Timo Valkola

Improvements to DNS privacy and integrity

Metropolia University of Applied Sciences

Bachelor of Engineering

IoT and Cloud Computing

Bachelor's Thesis

19 March 2023

Abstract

Author:	Timo Valkola
Title:	Improvements to DNS privacy and integrity
Number of Pages:	27 pages
Date:	19 March 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	IoT and Cloud Computing
Supervisors:	Janne Salonen, Head of School

The goal of this thesis was to talk about the most important recent improvements to a one of the most influential protocols for the whole internet, Domain Name System. Almost all the most fascinating new proposals are centred on how to provide better privacy and thus anonymity for DNS queries, encryption being a necessity for any kind of anonymity.

Based on evaluating different proposals, two of very recent developments (μODNS and DoHoT) were chosen for testing the ease of implementing them in an actual environment on how they performed in metrics and on how the overall experience felt for the user. Theoretically, both provide a large amount of collusion resistance on the ever-hostile internet environment where decentralization is highly needed if any type of privacy is required for users.

DoHoT was easily deployed to test environment located on thesis writer's home network. The performance was acceptable even for everyday use with some caveats. μODNS system was deployed to datacenter in Frankfurt and after some mishaps in configuration, a working solution was achieved.

It can be concluded that both protocols can readily be deployed on other systems with the help of this paper on how to manage the installation. Much improvements are needed for both protocols to make them internet standards for which there are other more performant proposals. With the correct implementation, μODNS should be able to achieve a real-world performance comparable to more simple solutions such as DNS over HTTPS. Unfortunately, μODNS can't be considered anonymous solution for client DNS queries based on the fact that there aren't enough public μODNS relays available in the internet.

Keywords: DNS, encryption, privacy

Tiivistelmä

Tekijä:	Timo Valkola
Otsikko:	Parannukset DNS prokollan yksityisyyteen ja eheyteen
Sivumäärä:	27 sivua
Ajankohta:	19.3.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	IoT ja pilvipalvelut
Ohjaaja:	Janne Salonen, Osaamisaluepäällikkö

Tämän insinöörityön tavoitteena oli tuoda esiin viimeisimmät ja tärkeimmät kehitetyt standardit ja kehityssuunnat, joita internetin yhdelle tärkeimmistä protokollista, DNS:lle, on kehitetty. Suurin osa kehityssuunnista ja mielenkiintoisista protokollaparannuksista on suuntautunut DNS-kyselyjen yksityisyyden ja anonymiteetin parantamiseen.

Työssä lähdettiin arvioimaan uusia protokollia ja standardeja ja arvion perusteella päädyttiin kahteen uuteen ehdotettuun protokollaan (μODNS ja DoHoT), joita lähdettiin toteuttamaan testiympäristöihin. Tarkoitus oli arvioida niiden toteuttamiskelpoisuutta ja suorituskykyä, sekä arvoida myös toteutuksen mielekkyyttä. Molemmat standardit tarjoavat myös huomattavan lisäsuojan internetissä vaikuttavia suuria toimijoita kohtaan, joilla on resursseja korreloida resurssien lataamista DNS-kyselyiden perusteella.

DoHoT-toteutus saatiin suhteellisen pienellä vaivalla toteutettua kotona sijainneeseen testiympäristöön. Sen suorituskyky normaalissa päivittäisessä käytössä oli kohtuullinen, mutta hidas ideaalisuorituskykyyn nähden. μODNS-ympäristö luotiin Akamain Frankfurtin konesaliin kahdelle virtuaalipalvelimelle, josta DNS-kyselyt välittyivät kokeellisille välityspalvelimille Tokiossa. Tämän toteutuksen kohdalla kohdattiin enemmän teknisiä haasteita.

Voidaan todeta, että molemmat toteutukset saatiin toimimaan ja DNS-kyselyt välittymään. Molemmissa protokollissa on silti huomattavasti kehitettävää, ennen kuin niitä voidaan odottaa laajaan käyttöön, sillä palveluiden konfigurointi vaati suhteellisen runsaasti tietotaitoa käyttäjältä. Totesimme myös, että mikäli μODNS palvelimia olisi runsaasti tarjolla lähellä tämän järjestelmän tarvitsemia välityspalvelimia, olisi toteutuksen viiveet todennäköisesti kohtuullisia.

Valitettavasti kolmella testipalvelimella jotka sijaitsevat Tokiossa, ei μODNS ole järkevä ratkaisu DNS-kyselyiden anonymiteetin kannalta. Pienestä massasta μODNS-protokollan mukaisia kyselyitä on helppo tunnistaa kyselyiden tekijä. Näin ollen tämä muuten mielenkiintoinen protokolla tarvitsisi huomattavan paljon käyttäjämassaa taakseen, ollakseen mielekäs tulevaisuuden ratkaisu.

Avainsanat: DNS, salaus, yksityisyys

Contents

List of Abbreviations

μODNS: Mutualized Oblivious Domain Name System. A novel and new solution to improve DNS query anonymity in a hostile environment.

CDN: Content Delivery Network. Delivers the content from internet to clients.

DNS: Domain Name System. Software that translates and provides IP addresses for queries that have human readable addresses as input.

Do53: Domain Name System over port 53. When traditional DNS is compared to the modern ones, Do53 is used as abbreviation to the standard unencrypted DNS queries.

DoH: Domain Name System over Hyper Transfer Protocol Secure. A more secure version of sending DNS queries over HTTPS.

DoHoT: Domain Name System over Hyper Transfer Protocol Secure over Tor. A new and novel solution for DNS query privacy and authenticity.

DoT: Domain Name System over Transport Layer Security. Similar solution as DoH, but relying on TLS on DNS query encryption.

DNSSEC: Domain Name System Security Extensions. Security extensions providing authenticity for DNS queries.

IETF: Internet Engineering Taskforce. A central body responsible for standardization regarding internet protocols and new proposals for standards.

ISP: Internet Service Provider. Usually a company that provides an internet connection for a local entity, for example Telia or Elisa in Finland.

ODoH: Oblivious DNS over HTTPS. A new standard already in use for more secure DNS queries with certain degree of anonymity.

ODNS: Oblivious Domain Name System. A protocol providing DNS queries with a degree of anonymity.

RFC: Request for Comment. New proposals as IETF standards.

RTT: Round trip time. Time it takes for a query to reach the destination and for the result to propagate back to the sender.

TCP: Transmission Control Protocol. A backbone connection-oriented protocol for internet traffic.

TLD: Top Level Domain. These domains are located right under internet root domain.

TTL: Time-to-Live. How long a value stays in a local cache before it is removed.

UDP: User Datagram Protocol. A backbone connectionless protocol that is used to send messages over internet.

1 Introduction

Domain Name System (DNS) can be considered one of the most fundamental building blocks for modern and complex internet to exist. Originally coming into existence to help with the growing pains of ever-increasing number of new devices that were centrally listed in a one text file [1], the original DNS was only about translating human readable and understandable hostnames (for example icann.org) to a matching IP address, used in routing the traffic to the right server. All this was done in plaintext with placing ultimate trust to infrastructure responsible for the query. Thus, anyone wiretapping or monitoring the client's connection would immediately know the servers they connected. This approach worked well then from performance standpoint and only minimal changes were made to the protocol compared to most recent ones that this thesis concentrates into.

Having no encryption and no authenticity guarantee for queries in a network of mostly western military instances and universities made more sense than in the current times with evermore hostile internet. Also, encryption standards were rudimentary and under constant development and even the traffic to servers was almost always unencrypted. DNS traffic encryption in this environment would have made very little sense. Authenticity of DNS query responses was a more acute problem, first properly and officially addressed in the early proposals for Domain Name System Security Extensions (DNSSEC) by Eastlake in Internet Engineering Taskforce's (IETF) Request for Comment (RFC) 2535 [2].

With the ever-increasing computing power, encryption of DNS queries is way less expensive in terms of monetary resources and time. Standards first developed to secure the traffic between server and client were found useful to secure the DNS queries as well with the advent of DNS over TLS (DoT) and DNS over HTTPS (DoH). Encryption of the queries is vitally important if any kind of resemblance of privacy is wanted for DNS.

This thesis will explain some of the basics in how DNS technically works from the privacy standpoint and expands on the very recent developments on how confidentiality and authenticity of query responses is achieved. Any meaningful level of privacy can't be achieved without some level of security so both must co-exist. Privacy of electronic communications (including the DNS queries) could be considered vital if any type of privacy is desirable. Availability and reliability of DNS remains a paramount importance in real life, but this paper is mostly limited to confidentially and authenticity with real world testing on the performance. However, as Schmid [3] points out, real-life attacks can't usually be rendered to only one aspect (for example authenticity) but expand on some particular violation to another attack vectors and can touch multiple adjacent systems in addition to the DNS.

Few of the most interesting new proposed protocol improvements take very seriously the possibility of collusion of the DNS relay network in addition to securing the queries. Given how centralized the current internet landscape is, this is a reasonable assumption. The main problem even when the authenticity of DNS resolvers is verified and confidentiality is achieved that the resolver will need to strip the encryption to find the correct IP address, thus seeing every single query that the client makes [4]. Mitigations have been provided in the form of Oblivious DNS (ODNS) but are inadequate to protect the privacy of queries from DNS relay and full-service resolver that collude [4]. The practical part of this thesis shows an actual implementation in a home network environment with a DNS system that tries to protect from the collusion as with usage of Tor protocol in the DNS queries. This DNS over HTTPS over Tor (DoHoT) can be considered quite extreme approach and comes with some serious limitations, especially regards to performance. DoHoT system is then compared to very novel solution, mutualized oblivious DNS (μODNS). The goal is to provide analysis on how these new techniques improve the privacy and security of the DNS and answer to the question whether the most extreme methods are practical to implement and use.

2 History of DNS and basic principles

As one of the most fundamental technologies that enables internet to exist, stands Domain Name System (DNS). It is the necessary bridge to transform human comprehensible names to IP addresses, better suited for computers. Originally DNS was proposed as a dynamic protocol by Mockapetris in the 1983, to better handle the growing rate of new network hosts that were all listed in one infeasibly large and centralized database file where each new computer connected to the ARPANET needed a new entry. This list consisted of the domain names and associated IP addresses. [1.]

The original RFC was revised multiple times, but the basic nature of the protocol has stayed the same. Each domain name needs an IP address where the traffic coming from the internet can be routed. When a computer is given the task to find the IP address for a domain name, it can first consult it’s own cache of recent DNS lookups. If this DNS cache is empty of such results or time-to-live (TTL) value of results have expired, then the query is passed into the DNS resolver in the form of User Datagram Protocol (UDP) packets. Resolver will in turn check whether that result exists in cache and so on. This is the recursive form of querying the IP [3]. Finally, if needed, the query can pass to a top-level domain (TLD) which under the root and foundation of the DNS itself. An example of querying the IP address for moodle.metropolia.fi can be seen in figure 1 [3]. This traditional process that has no guarantee of the authenticity of queries nor are the queries confidential (i.e. encrypted), is often called Do53 [4] in literature.

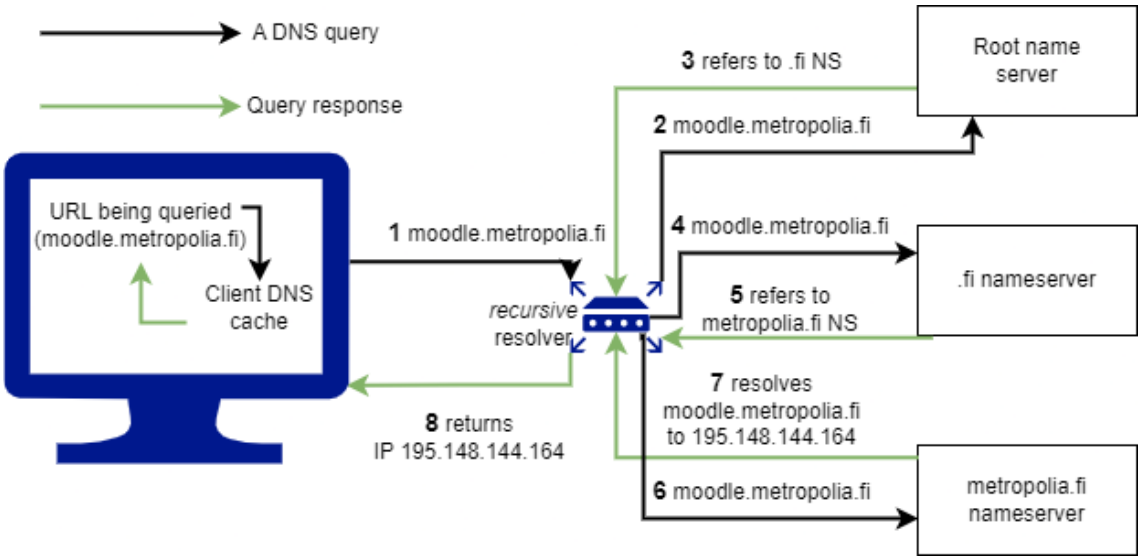

Figure 1. Logistics of a traditional Do53 DNS query for moodle.metropolia.fi. Recursive resolver can be a collection of resolvers, for example a forwarding resolver being adjacent to client and an area resolver at a company level. Quite often queries will only pass to recursive resolvers, the alternative being iterative resolver.

The basic principles of this multilayer process have stayed almost the same since the advent of DNS, even though slight modifications were made by Mockapetris in 1980s in the form of RFC 1035 [5]. Most of the latest RFCs related to DNS, have been about DNS related terminology in the current discussion, RFC 8499 being the latest official revision on the subject [6].

2.1 Improvements to DNS authenticity and integrity

DNS Security Extensions (DNSSEC) seeing it's first draft in 1997 was the initial step toward giving rudimentary protection against tampering the DNS queries. Although DNSSEC as a protocol could be considered very flawed in today's standard according to Schmid [3] and some others, it managed to provide some level of authenticity and integrity, although the protocol was basically left dead in the water for many years with wide adoption taking well over a decade, further hindering the original goals of the proposal.

DNSSEC makes it considerably harder to spoof and hijack the DNS resolution results by introducing a chain of trust from the TLD zone to the actual domain name. This is achieved with some additional resource records (RRs), hence the name “security extension”. It relies on a zone-signing key and key-signing key to validate the chain of trust between the hierarchical zones, ending with inherent trust to the TLD zone.

The inefficient adoption wasn’t completely due to the architecture of the DNSSEC but mostly due to the DNS operators not caring about the feature enough to implement it for the domains they served and some operators charging extra for the feature [7]. Situation has vastly improved only in recent years. One reason was the additional complexity that enabling DNSSEC for a domain required with constantly changing protocol that even experienced administrators had problems configuring correctly [3].

Schmid [3] also points inefficiency, usability, effectiveness and privacy as big factors. Fortunately, efforts by bigger and some smaller entities have rendered this complexity of adaptation almost moot. For example, Cloudflare offered DNSSEC enablement for domains hosted on some of registrars with a click of a button in 2018 [7]. This doesn’t mean that the other hurdles are solved for the protocol, especially user privacy as privacy wasn’t in DNSSEC’s design. The last iteration of extending DNSSEC with another extension, NSEC5 by Vcelak et. al. [8], offering a level of privacy for queries was scraped by IETF, never reaching any level of adoption. Had the NSEC5 model ever been adopted, it might have solved some problems with attackers enumerating all the records in zone thus finding vulnerabilities, leading to compromise of the systems collapsing any kind of privacy. Even this was speculative by the proposed extension that made the DNSSEC even more complicated.

2.2 Encryption augmentations

DNS being such a legacy protocol, privacy issues arising from the lack of encryption lacking from the original proposals have risen to the forefront of discussion just by looking some of the latest RFCs that are still categorized as experimental, RFC 9230 being a prime example by introducing Oblivious DNS over HTTPS (ODoH) [9] which itself relies on another proposed RFC standard, DNS over HTTPS (DoH). These proposed and especially experimental RFCs are under heavy development based on their modification history but it's fair to say that the techniques they introduce are likely to stay due to actual and rather quick implementations by largest networking companies, for example by Cloudflare [10] already implementing ODoH in 2020.

Two protocols, DoH and DNS over TLS (DoT), are in widespread use even though they weren't the first solutions to tackle DNS encryption problem. DoT secures a session between a client and the resolver by utilizing Transport Layer Security (TLS) and for most communication, opting for TCP packets. TLS based communication makes security of DoT as secure as the implemented TLS communication between the client and recursive resolver in the transport layer. DoH is very similar but works on the application layer by using HTTPS for communication between resolver and client. Public recursive resolvers with both DoT and DoH have been widely available from some of the largest internet traffic hubs such as Cloudflare and Google for a few years now, Google opening DoT service on 2019 [3].

DoH and DoT could be considered a great improvement security wise. They both employee tried and tested protocols that have great underlying support for infrastructure such as proxies and firewalls and thus work with minimal changes to overall network topology. Especially DoH paid special attention to interoperability with traditional Do53 queries in its query parameters. For collusion resistance these protocols did nothing. As mentioned, some of the largest public DNS resolvers with DoH and DoT resolvers are the same that also serve the

content, being sometimes technically in the position to know the full chain of the communication from the client.

However, the first proposal to tackle the lack of encryption was DNSCurve and the successor DNSCrypt [4, p. 3]. DNSCurve also could have acted as a substitution for DNSSEC but IETF never moved it to a status of an internet standard as pointed out by Schmid [3, p. 2446].

DNSCrypt protocol builds on the existence of resolver certificate that the DNSCrypt resolver provides for the client after the client (which can be a DNSCrypt-proxy) makes a DNS query to the resolver. The certificate exchange is facilitated with use of Ed25519 digital signature algorithm and X25519 Diffie-Hellman key exchange algorithm. Both are well studied and modern algorithms. With the key exchange completing successfully, symmetric authentication can then proceed, facilitating further communications for actual DNS name resolution. When there is a proxy facilitating the key exchange, the actual resolver won't immediately be aware of the original client making the requests and a basic level of anonymity of the queries is achieved from the resolver's point of view. This depends on the resolver though, whether it accepts only certain public keys from clients, thus limiting on who can make queries using the resolver [3, p. 2447]. Even though the actual use of DNSCrypt protocol has dwindled due to more advanced approaches, the name still lives on some very prolific projects naming scheme.

This possible loss of privacy is mostly due to possible collusion with DNS server infrastructure and other internet infrastructure that serves the content from the IP addresses resolved by DNS protocol. Quite a large amount of internet content comes from content delivery networks (CDNs). DNS servers that the client uses, have the knowledge of client IP address and the queries the client has made and those can be referenced to the communication between servers and the client by timing them. This fact is exaggerated considering that some entities have high presence in both hosting widely use DNS services and delivering content, Google being a prime example. This collusion problem is illustrated by Kubo and Kurihara in figure 2. A simple encryption scheme and even proxying the queries doesn't suffice for privacy in such a case of collusion to the level of colluding infrastructure.

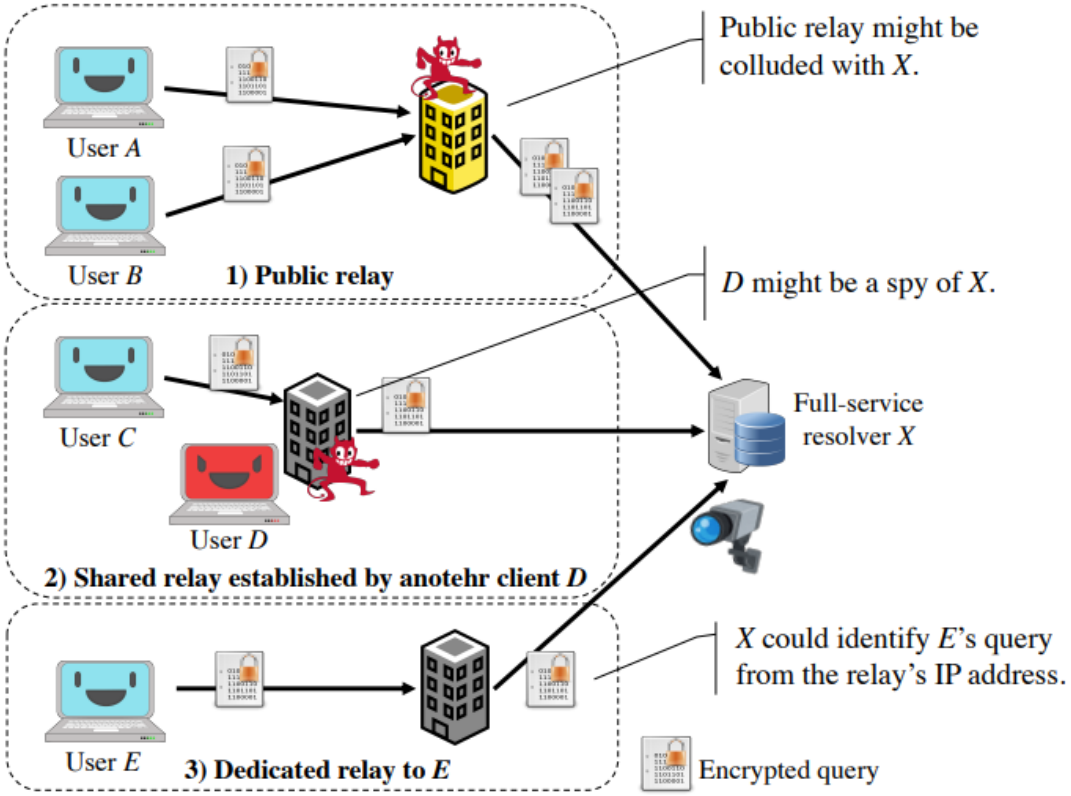

Figure 2. Formulation of the problem arisen from collusion of the DNS infrastructure according to Kubo and Kurihara. Copied from [4, p. 5].

3 Experimental solutions for better privacy

As already mentioned, there must be a point where plaintext query is matched with the DNS records and the result communicated back to the party making the query. If the whole communication hierarchy is colluding with the resolver, then no anonymization is possible. The problem becomes hard even if colluding parties are limited in number. One call already see that this complication has created the ever-increasing need for new and better protocols to tackle the problem as the widely adopted DoT and DoH protocols don’t solve it themselves.

3.1 ODNS

Schmitt et. al. presented a first iteration of collusion resistant DNS solution called Oblivious DNS (ODNS) [11]. The protocol’s main imperative was to limit server-side visibility for client’s identifiers possibly leaking client’s identity. They listed as their foremost motivation to hide user queries from internet service providers (ISPs), quite often operating recursive resolvers. Also, this approach helps with hiding the client identity behind two new entities that the protocol implements, namely ODNS stub resolver (that hides client queries by encryption) and ODNS resolver that doesn’t know the identity of the originating client as there is a normal recursive resolver in between [11, p. 232]. On the server side, only in between ODNS resolver and authoritative name server are the queries unencrypted. The process is described in figure 3.

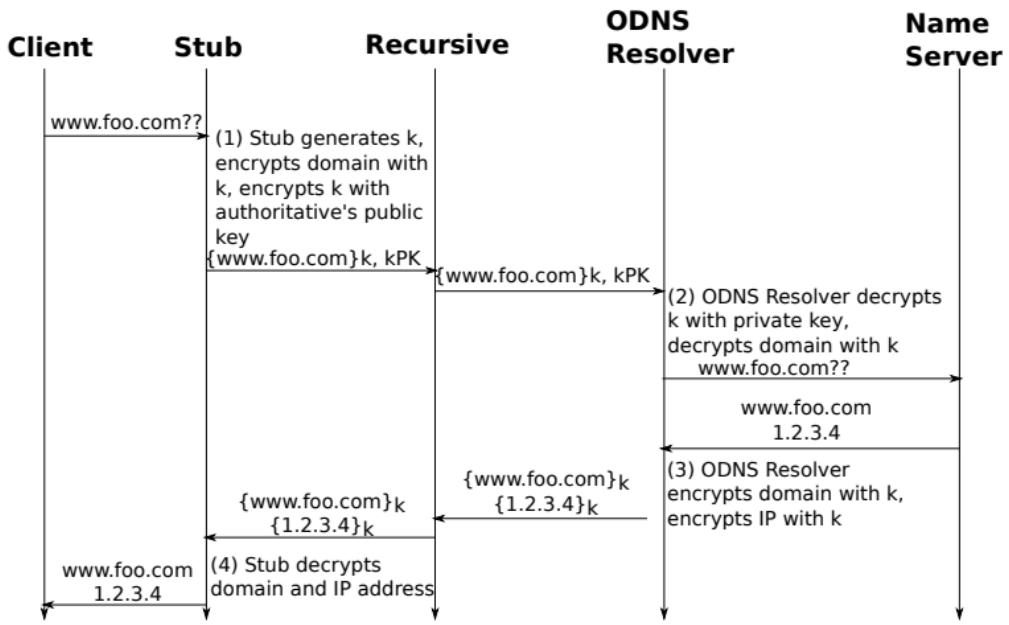

Figure 3. A high level view of ODNS protocol. Copied from [11, p. 232].

The mode of operation for ODNS has many privacy benefits. Unfortunately this solution is less than ideal for performance, at least in the extreme cases, due to ODNS resolvers inherent vulnerability to DDoS attacks as ODNS resolver can receive large amount of queries from clients that have identities hidden by design from the resolver. Also, the protocol demands asymmetric and symmetric encryption that is more taxing than some alternatives and naturally increases latency for connections. Latency can’t be avoided in any scenario where encryption is present.

This DDoS scenario can be overcome by using large number of separated replicas distributed in many different geolocations that share the same anycast address that is then advertised to TLD servers. Recursive resolver queries are then distributed to different ODNS resolvers that share the same anycast address by the Border Gateway Protocol (BGP) metrics. Based on the initial evaluation of Schmitt et. al., the ODNS doesn’t punish latency values much, at least not in a manner that an internet user might notice. [11, p. 233.]

As the authors admit, ODNS has its limitations when considering the threat model where recursive resolver and ODNS resolver collude. Then adversary will have no trouble associating IP addresses with queries [11, p. 241]. This might even be possible if adversary can time DNS queries from the client to the actual request of internet content for example from CDN. As Muffett points it out, the inherent weakness of this proxied system is the fact that the number of these proxies and resolvers are very limited (at first at least before widespread adaptation) and they make a very high value targets for wiretapping type of surveillance [12, p. 7]. This is further confirmed when looking at the formulation of μODNS protocol's research question [4, p. 2]. Thus we can conclude that ODNS has a large shortcoming when considering its usefulness for DNS query anonymization.

3.2 ODoH

Oblivious DNS over HTTPS can be characterized as a very new protocol that almost immediately gained traction in actual implementations, most likely because it was a joint project between researchers from Cloudflare, Apple and Fastly [10]. The protocol heavily relies on earlier technologies, namely HTTPS to encrypt the queries with (DoH) and also it introduces a proxy between queries that is trusted **not to** collude with actual resolver that sees the query and the IP address of proxied server as the proxy server has no way of knowing the actual query nor the answer for query. Proxy only sees the address of client and then passes the query for available target.

If no collusion really happens between proxy and resolver, ODoH would be the perfect solution regarding overall performance as well, as the public key encryption for DoH part of the protocol is a very prioritized computational problem with the prevalence of HTTPS traffic in general. Compared to DoH, ODoH only introduces a mere 1 ms of additional latency as measured by Cloudflare when the ODoH option was rolled out in 2020 [10]. Considering that the overall time for a query (when the result is not in the local cache of client) and a response, called

round trip time (RTT) is usually tens or hundreds of times higher, this additional latency is insignificant. At least ODoH theoretically and practically can protect the client's privacy by removing the ultimate trust from the resolver as the compromise of resolver or target doesn't lead to failure in security and privacy for the client. But as discussed earlier, there is no ultimate safeguard except trust in the companies or entities running the proxy and resolver servers.

3.3 DoHoT

DNS over HTTPS over Tor goes to very different territory compared to the other solutions, especially how it approaches the privacy of DNS queries. It widely differs from aforementioned protocols by using Tor network for DoH queries. As Muffett mentions, DoHoT is quite often confused with Tor daemon's in-built UDP Do53 DNS resolver [12, p. 2]. However, DoHoT just routes DoH queries over Tor network, utilizing .onion addresses or bouncing the DoH queries through Tor relay network to Tor exit nodes to normal resolvers with TCP/IP addresses. This whole approach can be considered rather extreme on the anonymization front even if it still leaves a gap for client detection since resolver might still be colluding with the party responsible to serving the actual content based on the DNS name resolution. Cloudflare unveiled their hidden Tor onion service that for forwards all the DNS queries to Cloudflare's own resolver reachable at 1.1.1.1 IP address [13]. This is described in figure 4.

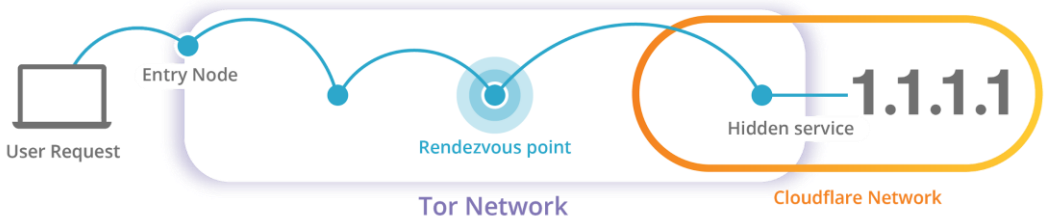

Figure 4. Cloudflare hidden onion resolver. Copied from [13].

Unlike using just Tor network for DoH queries, Cloudflare approach removes exit nodes from the picture altogether. This brings the benefit of not trusting Tor exit

nodes with any traffic, as Cloudflare resolver *is* an actual .onion service. Tor exit nodes are considered as one of the weakest points of Tor protocol as collusion of them can reveal the client identity with a timing attack. Thus, any meaningful DoHoT solution must use multiple exit nodes to protect against such inherent vulnerability or use onion services (i.e., dark web addresses). Proper DoHoT solutions also rely on Tor Bridges to bypass censorship to Tor network. They work as hidden entry-points to Tor network. This is a clear improvement compared to rather centralized approach of ODNS for an example. DNS over Tor also brings the benefit of constant-size cells to battle against traffic analysis based on query sample size analysis [14, p. 11]. This can be further enhanced by correct DoHoT client behaviour of trying to blend in with other clients using Tor network by minimizing the uniqueness of client DoH queries.

Unfortunately, even Tor network requires that clients actually use it, otherwise it's relatively easy to spot a very small amount of clients using it as Muffett admits [12, p. 6]. This problem is greatly exaggerated for example when adversarial or even colluding Tor end node or DoH resolver correlates the small amount of DNS queries with actual requests for network resources. Hiding from the plain sight requires multiple parties implementing the protocol. This can only be overcome with widespread adaptation that usually requires participation of for-profit companies that might not benefit of such extreme anonymization (e.g. Google, Amazon).

As always with Tor, performance is hindered compared to a standard DoH approach for example, as traffic must pass through multiple relays. Additional padding to reach constant cell size is a performance hindrance that needs to be paid if Tor network is to be used at all. Some consider this performance cost to be too high to even consider the usage of the protocol in real world implementations, like Kurihara and Kubo in their proposal of μODNS [4, p. 4]. This hindrance will later be tested in a real world implementation for a fast home network.

3.4 μODNS

As the name suggests, μODNS is a proposed improvement to ODNS protocol by Kurihara and Kubo [4]. The main weakness in ODNS lies in the fact that there is no practical resistance to colluding proxies and resolvers as admitted by the authors of ODNS. μODNS tries to improve this situation by recognizing that the client shouldn't need to trust most proxies or relays in DNS networks [4, p. 2].

Kurihara and Kubo offer a very similar problem formulation compared to the one ODoH tried to solve with the main difference being less trust put for nodes in DNS query topology. Queries are being sent and divided to multiple large public DNS entities (Cloudflare, Google, Quad9 etc.) through relays that might collude with the public resolvers [5, p. 5]. The larger the number of colluding resolvers, the harder the situation naturally becomes for client anonymity, but this protocol at least offers a theoretical relative safety, when the whole network infrastructure is not colluding. As with all the other protocols, anonymity heavily relies on people (or clients) actually using the protocol on the real world. Small number of active users should be relatively easy to spot with traffic analysis and timing attacks.

It is proposed that user should be able to trust at least the closest relay in their own network infrastructure that then acts as their trusted and dedicated relay, passing queries to other relays and resolvers as described in Figure 5. Compromise in this relay compromises the DNS security completely. In private network environment (i.e., at a company level or at home), this can be considered an acceptable approach, considering the client putting their trust in their own on-premises equipment for routing traffic for example. A relay that exists behind a router and a firewall is an additional attack surface but with good security procedures, shouldn't be considered the largest threat for the actual user. For practical usefulness, Do53 queries can be used from the client to the dedicated relay. It's good to know that in this scenario, wiretapping the connection between client and dedicated relay will lead to compromise of privacy for the queries. This can be considered acceptable because of wide interoperability achieved with different clients in home network for example.

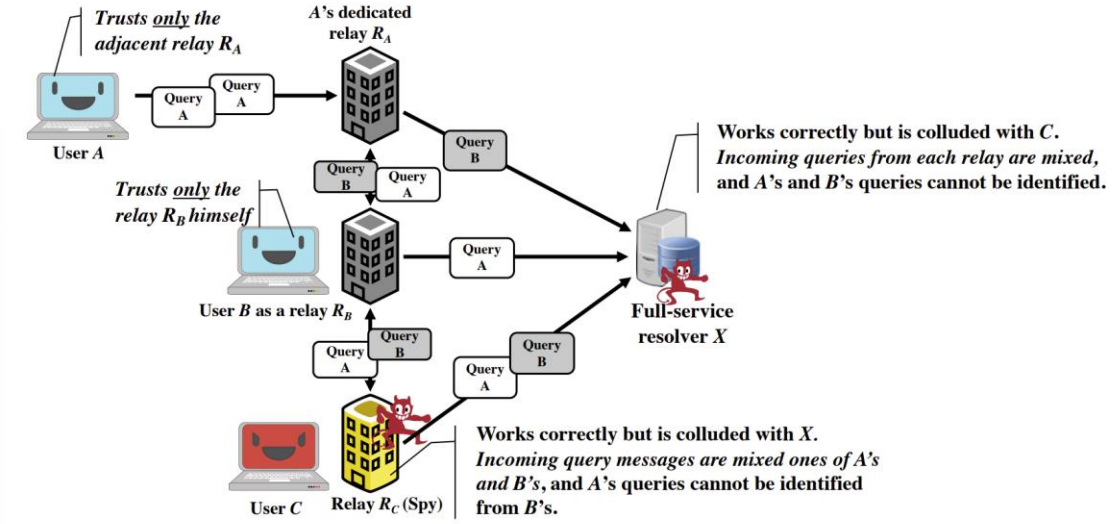

Figure 5. How μODoH works when implemented correctly according to proposal from Kurihara and Kubo. Copied from [4].

Interestingly, Kubo and Kurihara immediately admit that DoHoT could be considered a very feasible technical solution due to it being naturally collusion resistant and thus offering collusion resistance with the added benefit of providing encryption for the actual DoH queries [4, p. 4] even when they exit Tor. However, this is immediately rejected based on the performance of DoHoT due to the large number of nodes employing layered encryption in each node, this being the nature of Tor. No measurement or comparison is provided by them, comparing DoHoT and μODNS on performance. Only μODNS performance is measured by them. The practical part of this thesis tries to provide a real-world comparison of these two protocols.

4 Implementing DoHoT and μODNS systems

Of all the discussed protocol improvements to DNS security and privacy, μODNS and DoHoT seemed most interesting to be compared for their real world performance and ease of implementation and running the systems. Both

protocols can be considered very novel by integrating multiple recent improvements to DNS security and privacy. As such, there hasn't been many widely deployed implementations or instructions on these protocols yet. For DoHoT, there seemed to be only one, namely Muffett's implementation [15], with steps available for reproducing necessary components to take advantage of the DoHoT. This was selected as the approach for DoHoT system. The other implementation completely relied on Cloudflare with their pre-built binary, *cloudflared*, and also relying on their .onion service hidden resolver [16]. This approach seemed self-defeating for the purpose of query anonymity because of possible collusion.

For μODNS there wasn't much to choose from, except proof-of-concept (PoC) GitHub pages by Kurihara. At the time of writing, Kurihara had provided a PoC for client use in the form of DNSCrypt-proxy, modified to use μODNS relays as a possibility [17]. Also, the relay server, responsible to relay DNSCrypt v2 protocol messages to upstream μODNS relays, is a fork of DNSCrypt project's *encrypted-dns-server* [18]. These two were deployed in conjunction to test the feasibility of implementation and performance.

4.1 Implementing DoHoT

The most worthwhile DoHoT implementation publicly offered seemed to be Muffett's fork of DNSCrypt-proxy adjusted to DoHoT usage [15]. This fork provides a Do53 capable proxy, able to translate ordinary non-encrypted Do53 queries from local network clients through the proxy to DoH resolvers over Tor [12, p. 1]. DNSCrypt-proxy itself accepts many other protocols beyond DoH or DNSCrypt. This implementation seemed feasible to be reproduced, even though Muffett hadn't updated the source code after 2021. But most improvements or changes lie in more recent versions of this proxy so this wasn't considered as a problem. Most modifications for Muffett's DoHoT project centered on modifying the dnscrypt-proxy.toml configuration file which was further modified for the purpose of this thesis.

First, a Raspberry Pi 4 with 8 gigabytes of RAM was deemed suitable device for DoHoT implementation to be tested with. This way clients could pass their Do53 queries for the proxy server which then in turn would make the actual DoH queries over Tor, either to Cloudflare hidden .onion service or to other DoH public DNS servers found on DNSCrypt projects public resolver list [19].

A local firewall, acting also as a router and DHCP server, was used to reserve a static address for the Raspberry Pi. A standard lite version of Raspberry Pi OS was installed to a very fast 200 GB memory card and standard packages were updated and basic configuration done. Internet connection from the local network was 1 Gbps to both directions with latency to 1.1.1.1 as less than 3 milliseconds. Configuration were done mostly according to instructions provided by Muffett with some modifications on versions to install and changing parameters in configuration files [15]. Next, tor was installed using following command.

```
sudo apt install tor
```

Tor was then configured by uncommenting the line in /etc/tor/torrc.

```
SocksPort 9050 # Default: Bind to localhost:9050 for local connections.
```

A local ufw firewall was then set to allow connections only from hosts in local networks.

At the time of configuring the system, 2.1.2 was the latest version of DNSCrypt-proxy available for installation. This was installed with the following commands.

```
sudo -i # get to root; all further commands must run as root
wget https://github.com/DNSCrypt/dnscrypt-
proxy/releases/download/2.1.2/dnscrypt-proxy-linux_arm64-2.1.2.tar.gz
tar xzvf dnscrypt-proxy-linux_arm-2.1.2.tar.gz
tar xzvf dnscrypt-proxy-linux_arm64-2.1.2.tar.gz
rm dnscrypt-proxy-linux_arm64-2.1.2.tar.gz
mv linux-arm64/ dnscrypt-proxy/
```

Next, we had to configure the most important configuration file, dnscrypt-proxy.toml.

```
cd dnscrypt-proxy/
nano dnscrypt-proxy.toml
```

The file opened in text editor was then modified in the following parts, compared to the configuration Muffett provides in GitHub [15].

```
cache_size = 262144
```

This was done to achieve better real-world performance of the DNS proxy as the default value of 4096 was deemed to be too low as there was resources to spare. Also a mistake was first made, by configuring ODoH servers as the target resolvers, that first made the proxy to select only Cloudflare .onion hidden relay as the next hop relay. This was immediately recognized and resolved once the service was running and systemctl provided the status. After this mistake, Muffett's configuration for proxy was used, with the exception of a bigger cache size.

Following commands were then run to start dnscrypt-proxy as a system service.

```
chown -R root: ./
./dnscrypt-proxy -service install
./dnscrypt-proxy -service start
```

The Raspberry Pi installation was then forced to use itself for nameservice by modifying /etc/dhcpcd.conf file.

After these commands, it was then verified that proxy was running correctly with a status command. The output is also provided, showing 53 servers as possible resolvers.

```
systemctl status dnscrypt-proxy.service
...
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: -   536ms
doh-ibksturm
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: -   545ms
nextdns
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: -   547ms
doh.ffmuc.net-2
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: -   578ms he
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: -   632ms
jp.tiar.app-doh-ipv6
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: -   736ms
sby-doh-limotelu
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: -   928ms
quad101
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: -   976ms
publicarray-au2-doh
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: Server with
the lowest initial latency: doh-crypto-sx-ipv6 (rtt: 118ms)
Nov 21 18:28:35 raspi-DoHoT-DNSCrypt-test dnscrypt-proxy[252410]: dnscrypt-
proxy is ready - live servers: 53
```

Next, a Ubuntu 22.04 client was installed to a local Hyper-V server and configured to use this DNSCrypt-proxy installation as the DNS server. This worked instantly, as the DNSCrypt-proxy accepts Do53 queries and the queries are then translated to DoH queries forwarded to resolvers over Tor.

The overall configuration was rather simple after the mistake of configuring ODoH servers as the DNS relay server targets was spotted from the systemctl logs.

4.2 Implementing μODNS relay server and client proxy

Kurihara provided the necessary code base in his GitHub pages for the client module, responsible of translating the queries from the client computer to PoC μODNS queries, then relayed further to the dedicated relay. Installation to a local network environment was first considered, but lack of similar resources as with DoHoT implementation forced the choice to cloud service providers. Linode Dedicated virtual private server with 2 cores and 4 GB of RAM was selected for the client test server and for the relay server. Both were located in the same Frankfurt datacenter. A local VLAN was also established but then later deemed unnecessary for relaying μODNS queries.

Ubuntu 22.04 was picked as the operating system for client and also for the relay server. Both were secured with root login disabled, firewall configured to drop all

incoming connections with the exceptions of SSH allowed from the internet and also allowing traffic to all ports from the IP addresses on the other server (local VLAN address and the public address).

The configuration of the PoC encrypted-dns-server for μODNS was first achieved. Kurihara provided the basic template in his GitHub page for the configuration [18]. This configuration lacked many of the finer details on how to get the implementation running. Latest version of Rust language was installed with commands:

```
curl -sSf https://sh.rustup.rs | bash -s -- -y --default-toolchain nightly
source $HOME/.cargo/env
```

Next, multiple different methods were tested for actual encrypted-dns-server-modns installation. Reading from the Cargo's man page, the correct command to install Git packages was found:

```
cargo install --git https://github.com/junkurihara/encrypted-dns-server-
modns.git
```

Example configuration file was then searched for and modified.

```
cd /home/dnstestdev/.cargo/git/checkouts/encrypted-dns-server-modns-
35b4b6aa0f28bcbd/30fd556/
cp example-encrypted-dns.toml encrypted-dns.toml
nano encrypted-dns.toml
```

Modification instructions were provided by Kurihara in his GitHub repository [18]. Following modifications were then made (--- being used as a delimiter, as the configuration file was long) :

```
listen_addrs = [{ local = "0.0.0.0:443", external = "172.104.133.241:443" }]
---
cache_capacity = 500000
---
provider_name = "myoblivious.testingthesisproj.dump"
---
type = "prometheus"
listen_addr = "0.0.0.0:9100"
path = "/metrics"
```

The last part of the changes provided above was to enable local Prometheus instance to provide metrics of μODNS queries. Otherwise, a larger cache was

desired for better performance and default names and IP addresses were changed.

Then a real problem persisted that seemed to halt the whole project of testing the server implementation. For some reason still unknown, the Rust binary file was unable to start. It was correctly prepared with commands:

```
strip ~/.cargo/bin/encrypted-dns-modns
cargo build
```

The error given was always:

```
Error: No such file or directory (os error 2)
```

Rust, the cargo packages and even the whole server was reinstalled. For some reason on the third reinstall of cargo, the command providing also debug messages, worked:

```
sudo RUST_LOG=debug target/debug/encrypted-dns-modns& --config=encrypted-
dns.toml
```

The following debug log was then printed, showing a working implementation:

```
State file [encrypted-dns.state] found; using existing provider key
[2023-03-14 15:10:03] [INFO] - Public server address: 172.104.133.241:443
[2023-03-14 15:10:03] [INFO] - Provider public key:
b6897a2c6fae9a055008eb255abb27afe9c26c7afd5795499a659ead8f9069e1
[2023-03-14 15:10:03] [INFO] - Provider name: 2.dnscrypt-
cert.myoblivious.testingthesisproj.dump
[2023-03-14 15:10:03] [INFO] - DNS Stamp:
sdns://AQcAAAAAAAAAEzE3Mi4xMDQuMTMzLjI0MTo0NDMgtol6LG-umgVQCOslWrsnr-
nCbHr9V5VJmmWerY-
QaeEyMi5kbnNjcnlwdC1jZXJ0Lm15b2JsaXZpb3VzLnRlc3Rpbmd0aGVzaXNwcm9qLmR1bXA
[2023-03-14 15:10:03] [INFO] - DNS Stamp for Anonymized DNS relaying:
sdns://gRMxNzIuMTA0LjEzMy4yNDE6NDQz
```

After a working relay server was established, a client server making PoC μODNS queries was configured.

Latest version of Go programming language was first installed according to the instructions on project installation documents [21].

Kurihara’s fork for DNSCrypt-proxy was then cloned:

```
git clone https://github.com/junkurihara/dnscrypt-proxy-modns.git
```

Necessary modifications were made to example-dnscrypt-proxy.toml file which was then saved as dnscrypt-proxy.toml to work as the setting file for the proxy. Most important changes were to set the nexthop route as the server configured in the earlier part. This was achieved by appending following to mu-ODNS part of the configuration:

```
routes = [
  { server_name = '*', via = [
    # PoC uODOH relay in Akamai Cloud right next to this client
    { stamp = 'sdns://gRMxNzIuMTA0LjEzMy4yNDE6NDQz', nexthop = true }
```

The correct stamp added here was generated earlier when the relay server binary was started.

To route all local DNS queries to the proxy, it was deemed necessary to modify /etc/resolv.conf system file to the following value:

```
nameserver 127.0.0.1
options edns0
```

Next step was to start the configured local proxy:

```
$ sudo ./dnscrypt-proxy&
[2023-03-14 15:47:38] [NOTICE] dnscrypt-proxy 2.1.4
[2023-03-14 15:47:38] [NOTICE] Network connectivity detected
[2023-03-14 15:47:38] [NOTICE] Now listening to 127.0.0.1:53 [UDP]
[2023-03-14 15:47:38] [NOTICE] Now listening to 127.0.0.1:53 [TCP]
[2023-03-14 15:47:38] [NOTICE] Source [public-resolvers] loaded
[2023-03-14 15:47:38] [NOTICE] Source [relays] loaded
[2023-03-14 15:47:38] [NOTICE] Anonymized DNS: routing everything via
[{sdns://gRMxNzIuMTA0LjEzMy4yNDE6NDQz true}]
[2023-03-14 15:47:38] [NOTICE] Anonymized DNS: relay randomization turned on
[2023-03-14 15:47:38] [NOTICE] Anonymized DNS: max relays = 3, min relays = 1
[2023-03-14 15:47:38] [NOTICE] Anonymized DNS: nexthop relay is chosen from
specific set of relays
[2023-03-14 15:47:38] [NOTICE] Anonymized DNS: protocol v2 is used
[2023-03-14 15:47:38] [NOTICE] Firefox workaround initialized
[2023-03-14 15:47:38] [NOTICE] Anonymizing queries to [saldns01-conoha-ipv4]
via relays [[sdns://gRMxNzIuMTA0LjEzMy4yNDE6NDQz]]
[2023-03-14 15:47:38] [NOTICE] [saldns01-conoha-ipv4] OK (DNSCrypt) - rtt:
240ms
[2023-03-14 15:47:38] [NOTICE] Anonymizing queries to [saldns03-conoha-ipv4]
via relays [[sdns://gRMxNzIuMTA0LjEzMy4yNDE6NDQz]]
[2023-03-14 15:47:38] [NOTICE] [saldns03-conoha-ipv4] OK (DNSCrypt) - rtt:
243ms
[2023-03-14 15:47:38] [NOTICE] Anonymizing queries to [saldns02-conoha-ipv4]
via relays [[sdns://gRMxNzIuMTA0LjEzMy4yNDE6NDQz]]
[2023-03-14 15:47:39] [NOTICE] [saldns02-conoha-ipv4] OK (DNSCrypt) - rtt:
239ms
[2023-03-14 15:47:39] [NOTICE] Sorted latencies:
[2023-03-14 15:47:39] [NOTICE] -   239ms saldns02-conoha-ipv4
[2023-03-14 15:47:39] [NOTICE] -   240ms saldns01-conoha-ipv4
[2023-03-14 15:47:39] [NOTICE] -   243ms saldns03-conoha-ipv4
[2023-03-14 15:47:39] [NOTICE] Server with the lowest initial latency:
saldns02-conoha-ipv4 (rtt: 239ms)
[2023-03-14 15:47:39] [NOTICE] dnscrypt-proxy is ready - live servers: 3
```

As can be seen from the command output, the proxy was able to connect through the configured dedicated relay server in the same Frankfurt datacentre.

5 Performance results

Performance for both implementations was tested by creating a bash script that tested name resolution latency by running dig command for the top 999 domains available and listed in GitHub by a GitHub user called bejaneps [20]. One domain was lost from the top 1000 list due to a configuration mistake but 999 top domains were deemed more than enough.

The script with the omission of the full list of domains was as follows:

```
#!/bin/bash

# Top 999 websites being queried
domains=("fonts.googleapis.com" "facebook.com" "twitter.com"...)

# Create a CSV file and write the header
csv_file="dig_times.csv"
echo "Domain,Time" > "$csv_file"

# Initialize min, max, and total_time
min_time=""
max_time=""
total_time=0

# Query all the websites
for i in "${domains[@]:0:999}"; do
  # Run the "dig" command and extract the time
  time=$(dig "$i" | grep "Query time" | awk '{print $4}')

  # Write the results to the CSV file
  echo "$i,$time" >> "$csv_file"
  echo "Processed $i - Time: $time"

  # Update min_time, max_time, and total_time
  if [ -z "$min_time" ] || [ "$time" -lt "$min_time" ]; then
    min_time="$time"
  fi
  if [ -z "$max_time" ] || [ "$time" -gt "$max_time" ]; then
    max_time="$time"
  fi
  total_time=$((total_time + time))
done

# Calculate the mean time
mean_time=$(echo "scale=2; $total_time / 999" | bc)

echo "Results saved in $csv_file"
echo "Minimum time: $min_time"
echo "Maximum time: $max_time"
echo "Mean time: $mean_time"
```

For DoHoT implementation this provided the results of minimum time for a domain name query to complete as 80 milliseconds, maximum time as 4220 milliseconds and most importantly, mean time for all the queries as 311,61 milliseconds. This can be compared to queries made by traditional Do53 systems where the domain name query can complete in less than 10 milliseconds as achieved by the local fast test Ubuntu client by using another traditional Do53 relay deployed on the local network.

μODNS was devised partly for the purpose to provide better performance than DoHoT, but the results aren't much better when comparing the mean time for domain name query to complete. The test system achieved a mean query resolution completion in 339,80 milliseconds, with one outlier result, namely raw.githubusercontent.com being in the local cache and giving a result of 0

millisecond as the lowest value. A maximum query completion value was 3968 milliseconds which is quite comparable to DoHoT implementation.

It is necessary to remember however, that the PoC μODNS was relying on the three servers hosted in Japan where the queries were passed from the client and through the relay located in Frankfurt. This provided the additional latency of around 240 milliseconds. This should be circumvented by installing an actual μODNS server (or rather servers) geographically closer to the dedicated relay. As of this day these three servers in Tokyo remain the only public accessible public μODNS compatible target relays. The timeframe and resources available limited this thesis to implement a dedicated relay and client proxy only.

When comparing the performance of the test system implemented, to the one deployed by Kurihara and Kubo [4, p. 14], it can be seen the performance of deployed test system was even better when omitting the geographical distance that introduced the additional 240 ms latency. This isn’t something not to expected, as the codebase for both Go and Rust programming languages have seen multiple versions compared to their state 2021 when Kurihara and Kubo released the original paper. Also, the DNSCrypt-proxy employed is newer version. It can be extrapolated that a μODNS system could achieve under 100 ms average resolution time if correct locations for servers were chosen.

6 Feasibility and conclusion

One of the main motifs for μODNS was better performance for queries, but the DoHoT system performed almost to the same level when comparing the mean values. As mentioned, this was only possible because of the non-ideal implementation for μODNS.

As one of the goals of this thesis was to provide a guideline of feasibility of the implementation, DoHoT fared a lot better. A modern laptop running Windows 11 Pro was used with this DoHoT implementation configured as the DNS server for over three months. Streaming services were used, some multiplayer games played, code developed and internet was browsed just as these things were done in another computer in local network, which used DoH queries to dns.adguard-dns.com. In real world, slower page loads were given on almost all webpages but after a few minutes of browsing it wasn't bothering as much as in the start. Video streaming wasn't affected at all, just the page load times as discussed earlier. Quite often the large DNS cache in Raspberry Pi was able to provide the DNS resolutions almost instantly. A large DNS cache was deemed necessary in cases where constant DNS queries were made.

When considering the ease of implementation, the DoHoT system was clearly the more stable one, not requiring the installation of latest nightly builds of Go or Rust and easily starting on the first try and remaining stable for extended periods of many months. It also required the configuration of only one server. A full implementation including multiple μODNS relays would be feasible in corporate environment where there was necessary resources for a proper implementation for servers located relatively close to each other geographically. Stability would need to improve a lot, as Kurihara mentions in the GitHub page for the relay and proxy, both are PoC currently. For production environment this is almost always unacceptable.

As for the query privacy and possible collusion or wiretapping (and thus timing attacks), it's hard to say if such can be yet achieved on μODNS implementation

at least. As almost all μODNS traffic is centred on the three μODNS servers in Tokyo, it is most likely that queries relayed there are easily recognizable from the rest of internet based on multiple metrics. This could only be defeated by increasing the amount of clients making such queries. By making it easier to deploy test proxies and dedicated relays, it is likely that other parties would start to implement public μODNS capable relays on large. However, there is always the possibility of newer and better standards to appear and take hold.

If the relatively slower performance is acceptable, then DoHoT capable proxy can easily be deployed. This approach brings the huge benefit of actually masking the queries to rather much wider network, Tor. Tor is built with privacy-first mindset and DoHoT queries are rather well masked in all the other traffic flowing through Tor. There is even Cloudflare’s .onion service inside Tor network that can be reached, were the access to DoH servers monitored and blocked.

References

1 Mockapetris, P. 1983. Domain names: Concepts and facilities. RFC 882. DOI 10.17487/RFC0882. <https://www.rfc-editor.org/info/rfc882>.

2 Eastlake 3rd, D. 1999. Domain Name System Security Extensions. RFC 2535. DOI 10.17487/RFC2535. <https://www.rfc-editor.org/info/rfc2535>.

3 Schmid, G. 2021. Thirty Years of DNS Insecurity: Current Issues and Perspectives. IEEE Communications Surveys and Tutorials 23, no. 4 (2021): 2429-2459. <https://doi.org/10.1109/COMST.2021.3105741>.

4 Kurihara, J.,Takeshi K. 2021. Mutualized Oblivious DNS (μODNS): Hiding a Tree in the Wild Forest. <https://doi.org/10.48550/arxiv.2104.13785>.

5 Mockapetris, P. 1987. Domain names - implementation and specification. STD 13. RFC 1035. DOI 10.17487/RFC1035. <https://www.rfc-editor.org/info/rfc1035>.

6 Hoffman, P., Sullivan, A., Fujiwara, K. 2019. DNS Terminology. BCP 219. RFC 8499. DOI 10.17487/RFC8499. <https://www.rfc-editor.org/info/rfc8499>.

7 Isasi, S., Shrestha, V. 2018. Expanding DNSSEC Adoption. Online. Cloudflare. <https://blog.cloudflare.com/automatically-provision-and-maintain-dnssec/>. Accessed January 21, 2022.

8 Vcelak, M., Goldberg, S., Papadopoulos, D., Huque, S., Lawrence, D. 2019. NSEC5, DNSSEC authenticated denial of existence. Internet-Draft draft-vcelak-nsec5-08. Online. IETF. <https://www.ietf.org/archive/id/draft-vcelak-nsec5-08.txt>. Accessed January 22, 2023.

9 Kinnear, E., McManus, P., Pauly, T., Verma, T., Wood, C. 2022. Oblivious DNS over HTTPS. RFC 9230. DOI 10.17487/RFC9230. <https://www.rfc-editor.org/info/rfc9230>.

10 Singanamalla, S., Verma, T. 2020. Improving DNS Privacy with Oblivious DoH in 1.1.1.1. Online. Cloudflare. <https://blog.cloudflare.com/oblivious-dns/>. Accessed November 29, 2022.

11 Schmitt, P., Edmundson, A., Mankin, A., Feamster, N. Oblivious DNS: Practical privacy for DNS queries. Proceedings on Privacy Enhancing Technologies; 2019 (2):228–244. Stockholm, Sweden. Feb. 2019. Online. <https://www.petsymposium.org/2019/files/papers/issue2/popets-2019-0028.pdf>. Accessed January 29, 2023.

12 Muffett, A. 2021. "No Port 53, Who Dis?" A Year of DNS over HTTPS over Tor. Online. <https://www.ndss-symposium.org/wp-content/uploads/dnspriv21-03-paper.pdf>.

13 Sayrafi, M. 2018. Introducing DNS Resolver for Tor. Online. Cloudflare <https://blog.cloudflare.com/welcome-hidden-resolver/>. Accessed March 5, 2023.

14 Siby, S., Juarez, M., Diaz, C., Vallina-Rodriguez, N., Troncoso, C. 2019. Encrypted DNS --> Privacy? A Traffic Analysis Perspective. Network and Distributed Systems Security (NDSS) Symposium 2020. <https://dx.doi.org/10.14722/ndss.2020.24301>.

15 Muffett, A. DoHoT: making practical use of DNS over HTTPS over Tor. Online. <https://github.com/alecmuffett/dohot>. Accessed March 6, 2023.

16 DNS over Tor. Online. Cloudflare. <https://developers.cloudflare.com/1.1.1.1/other-ways-to-use-1.1.1.1/dns-over-tor>. Accessed 17 March, 2023.

17 Kurihara, J. A forked version of dnscrypt-proxy for µODNS. Online. <https://github.com/junkurihara/dnscrypt-proxy-modns>. Accessed 17 March, 2023.

18 Kurihara, J. A forked version of encrypted-dns-server for µODNS. Online. <https://github.com/junkurihara/encrypted-dns-server-modns>. Accessed 17 March, 2023.

19 Anonymous. Online. DNSCrypt. <https://raw.githubusercontent.com/DNSCrypt/dnscrypt-resolvers/master/v3/public-resolvers.md>. Accessed 17 March, 2023.

20 benjaneps. Online. GitHub. <https://gist.githubusercontent.com/bejaneps/ba8d8eed85b0c289a05c750b3d825f61/raw/6827168570520ded27c102730e442f35fb4b6a6d/websites.csv>. Accessed 17 March, 2023.

21 Download and install. Online. Go Project. <https://go.dev/doc/install>. Accessed 17 March, 2023.