section
15 sqlbackup: backup_database.sh
16
17 [commands]
18 #full paths
19 bgpdump: /usr/local/bin/bgpdump
20 whois:   /home/florian/bin/whois
21
22
23 [mysql]
24 host:     localhost
25 database: production
26 user:     xxxxxxxxxx
```

```
27 password: xxxxxxxxxx
28
29
30 [smtp]
31 server:     mail.tu-berlin.de
32 to:         florian@inet.tu-berlin.de
33 sender:     florian.streibelt@tu-berlin.de
34 name_to:    Florian Streibelt
35 name_from: edns-experiment
36 prefix: [ecs]
37
38
39 # Settings for our importers, could have been
40 # hardcoded as well, but we might need
41 # additional values, so it won't hurt
42
43 [importer_hosts]
44 id: 1
45 name: Hosts-Importer
46 description: Imports hosts from file
47
48 [importer_routeview]
49 id: 2
50 name: Routeview-Importer
51 description: Imports routes from routeviews as network/masks
52
53 [importer_xpnetworks]
54 id: 3
55 name: Expanding-Networks-Importer
56 description: Imports list of networks/masks from file and expands them to
       IPs
57
58 [importer_networks]
59 id: 4
60 name: Networks-Importer
61 description: Imports list of networks/mask from file
62
63 [importer_splitworks]
64 id: 5
65 name: Subnetting-Networks-Importer
66 description: Imports list of networks/mask from file and splits them into
       subnets of defined size
67
68 [importer_cymru-mapping]
69 id: 6
70 name: AS-IP-Networks-Importer
71 description: Imports list of networks/mask, IP-Adresses and ASs from CYMRU
       mapping
72
73
74 [importer_virtual-subsamples]
75 id: 7
76 name: Virtual-Subsample-Importer
77 description: Selects defined number of datapoints from given imports
```

```
78
79
80
81
82 [worker_googleauth]
83 id: 1
84 resolver: 216.239.32.10
85 hostname: www.google.com
86 description: Google Authoritative
87
88
89 [worker_googleanycast]
90 id: 2
91 resolver: 8.8.8.8
92 hostname: www.google.com
93 description: Google Anycast
94
95 [worker_torrentfreak]
96 id: 3
97 resolver: 85.195.84.42
98 hostname: www.torrentfreak.com
99 description: Torrentfreak Authoritative
100
101 [worker_squeezebox]
102 id: 4
103 resolver: 79.125.21.137
104 hostname: www.mysqueezebox.com
105 description: Logitech Squeezebox
106
107 [worker_ga-akamai]
108 id: 5
109 resolver: 8.8.8.8
110 hostname: www.akamai.com
111 description: Akamai Website
112
113 [worker_ga-cdn77]
114 id: 6
115 resolver: 8.8.8.8
116 description: CDN77 Website
117 hostname: www.cdn77.com
118
119 [worker_ga-limelight]
120 id: 7
121 resolver: 8.8.8.8
122 description: Limelight Website
123 hostname: www.limelight.com
124
125 [worker_ga-cloudflare]
126 id: 8
127 resolver: 8.8.8.8
128 description: Cloudflare Website
129 hostname: www.cloudflare.com
130
131 [worker_ga-azure]
```

```
132 id: 9
133 resolver: 8.8.8.8
134 description: Microsoft Azure Host
135 hostname: az15570.vo.msecnd.net
136
137 [worker_ga-amazon-inet]
138 id: 10
139 resolver: 8.8.8.8
140 description: Amazon CloudFront INET Instance
141 hostname: d1e708se69b2p3.cloudfront.net
142
143
144 [worker_edgecast]
145 id: 11
146 resolver: 72.21.80.6
147 description: EDGECAST - supporter of afasterinternet
148 hostname: wac.67ef.edgecastcdn.net
149
150 [worker_cachefly]
151 id: 12
152 hostname: sedo.cachefly.net
153 resolver: 205.234.175.2
154 description: Cachefly
155
156 [worker_hicloud]
157 id: 13
158 description: Hi-cloud
159 hostname: query.hicloud.com
160 resolver: 221.204.186.7
161
162 [worker_edgcast-website]
163 id: 14
164 description: edgecast website at www.edgcast.com
165 hostname: www.edgecast.com
166 resolver: 72.21.80.5
167
168
169 [worker_youtube_ui]
170 id: 15
171 resolver: 216.239.32.10
172 hostname: www.youtube.com
173 description: YouTube Website
174
175
176
177 [worker_localhost]
178 id: 42
179 description: Local test-only worker
180 hostname: localhost
181 resolver: 127.0.0.1
182
183
184 # id=0 is the 'conventional' worker!
185
```

```
[vpoint_localhost]
id: 1
description: worker called directly, but using json api
type: intern

[vpoint_ssh_xxxxxxx.edu]
id: 10
description: xxx.edu worker on xxxxx.edu
type: ssh
host: xxxxxxx.edu
user: xxxxxxx
pass: xxxxxxx
runcmd: /home/guest/.../remoteclient.py

[vpoint_ssh_xxxxxxx.net]
id: 10
description: xxxxxx worker on xxxxxxx.net
type: ssh
host: xxxxxxx.net
user: xxxxxxx
pass: xxxxxxx
runcmd: /home/guest/.../remoteclient.py


[vpoint_ssh_local_test]
id: 23
description: worker running local via ssh
type: ssh
host: localhost
user: xxxxxxx
pass: xxxxxxx
runcmd: /home/.../remoteagent.py
```

Bibliography

[1] A Faster Internet Consortium. http://www.afasterinternet.com.

[2] Alexa top sites. http:///www.alexa.com/topsites.

[3] Google Global Cache. http://ggcadmin.google.com/ggc.

[4] Google Public DNS. https://developers.google.com/speed/public-dns.

[5] Mapping CDN domains. http://b4ldr.wordpress.com/2012/02/13/mapping-cdn-domains/.

[6] MaxMind, GeoIP databases. http://www.maxmind.com.

[7] OpenDNS. http://www.opendns.com.

[8] RIPE Routing Information Service. http://www.ripe.net/ris/.

[9] Routeviews Project, University of Oregon. http://www.routeviews.org/.

[10] D. Eastlake 3rd and A. Panitz. Reserved Top Level DNS Names. RFC 2606, IETF, Jun 1999.

[11] J. Abley and K. Lindqvist. Operation of Anycast Services. RFC 4786, IETF, Dec 2006.

[12] V. K. Adhikari, S. Jain, Y. Chen, and Z. L. Zhang. Vivisecting YouTube: An Active Measurement Study. In *IEEE INFOCOM*, 2012.

[13] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Comparing DNS Resolvers in the Wild. In *ACM IMC*, 2010.

[14] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Web Content Cartography. In *ACM IMC*, 2011.

[15] Bernhard Ager, Nikolaos Chatzis, Anja Feldmann, Nadi Sarrar, Steve Uhlig, and Walter Willinger. Anatomy of a large european ixp. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 163–174, New York, NY, USA, 2012. ACM.

[16] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, IETF, Mar 2005.

[17] M. Axelrod. The Value of Content Distribution Networks. AfNOG 9, 2008.

[18] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan. Mapping the Expansion of Google's Serving Infrastructure. In *ACM IMC*, 2013.

[19] N. Chatzis, G. Smaragdakis, J. Boettger, T. Krenc, and A. Feldmann. On the benefits of using a large IXP as an Internet vantage point. In *ACM IMC*, 2013.

[20] Lorenzo Colitti, Erik Romijn, Henk Uijterwaal, and Andrei Robachevsky. Evaluating the effects of anycast on dns root name servers. *RIPE document RIPE-393*, 2006.

[21] C. Contavalli, W. van der Gaast, S. Leach, and E. Lewis. Client subnet in DNS requests (IETF draft). `http://tools.ietf.org/html/draft-vandergaast-edns-client-subnet-01`.

[22] A. Dhamdhere and C. Dovrolis. Twelve Years in the Evolution of the Internet Ecosystem. *IEEE/ACM Trans. Networking*, 19(5), 2011.

[23] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. Reducing Web Latency: the Virtue of Gentle Aggression. In *ACM SIGCOMM*, 2013.

[24] M. J. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan. Geographic Locality of IP Prefixes. In *ACM IMC*, 2005.

[25] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632, IETF, Aug 2006.

[26] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519, IETF, Sep 1993.

[27] Google. What is 1e100.net? `http://support.google.com/bin/answer.py?hl=en&answer=174717`.

[28] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, IETF, Aug 2012.

[29] C. Huang, A. Wang, J. Li, and K. Ross. Measuring and Evaluating Large-scale CDNs. In *ACM IMC*, 2008.

[30] B. Krishnamurthy and J. Wang. On Network-aware Clustering of Web Clients. In *ACM SIGCOMM*, 2001.

[31] R. Krishnan, H. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. Moving Beyond End-to-end Path Information to Optimize CDN Performance. In *ACM IMC*, 2009.

[32] C. Labovitz, S. Lekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet Inter-Domain Traffic. In *ACM SIGCOMM*, 2010.

[33] Z. Mao, C. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers. In *USENIX ATC*, 2002.

[34] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 123–133, New York, NY, USA, 1988. ACM.

[35] E. Nygren, R. K. Sitaraman, and J. Sun. The Akamai Network: A Platform for High-performance Internet Applications. *SIGOPS Oper. Syst. Rev.*, 2010.

[36] E. Osterweil, D. McPherson, S. DiBenedetto, C. Papadopoulos, and D. Massey. Behavior of DNS' Top Talker, a .com/ .net View. In *PAM*, 2012.

[37] J. S. Otto, M A. Sánchez, J. P. Rula, and F. E. Bustamante. Content delivery and the natural evolution of DNS - Remote DNS Trends, Performance Issues and Alternative Solutions. In *ACM IMC*, 2012.

[38] C. Partridge, T. Mendez, and W. Milliken. Host Anycasting Service. RFC 1546, IETF, Nov 1993.

[39] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Com. Networks*, 31(23–24), 1999.

[40] I. Poese, B. Frank, B. Ager, G. Smaragdakis, and A. Feldmann. Improving Content Delivery using Provider-aided Distance Information. In *ACM IMC*, 2010.

[41] I. Poese, S. Uhlig, M. A. Kaafar, B. Donnet, and B. Gueye. IP Geolocation Databases: Unreliable? *ACM CCR*, 41(2), 2011.

[42] Y. Rekhter, Ed., T. Li, Ed., S. Hares, and Ed. A Border Gateway Protocol 4 (BGP-4). RFC 4271, IETF, Jan 2006.

[43] M A. Sanchez, J. .S. Otto, Z. S. Bischof, D. R. Choffnes, , F. E. Bustamante, B. Krishnamurthy, and W. Willinger. Dasu: Pushing Experiments to the Internet's Edge. In *USENIX/ACM NSDI*, 2013.

[44] J. Schlyter and W. Griffin. Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints. RFC 4255, IETF, Jan 2006.

[45] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar. Answering What-if Deployment and Configuration Questions with Wise. In *ACM SIGCOMM*, 2009.

[46] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci. Unconstrained Endpoint Profiling (Googling the Internet). In *ACM SIGCOMM*, 2008.

[47] S. Triukose, Z. Wen, and M. Rabinovich. Measuring a Commercial Content Delivery Network. In *WWW*, 2011.

[48] P. Vixie. Extension Mechanisms for DNS (EDNS0). RFC 2671, IETF, Aug 1999.

[49] P. Vixie. DNS Complexity. *ACM Queue*, 5(3):24–29, 2007.

[50] P. Vixie. What DNS is Not. *Comm. ACM*, 52(12):43–47, 2009.

[51] M. Wong and W. Schlitt. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408, IETF, Apr 2006.