EFFECTS OF SECURED DNS TRANSPORT ON RESOLVER PERFORMANCE

Etienne Le Louët[1,2], Antoine Blin[2], Julien Sopena[2], Kamel Haddadou[1], Ahmed Amaou[1]
[1]GANDI, Paris, France, [2]Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

NOTE: Corresponding author: Etienne Le Louët, etienne.lelouet@gandi.net

Abstract *– Designed 40 years ago, DNS is still a core component of the Internet: billions of DNS queries are processed each day to resolve domain names to IP addresses. Originally designed for performance and scalability, its transport protocol is unencrypted, leading to security flaws. Recently, secure protocols have emerged, but the question of their scalability and sustainability remains open. In this paper, we study the cost of switching from the legacy DNS transport to the newer ones, by first characterising the shape of the traffic between clients and secured public resolvers. Then we replicate said traffic, to measure the added cost of each protocol. We found that, while connections usually stayed open, many closures and openings were made in some cases. Comparing these profiles over different DNS transports, we observe that switching from the legacy protocol to a more secure one can lead to an important performance penalty.*

Keywords – DNS, DOH, DOT, HTTP/2, resolver, TLS

1. INTRODUCTION

Introduced in 1983, the dns is a core component of the Internet, as nearly every communication on it is preceded by at least one DNS query, to transform a human-readable domain name into an Internet Protocol (IP) address. Nowadays, DNS is used for much more than name-to-IP address translation: it can hold mailbox data, x.509 certificates, or configuration information for various services. It had originally been developed with a focus on performance and scalability by using the udp as its transport protocol to achieve both the lowest latency and server load, but concerns regarding confidentiality and integrity have since emerged. New standards, dot [1] and doh [2], have been proposed within the ietf to secure it, by encrypting queries and responses using the tls protocol. While these new standards provide both confidentiality and integrity, the question of their cost remains open, as there is no information on the energy or environmental sustainability of transitioning all DNS traffic from the old, unsecured protocol to the new ones. In this paper, we propose an estimation of the additional server resources required to transition from a non-encrypted, non-connected, DNS protocol to a secure but costly protocol, by first observing how existing secured DNS clients use the service, and then measuring the added cost of these protocols in a controlled environment.

First, we conducted a characterization of the behaviour of DoH clients and public resolvers, in order to gather the different patterns and settings applied by both entities in their use of the secured protocols (number of connection openings, connection duration, number of queries allowed per connection...). We noticed that, while they tend to try and keep a single connection alive, browsers can, in certain cases close and re-open them very frequently.

Then, we realised multiple benchmarks using two secured resolver implementations in order to, first, compute their performance baseline when using the unconnected legacy UDP protocol, then to measure the hardware resources consumption of each of the steps (connection establishment and upkeep, as well as message processing) added by the new DNS protocols. We observe that while the additional memory consumption generated by the use of the new secured protocols is noticeable, the rate of increase is not important enough to be a problem in terms of scalability. In terms of Central Processing Unit (CPU) overhead, transitioning from UDP to DoH can lead to a 70% decrease in performance, for relatively long-lived Transmission Control Protocol (TCP) connections. The cost of encryption is in large part added by the tls key exchange. For DoH (the most popular protocol) the cost of implementing the http2 layers lead to a performance penalty.

© International Telecommunication Union, 2024
Some rights reserved. This work is available under the CC BY-NC-ND 3.0 IGO license: https://creativecommons.org/licenses/by-nc-nd/3.0/igo/.
More information regarding the license and suggested citation, additional permissions and disclaimers is available at:
https://www.itu.int/en/journal/j-fet/Pages/default.aspx

The rest of the paper is organised as follows: Sections 2 and 3 describe the technical background and related work, Section 4 proposes a characterization of client and resolver behaviour in order to understand the shape of the traffic near the resolvers, Section 5 proposes a benchmark and an analysis of server side performance and in Section 6, we conclude.

2. TECHNICAL BACKGROUND

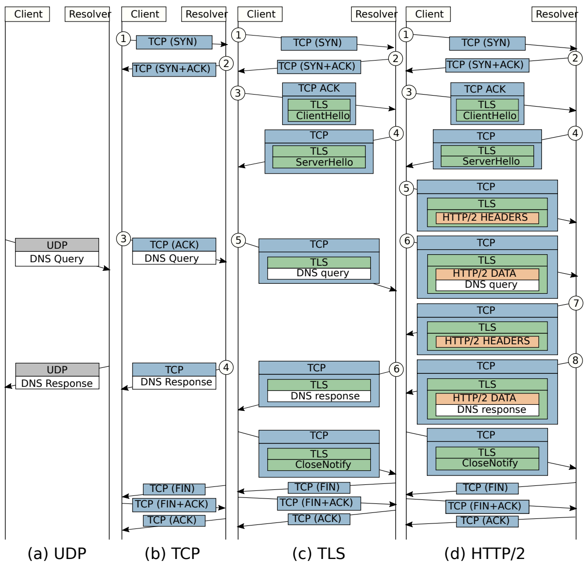


Fig. 1 – Comparison of DNS over UDP, TCP, TLS (DoT) and HTTP/2 (DoH)

From a high-level point of view, DNS is a registry service queried by a client to resolve the IP address corresponding to a domain name. It has been implemented as a tree-like database, in which multiple servers hold only a fraction of the information; looking for information on it is therefore a depth-first search starting from the root. However, if every client looking to translate a domain name to an IP address were to realise such a process, it would result in prohibitive latency and an overload on the servers closest to the root; that is why iterating through the tree is delegated to resolvers, servers which, upon receiving a query from a client, can answer it either from their cache, that will have a higher *hit rate* since it likely received the same request from another client earlier, or by doing the resolution themselves.

In contrast to the other, historically text-based, web protocols, the DNS message protocol has been implemented using binary message format in order to target the highest performances. Both UDP and TCP have been selected as transports. The first one, UDP, is connection-less (Fig. 1a), and provides high performance at the cost of reliability and message payload size. As such, it is the recommended protocol to transport standard DNS queries (which represents most of the DNS traffic). The second one, TCP, requires the exchange of three messages (arrows 1 to 3 on figures 1b c d) to establish a connection, before sending any DNS messages. It has been mostly used to transport special DNS messages (zone transfers) that do not fit into a UDP datagram. As these legacy DNS transport protocols are unsecured, DNS is vulnerable to a variety of attacks, detailed in Section 3. Several secured protocols have been proposed to deal with the aforementioned security flaws.

The first one, DoT [1], uses a TLS connection [3] to provide both integrity and confidentiality. It relies on a TCP connection to establish a TLS session between the client and the server. Two messages (arrows 3 and 4 in Fig. 1c) are exchanged between both endpoints to derive, from their respective pairs of asymmetric keys, a symmetric key used to encrypt the DNS binary messages. During this process, the client also validates the identity of the resolver by using the latter's digital certificate.

DoH has been proposed as an alternative to offer a secure DNS transport. It relies on HTTP/2 [4] to carry the DNS messages, and may be seen as an additional layer built on top of TLS. Once a session is established (arrows 3 through 4 in Fig. 1), the multiple streams of the HTTP/2 protocol are used to transport DNS queries, either directly in their binary format as the body of an HTTP POST query (arrows 5, 6, 7 and 8 in Fig. 1d) or as the base64-encoded url parameter of a GET query (as it is less common it is not considered here).

Originally designed for performance, the legacy DNS protocol doesn't offer any security guarantees. To prevent data leaks and corruption, new protocols based on existing technologies used to guarantee security on the web have been pushed in order to secure DNS. Switching from a connection-less unencrypted protocol to connected ones making extensive use of cryptography seems to go against the original goals that drove the development of DNS, and therefore the cost of this transition must be analysed.

3. RELATED WORK

We classify the work related to DNS security in three categories: work that focuses on describing and proposing mitigation for different security flaws, work that aims to compare the client-side cost of secured DNS protocols, and finally, work that focuses on evaluating their adoption.

Security issues and guarantees: When studying the security of an information system, three properties are to be considered: confidentiality, integrity and availability.

Confidentiality: UDP and TCP did not offer any kind of confidentiality to the messages they carry, meaning that any malicious actor could use captured DNS messages to breach a user's privacy [5]. DoT and DoH circumvent this flaw by using the TLS protocol to carry their messages, therefore guaranteeing confidentiality. However, several studies have shown that some characteristics of DNS traffic can be exploited to, in some cases, de-anonymize encrypted DNS traffic. In [6], Siby et al. show that, despite its use of encryption, it is still possible to determine the content of a DoH flow containing un-padded queries and responses, by using traffic analysis techniques. In [7], Bushart and Roshow show that even state-of-the-art padding strategies are weak against some traffic analysis attacks. However, it is worth noting that the machine-learning models used in these attacks can only de-anonymize DNS flows they were previously trained on (usually popular websites), and that techniques such as arbitrarily delaying queries and responses, or the use of proxy networks such as tor can be powerful mitigation against these attacks. Furthermore, these attacks require both a constant update of the model used to target websites in order to cope with their modification, and the knowledge of the source of the traffic, as clients have different behaviours regarding inter-query timings and message size.

Integrity: The DNS protocol did not initially offer mechanisms guaranteeing the integrity of data, meaning that an adversary could edit a DNS response, thus redirecting a client towards fraudulent services [8]. DNSSEC was later standardized, and guarantees the integrity of data exchanged between the resolver and name servers. On the other hand, the data exchanges between client and resolver still use the legacy protocol, leaving them vulnerable to the aforementioned attacks. As TLS guarantees the integrity of the messages it transports, using DoT or DoH in combination with a trusted resolver that validates the integrity of records by using DNSSEC, can protect against this category of attacks.

Availability: The two aforementioned properties are necessary but not sufficient to fully protect a client. DNS is one of the most commonly filtered protocols (by governments or isp [9]). DoT, which uses port 853 by default can be easily blocked by port-based filters, while DoH is not, as it relies on a widely used protocol. It is still vulnerable to fingerprinting techniques, able to detect whether or not an encrypted flow contains DoH queries and response, like Vekshin et al. prototyped in [10]. However, as we said earlier, these techniques require models trained on a variety of clients, resolver and traffic shape that require constant updating, so it is unrealistic to expect them to be used globally. Despite the remaining security limitations, the benefits provided by DoT and DoH complete the efforts first undertaken with the introduction of DNSSEC.

Client-side performance: Various studies focus on the client-side cost of DoT or DoH. Hounsel et al. [11], [12] compare the page load times using different combinations of DNS transports, network types and public resolvers. Boettger et al. [13] also compare the resolution times and protocol overhead of different secure DNS transports when using persistent or non-persistent connections. These studies find that connection reuse is beneficial for the client, and that secure DNS adds no noticeable cost to clients, except on some cellular networks.

Chhabra et al. [14] also study the impact of switching to DoH, leveraging several vantage points, which allows them to correlate the speed-up or slow-down measured when switching from doudp to DoH with the level of investment in Internet infrastructure by a country.

Finally, in [15], Kosek et al. compare the performances (in terms of latency) of DoH with those of DoUDP, but also with those of doq. They found that the use of DoQ lead to faster page load times than with DoH, due the faster connection establishment. However, the improvement over DoH lessens as the complexity of the loaded page (and therefore the number of resolutions needed to load it) increases, since the latency gained by a faster handshake is amortized by the connection reuse, which in turns allows both DoH and DoQ (connection-based protocols) to catch up with DoUDP (as discussed by Boettger et al. [13]).

Protocol adoption: In [16] Garcìa et al. analyse both the number of available DNS-over-encryption resolvers, as well as the use of DNS-over-encryption by various users. While the amount of DoH traffic had stayed stationary, representing about 1% of the current DNS traffic, the number of available DoH servers is steadily growing. Such a growth raises the question of how the energy sustainability of the generalized use of DNS over HTTPS

While the work discussed here offers valuable insights on the more recent DNS transport protocols, ours is the only work that focuses on the impact on resolver resource consumption of implementing such protocols; the existing literature focuses on client-side or network-side metrics such as latency or packet number or size, while our work focuses on the memory or CPU consumption of implementing such protocols, on the server-side.

4. BEHAVIOUR OF CLIENTS AND RESOLVERS

As the newer DNS transports are connection-based, new questions arise: while the protocol defines how to query the service, it doesn't specify how the underlying connections should be managed by both the client and resolver. The sequence of connections opening and requests sent is mostly controlled by the client. But to focus only on the client's behaviour is not enough, as the server has the choice to accept, reject, close or keep said connections opened.

The objective of this experiment is to characterize the shape, in terms of number of establishments, closures and messages sent over individual connections, of the traffic between already existing clients and publicly available resolvers. We specifically choose existing clients and public resolvers who are widely used and whose configuration and behaviour we do not control, as we want to determine how the implementations currently deployed use the protocol, so we can generate similar traffic when measuring server-side performances.

In Section 4.1, we describe the experimental setup used for the measurements, while Section 4.2 contains an analysis of the different behaviours observed from the clients and the resolvers.

4.1 Experimental setup

As DoH gained more traction than DoT and is therefore available in more software and on more public resolvers, the only traffic generated in this experiment is DoH traffic. To characterize the shape of the traffic, we make a DoH-enabled client send queries at various rates (one every 50 ms, 1000 ms and 60 000 ms) for 30 minutes, to a DoH-enabled public resolver. We run the client in a docker container for two reasons: the isolation provided by the network name spaces give us a way to isolate its network traffic for capture using Tshark [17], and using a container allows for easier reproducibility of our experiment. We enforce no resource restrictions on the container we use; therefore, the only overhead is the additional network latency and CPU use due to the more complex network path induced by the use of network name spaces. We then further filter the network traffic based on the target's resolver IP, the port used by DoH (443), and transport protocol (TCP), leaving us with a trace containing only the DoH traffic emitted by the client. We then extract the following metrics from the network trace : the number of TCP connections established between client and resolver, their duration, the origin (client or the server) and method (TCP FIN or RST, HTTP/2 GOAWAY) of its closure. Another metric we consider is the number of queries on each connection. However, as traffic is encrypted, we make our client dump the TLS secrets to a file, in the nss Key Log Format, so that Tshark can decipher the traffic, allowing us to count the exact number of queries and responses exchanged in every TCP connection.

Part of the reason for DoH's popularity is its integration in popular web browsers. For this reason, we elected to observe the behaviour of Chromium [18], as it is the basis for other popular browsers [19], such as Chrome or Edge. We also chose to observe Firefox's [20] behaviour, as it was among the first browsers to implement a DoH stub resolver.

Even though web browsers are likely to be one of the biggest source of DNS traffic, there is other software on an end-user's machine that can generate DNS traffic as well, and, in the majority of cases, traffic is unsecured, as it relies on the stub resolver provided by the host OS. A new category of software, called proxies, has emerged to resolve this issue. They run on the user's device, listening on a local port, and are configured as the system's resolver, meaning that they capture all DNS queries emitted by software on the system, and transmit them to a configured resolver over a secured channel. This means that, by installing and configuring it, all software transparently benefits from a secured DNS channel to a trusted resolver, that is shared among all running applications, which saves both client and server resources when compared to a model in which each client individually implements secured DNS transports.

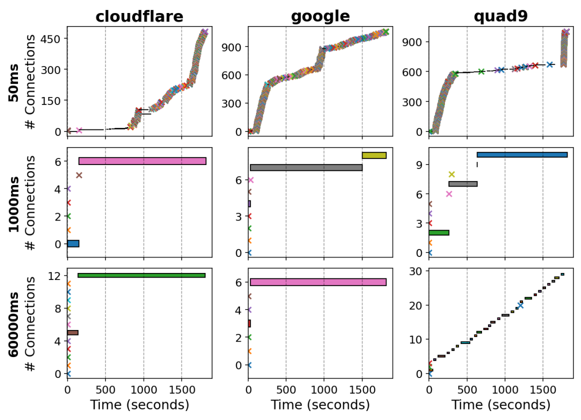
As the goal of this experiment is to measure how both parties use the underlying TCP connections, we need to measure how current DNS over HTTPS deployments handle traffic. This is why we chose to observe the behaviour of three widely used, DoH-enabled public resolvers : Quad9, Google and Cloudflare. Since we have three different clients, delays and resolver, this brings the total amount of experiments we run to test every combination to 27.

We gather the list of domain names to resolve from a public list of domain names [21], filtered to keep the ones that still have an A record corresponding to a server accepting HTTP traffic. In order to generate the appropriate traffic using the browsers, we configure them to use the selected DoH resolver, and we loaded a JavaScript script making HTTP HEAD requests to the domain names in the file. We choose HTTP HEAD request, as they require the browser to resolve the domain name to contact the server, but cause very little data to be returned. To generate traffic using DNScrypt-proxy, we use a C program making DNS resolutions for the domain in the list at the same rates as the ones configured for the browser, using the system's configured resolver, which, in this case is DNSCrypt-proxy.

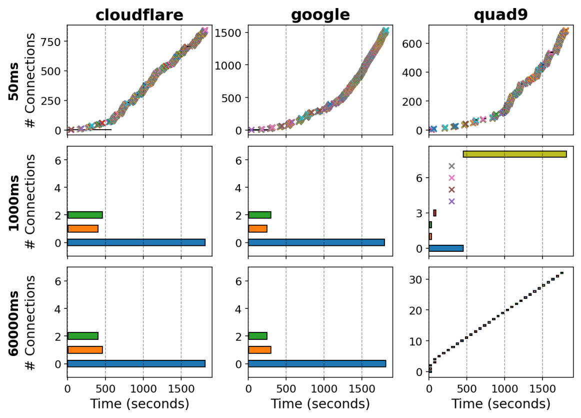
Both these scripts, in addition to the configuration files for the browsers, DNSCrypt-proxy, as well as the scripts used to analyse the traffic, are available in a public git repository [22].

4.2 Results

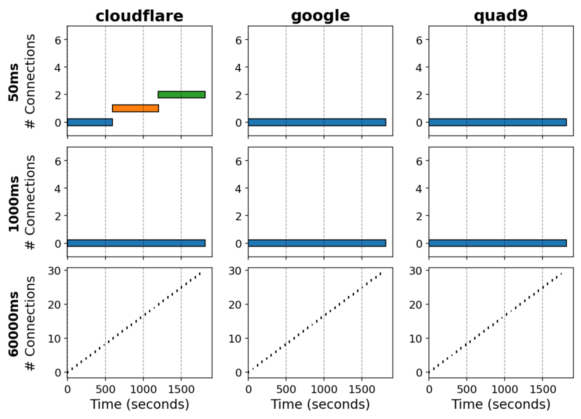
Figures 2a, 2b, 2c present, for each combination of software, inter-query delay and public resolver, the number and length of connections to the resolver established by the client. For example, on Fig. 2c, the top-left figure presents the number and length of TCP connections that DNSCrypt established towards the Cloudflare resolver. Each sub-figure can be seen as a Gantt diagram : the x-axis represents the time in



(a) Client : Firefox



(b) Client : Chromium



(c) Client : DNSCrypt-proxy

Fig. 2 – Connection use by clients (Firefox, Chromium and DNSCrypt-proxy) for a set of resolvers (Cloudflare, Google and Quad9) and query delays (50 ms, 1 000 ms, 60 000 ms)

the experiment, and every connection is represented on a single line as a coloured rectangle, its leftmost and rightmost edges marking its start and end date respectively. Connections shorter than a second are represented by a cross. For example, by observing the top-right graph of Fig. 2c, we can see that, when the inter-query delay is 50 ms, DNScrypt-proxy established only one connection to the quad9 resolver.

Conversely, by observing the top-right graph of Fig. 2a, we observe that, when faced with the same inter-query delay of 50 ms, Firefox established a lot of short-lived (less than 1 s) connections to the quad9 resolver.

DNScrypt-proxy: DNScrypt-proxy generates the least aggressive load towards the server. Indeed, its main behaviour is to open and keep open a single TCP connection that it will use to perform all requests, regardless of intensity of the traffic generated. We can infer this by observing the middle row two top-rightmost graphs on Fig. 2c. In addition, an internal timer is set to trigger the close of the TCP connection 5 seconds after the last message was sent over it, freeing both client and server resources. We infer this by observing the bottom row on Fig. 2b, or the network trace, in which we see that the TCP connection closure is always initiated by DNSCrypt-proxy, through a TLS alert message with the "Close Notify" description.

Web browsers: The browsers have a more aggressive usage of DNS resources. At the beginning of the sessions, they try to maximise the probability of having a successful connection to the DNS server by opening several connections in parallel to the same server, likely to speed up the early resolutions that browsers usually do, (figures 2a and 2b), leading to an increase in server resources usage. The following use of these opened connections depends on the intensity of the traffic. When the traffic has a low intensity, with a request frequency lower than 1 Queries Per Second (QPS), (see the bottom two rows of figures 2a and 2b), a single connection is mainly used to handle the traffic, the remaining connections eventually being closed. We sometimes observe connection closures, forcing a re-opening (1 000 ms delay row on figures 2b and 2b), or multiple connections at the same time (1 000 ms and 60 000 ms delay rows on Fig. 2b), but these events are not numerous enough during the lifetime of an experiment to be significant. Under a DNS traffic with a high intensity (above 1 query per second) the connection pattern of the web browsers changes drastically. Not only does the browser fail to generate the traffic we ask for, we also observe connections being opened and closed in sequence, (see top row on figures 2a and 2b). Every connection shutdown originates from the client, either through an HTTP/2 GOAWAY message for Firefox, or directly through a TCP FIN message for Chromium.

Each of these connections is used to carry few to no DNS messages. From a server perspective, such behaviour represents the worst case, as, with each connection opening being costly, this leads to huge resource consumption.

Resolvers: Clients are not solely responsible for the connection patterns. The resolvers have the choice to accept or deny the connections and the traffic issued from the clients. We have observed that Google has the most permissive resolver configuration of those we tested, as we didn't observe any limitation in terms of number of connections, their duration and the number of QPS per connection. Quad9 closes unused connections through an HTTP/2 GOAWAY message after around 30 seconds of inactivity (Fig. 2a, bottom-right). Cloudflare does not impose any restriction on the connection duration, but limits the maximum number of requests per connection to 10 000 (Fig. 2b, top-left). When this limit is reached, the server notifies the client through an HTTP/2 GOAWAY message paired with a TLS alert message with the "Close Notify" description.

The intended behaviour of clients and resolvers seems to be to keep one TCP/TLS connections alive while they are used, as re-opening a connection leads to an increased cost in CPU resources (as the TLS handshake is relatively costly), or in latency (as each connection establishment requires multiple round-trips).

5. SERVER SIDE PERFORMANCE

Moving from UDP, an unconnected protocol historically used for communication between clients and resolvers, to more complex connected ones can lead to an increase in the consumption of hardware resources on the resolver side: the handling, by the resolver, of DNS queries transported in a UDP datagram simply requires receiving the datagram and then sending another one containing the answer once the cache-lookup or resolution is completed. On the other hand, the use of session-based protocols is more complex, as they require multiple round trips and additional computations for the establishment of a session, the management of the state associated with the said session, the encoding and decoding of messages, in addition to the already-existing cost of handling DNS queries.

Questions about scalability and resource consumption arise regarding the cost of these additional steps. In order to properly evaluate their cost, we realised a series of synthetic benchmarks, first using DNS over UDP (UDP) as a baseline, then DNS over TCP (TCP), DNS over TLS (DoT), and DNS over HTTPS (DoH). While it offers no privacy guarantees, measuring how TCP performs is still interesting because, as we have seen previously in Section 2, both secured protocols have been built on top of TCP. Thus, the comparison between UDP and TCP is a good performance indicator of the cost added by the TCP connection. Using the same approach, comparing TCP and DoT allows us to measure the performance cost of the TLS session establishment and traffic encryption, and comparing DoT with DoH gives us insights about the cost of the added HTTP/2 layers.

In Section 5.1, we describe the experimental environment of our benchmarks. Section 5.2 presents the results of a benchmark of the legacy UDP-based protocol, while sections 5.3, 5.4 and 5.5 describe the multiple synthetic benchmarks we realised to characterize the costs of the different steps of the connection-based protocols.

5.1 Experimental setup

We elected to run our benchmarks on a DNS architecture deployed on the Grid5000 [23] platform. Our testbed is composed of 22 Dell PowerEdge R640, each of them with an 18-core CPU with a base clock of 2.2 GHz and a turbo frequency of 3.9 GHz, 96 GiB of RAM (Random Access Memory) and a 25 Gbps nic, all connected together through the same switch. We run our server on one of those machines, use twenty of them as our clients and the remaining one as the experiment monitor in charge of deploying and running the various actors and measurement tools on their respective machines.

We selected three secured DNS implementations to test: Knot-resolver [24], as it is used by important industry players (most notably Cloudflare), Dnsdist [25], that is not a server *per se*, but acts as a proxy and load balancer between a client and another server, it can either answer from its cache, or forward the query to another server. Since it is compatible with both DoH and DoT, it can be used to modernise an existing DNS infrastructure by adding support for these protocols without having to make extensive modifications to the underlying server(s) software or configuration. The last implementation we tested is Adguard-Dnsproxy [26], as it is a relatively recent implementation that promises support for newer protocols such as DNS over QUIC or DNS over HTTP/3. As we need a very high number of clients to reach 100% load on one core in our setup, we configure both software to only run on a single core of our server machine using Linux *cgroups*.

As we aim to focus on the server-side cost of transitioning from a legacy UDP-based protocol to a session-based protocol for the client to server connections, we decided to exclude the cost of retrieving the records from the hierarchy of DNS name servers from the resolving process, to avoid measurements noise that could occur when querying external uncontrolled name servers. At the beginning of each experiment, we fill the cache of our servers with the DNS records that will be queried during the experiment, meaning that all subsequent queries from the clients result in a cache hit. To reduce experimental variability as much as possible, all of the names that are queried for during the rest of the experiment are composed of a number from 0 to 2 000, padded to four characters, followed by a non-existing top-level domain.

Traffic is generated using Flamethrower [27], a DNS benchmarking utility compatible with all benchmarked protocols. We patched its code so it would be able to keep the underlying connections opened for a configurable duration, as its default behaviour was to open a connection, send a set number of queries, wait for the answers to these queries (up to a configurable timeout), and then close the connection, only to re-open it again for the next batch of queries. As we wanted to be able to control the length of the connections, our new implementation opens a connection for a set duration, sends batches of queries on that connection at a configurable frequency (waiting for their answer or timeout), and only closes the connection once a configurable timer, separate timer has run out. A fork of Flamethrower including these changes is available on GitHub [28]. When benchmarking DoH, we send our queries in the body of an HTTP/2 POST query, as we detected in Section 4 that it was how clients operated. For all these experiments, we tuned Flamethrower's parameters (number of queries per batch, and delay between each batch), so that our resolver process would end up using 100% of the CPU it was pinned on (as measured by both htop, ps, and confirmed by the output of linux-perf).

5.2 Baseline

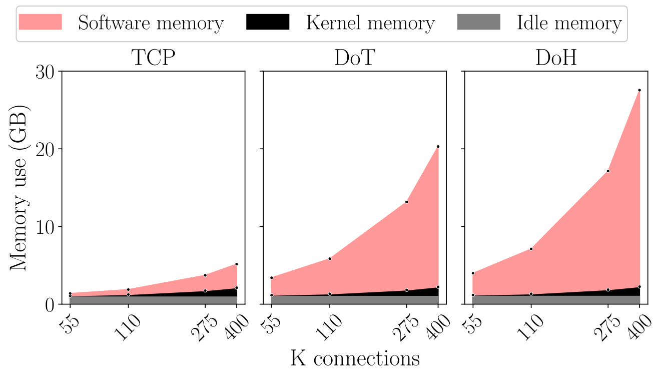
As it is the legacy most widely used and the most efficient transport for DNS, measuring how UDP performs gives us a baseline in terms of performances. To obtain this baseline, we run several experiments in which DNS over UDP queries at a set rate, which increase each experiment. As UDP offers no flow control mechanism, at some point the client will start sending more traffic than the server can answer, meaning that it will not answer some of them, leading to losses. We then take note of the maximum amount of queries per second handled by the server, and use it as our baseline.

At best, Knot-resolver answered 105 000 QPS out of the 120 000 QPS sent by our clients, while Dnsdist answered 223 000 QPS, out of the 240 000 sent. We investigated these losses and noticed that they were due to a saturation of the CPU, both servers being unable to process queries at such a rate, leading to the kernel-side UDP reception buffer filling up and packets having to be discarded. Increasing the size of the reception buffer is useless: since we send queries at a constant rate, it will just delay the moment the buffer fills up and the kernel starts discarding packets. We explain the difference in performances of almost 50% between Knot-resolver and Dnsdist by the fact that Dnsdist is a proxy and load balancer whose purpose is to pass queries to an upstream server as efficiently as possible, therefore having very few things to do when receiving a query other than answering it from its cache or forwarding it, while Knot-resolver most likely has to do more processing, even in the case of a cache hit (query policy, response padding).

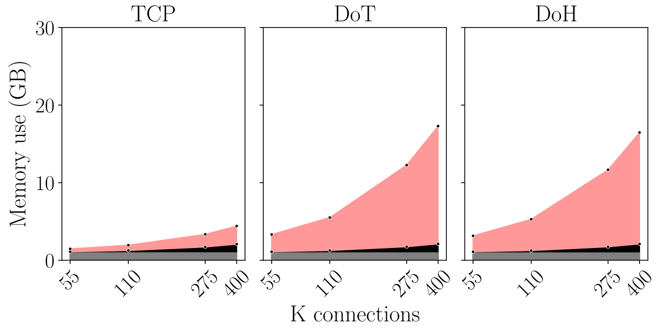
5.3 Memory usage of keeping connections alive

Transitioning from UDP, a connection-less protocol, to connection-based ones raises the question of the maximum number of simultaneous connections a server can handle. Therefore, we have devised an experiment aiming to measure the limits (in terms of memory) of the number of connections that can be handled by a server.

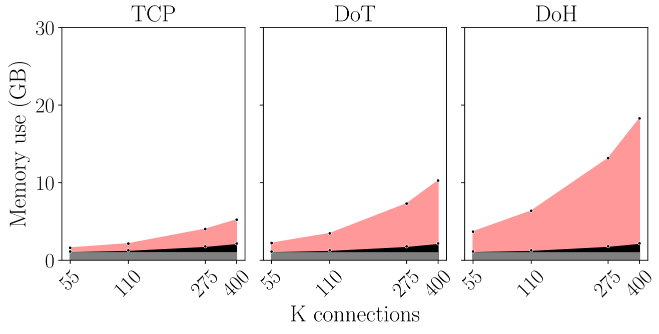
For each protocol we tested (TCP, DoT and DoH), we used Flamethrower to generate as many connections towards our server as possible, spread across our 20 machines. We configured both client and server so that they would not close established connections,



(a) Knot-resolver



(b) Dnsdist



(c) Adguard-dnsproxy

Fig. 3 – Memory usage of the servers relative to the number of connections

and increased the kernel-side limits on the number of outgoing or incoming connections. We measure two separate values: the resident set size (amount of memory used by a process present in physical RAM) of the server, and the total amount of memory used on the machine. By calculating the difference between these two values, we are able to estimate the amount of memory used by the kernel, as the in-physical RAM memory imprint of the processes other than the server is negligible. There was an option in Dnsdist allowing for the release of memory associated with idle connections, which we chose to deactivate as our interest lied in estimating how much memory an active connection would consume.

Fig. 3 shows, for each server and protocol combination, the total physical memory used relative to the number of connections. At the top (in red) is the resident set size of the server, in the middle (in black) is memory used by the kernel, and, at the bottom (in grey) is the amount of memory used when the server is idle, presented for reference. The memory used by the kernel is the same in every experiment, which is consistent with the fact that the kernel only handles TCP connections, the common part between the three protocols. For all servers, we observe an increase in memory consumption when switching from TCP to DoT, as the handling of TLS sessions requires additional state, handled by the server. When considering the difference between DoT and DoH, we can notice, for both Adguard-dnsproxy and Knot-resolver, a clear increase in consumption induced by the use of DoH, due to the fact that their DoH stack is built upon their TLS stack. Therefore the memory consumed by HTTP2's protocol layers are added to the memory consumed by the TLS layer. For Dnsdist however, we observe that DoH has a lower memory consumption than DoT. While this seems counter-intuitive, it is consistent with the memory consumption per connection and protocol announced in its documentation (that advertises a higher per-connection memory consumption for DoT than for DoH). As the number of simultaneous connections never reaches 400 000 in the following experiments, we conclude that memory consumption won't be the limiting factor in our setup.

Table 1 – Parameters used when measuring the cost of handling queries

Software	Protocol	Number of connections
Knot-resolver	TCP	20 / 40 / 100 / 500 / 1000 / 2000 / 4000 / 8000
	DoT	
	DoH	
Dnsdist	TCP	20 / 40 / 100 / 500 / 1000 / 2000 / 4000 / 8000
	DoT	
	DoH	
Adguard-dnsproxy	TCP	20 / 40 / 100 / 500 / 1000 / 2000 / 4000 / 8000
	DoT	
	DoH	

5.4 Cost of handling queries

While the use of these new connected protocols seems to cause no issue regarding memory consumption, it can induce a CPU overhead due to the additional steps required when handling messages. These can be broken down into two parts: first, the connection establishment, and then, the handling of individual messages (see Section 2). The experiment described here aimed to estimate the cost of handling individual queries.

In order to measure the additional cost per request, we must take into consideration the number of simultaneously opened connections over which requests are sent. To do this, we sent a fixed amount of traffic over a set number of already opened connections. We repeat that experiment multiple times for each protocol / server combination, with a variable number of connections for each experiment. The various experimental parameters chosen are presented in Table 1. The total fixed amount of queries sent, as well as the minimum number of connections was chosen to ensure that the CPU utilization of the server process and the frequency of the core it ran on, were as high as possible.

Each point on Fig. 4 represents the average number of queries per second that were successfully answered by the tested server, with bars presenting the minimum and maximum value reached, for a specific protocol and a specific number of connections. We also represented the max traffic handled with UDP for comparison purposes. For every connected protocol there is a performance drop when compared to UDP. This is partly because, at the kernel level, handling a message sent over a TCP connection involves extra steps when compared to UDP, such as checking the consistency of acknowledgment numbers or calculating the congestion window size. Furthermore, we observe that, for Dnsdist, there is a noticeable drop in performance between TCP and DoT, while for Knot-resolver and Adguard-dnsproxy, their performances are very similar. We explain these differences by the fact that Knot-resolver or Adguard-dnsproxy are less efficient at handling DNS queries, meaning that the cost added by symmetric encryption is absorbed by the cost of handling DNS queries. As Dnsdist is more efficient, the per-message cost added by symmetric encryption is more noticeable. When comparing DoH to other protocols, we notice, for all three servers, a huge drop in performance (by a factor of two), that we explain by the added cost of handling the HTTP/2 protocol layers, which includes sending and receiving the control messages, allocating the data structures for new streams, and using the HPACK algorithm to decode message headers. For every protocol, we observe that performances tend to drop when the number of connections increases (up to a 40% decrease when considering Dnsdist over DoH), except for Dnsdist over TCP.

5.5 Overhead of establishing connections

In the following experiment, we measure the cost of opening and closing connections for each proto-

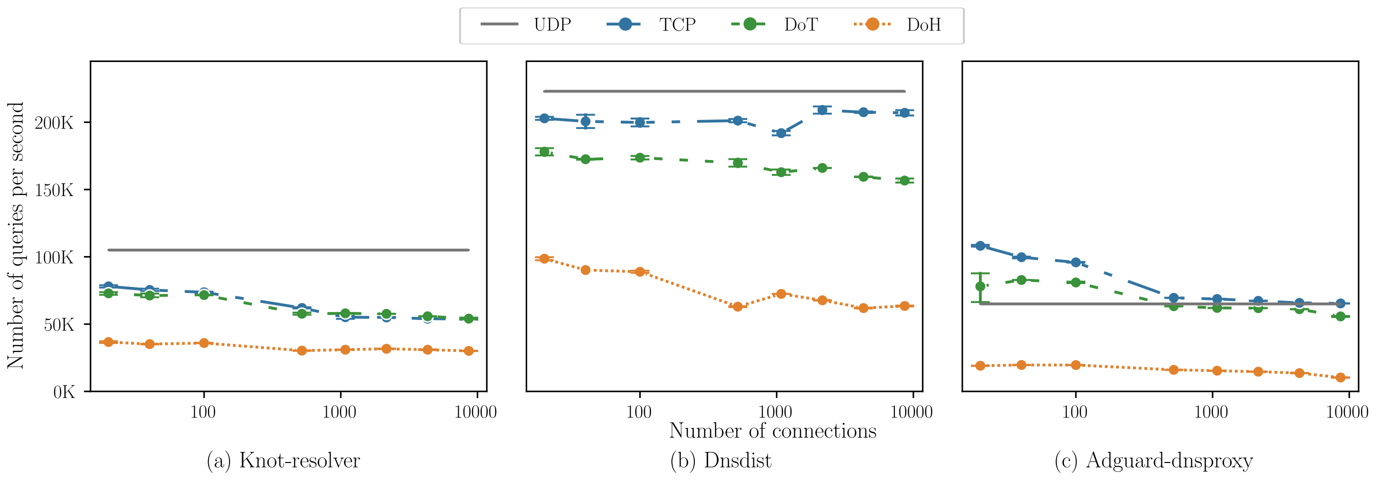


Fig. 4 – Queries per second handled by the servers when connections last all experiment

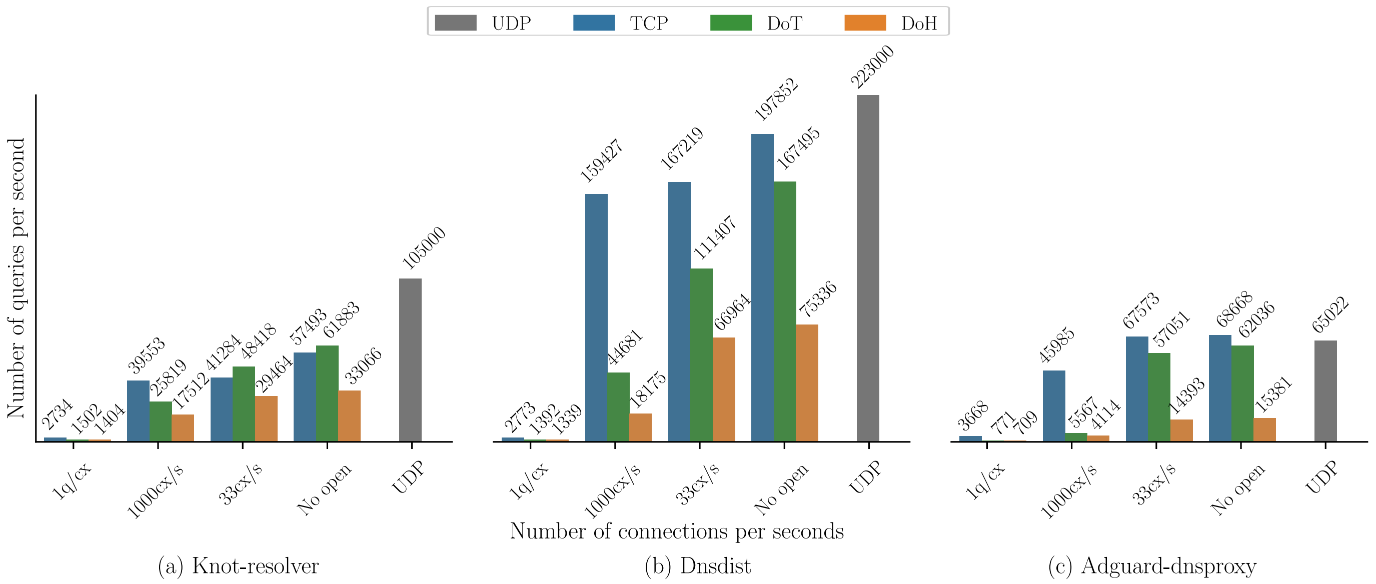


Fig. 5 – Queries per second (qps) handled per server and protocol according to number of connection establishments per second

Table 2 – Parameters used when measuring the cost of establishing connections

Software	Protocol	Number of connections	Connection duration
Knot-resolver	TCP	1000	1 query per connection / 1s / 30s
	DoT		
	DoH		
Dnsdist	TCP		1 query per connection / 1s / 30s
	DoT		
	DoH		
Adguard-dnsproxy	TCP		1 query per connection / 1s / 30s
	DoT		
	DoH		

col. We use the same experimental parameters as the ones used in the previous experiment (Fig. 4) to plot the points corresponding to 1000 connections, but this time taking into account connection establishment and tear down. Thus, by comparing this experiment and the previous one, we can infer the overhead of connection establishment (TCP connection establishment, TLS key exchange). We run two sets of experiments, one in which our clients close and re-establish a connection every 30s, the other in which clients close and re-establish connections every second (to match the long and short-lived connections we observe in Fig. 4). The parameters used for this experiment are presented in Table 2. In order to avoid the case in which the server has to handle a batch of connection establishment every 1 or 30 seconds, then only queries, and then another batch of connection establishment, we ensure that clients are started sequentially, with a set delay in between them. This, coupled with the fact that clients are configured to close and re-open connections on a strict timer, means that, in each experiment, the server has to handle a fixed amount of connection establishments every second : in the first experiment, the server has to handle 33 connection establishments every second, while in the second experiment, it has to handle 1 000 connection establishments every second. In addition to being set by design, these rates were confirmed to have been respected through an analysis of the network trace generated by each experiment. We run an additional batch of experiments to reproduce the very short-lived connections also observed in Section 4 : during this experiment, the clients are configured to establish a connection, send a single query over it, wait for its answer, close the connection and immediately start over.

The results of the experiments are presented in Fig. 5, in the form of the number of queries per second handled by a server for each combination of protocol and connection duration. We also plotted the maximum queries per second we reached in the previous experiment (Fig. 4) as a baseline for comparison. For all three servers, we observe the same performance variations between protocols, but with different orders of magnitude: when few connection establishments occur (33 connection establishments per second), we observe a variable decrease in performance (of up to 20%) for TCP and DoT, relative to the experiment in which no connection establishments occur. However, we do not observe this performance loss for DoH, as the CPU cost of query handling is still the bottleneck. When the frequency of connection establishments increases (connections last for 1 second), the performances of TCP decreases less than that of both encrypted protocols, as they both see a decrease in performances due to the added cost of the TLS key exchange. When connections are used for one query only, the cost of establishing TCP connections induces a collapse of performances for all protocols.

5.6 CPU usage of resolver process

We also measured the time spent by the processes of Knot-resolver, Dnsdist and Adguard-dnsproxy in kernel or user mode. Fig. 6 presents, for these severs, the percentage of their total scheduled time they spent in user or kernel mode, depending on the amount of connections per second they had to handle. We base our analysis on the fact that, for these software, TCP and UDP is handled in the kernel, while TLS and HTTP/2 are handled in userland.

UDP is a very simple protocol implemented entirely in the kernel. As there is no connection or encryption, all of the user time is spent on the DNS resolution service, and not on the handling of higher protocol layers, such as TLS or HTTP/2. Therefore, comparing how the servers handle it shows us how they handle the base DNS protocol. We can infer, from the fact that Knot-resolver and adguard spend a higher fraction of their time in user mode when compared to Dnsdist, for a lower amount of queries handled, that they spend more time handling base DNS messages than Dnsdist, which is consistent with the fact that they are less efficient than Dnsdist at handling base DNS queries.

Studying the behaviour of these servers on more complex protocols, we observe that handling DoT or DoH results in more time spent in userland. This phenomenon is less marked for Knot-resolver when it is handling few DoH connection establishments (No open or 33cx/s on Fig. 6(a)), as it makes more per-query syscalls then Dnsdist or adguard in order to handle the HTTP/2 protocol.

For all three servers however, the more TLS connection establishment occurs (which is the case when handling DoT or DoH at 1000cx/s or 1q/cx), the more time is spent in userland, as this is where the (costly) TLS handshake is handled.

5.7 Resolver power consumption

We measured the energy consumption of the server, first at idle, then during the experiments (when one core is fully loaded), and the delta between the two gives us the lower bound of the energy consumed by the various protocols. This delta is then used to compute the cost, in KWh, of one request and we use this cost to obtain the energy consumed, in kWh, of handling 10k QPS for a day (results in Table 3). As this cost does not include the idle consumption, if the cost per query is really low (as is the case with DNS over UDP), our projection results in low estimated energy costs. In the opposite case, when the cost per query is very high, this can result in a very high estimated energy consumption, as handling 10K QPS in this case, would necessitate running several cores at 100%, or even running additional machines. If connection use is fair (30s) the use of secured protocols is worth considering, even though it increases energy consumption by a factor of two for DoT and four for DoH, but when we consider a higher load, in which connection openings occur more frequently the use of such protocols become more costly (up to 15 times for DoH at 1s), or unsustainable in the case

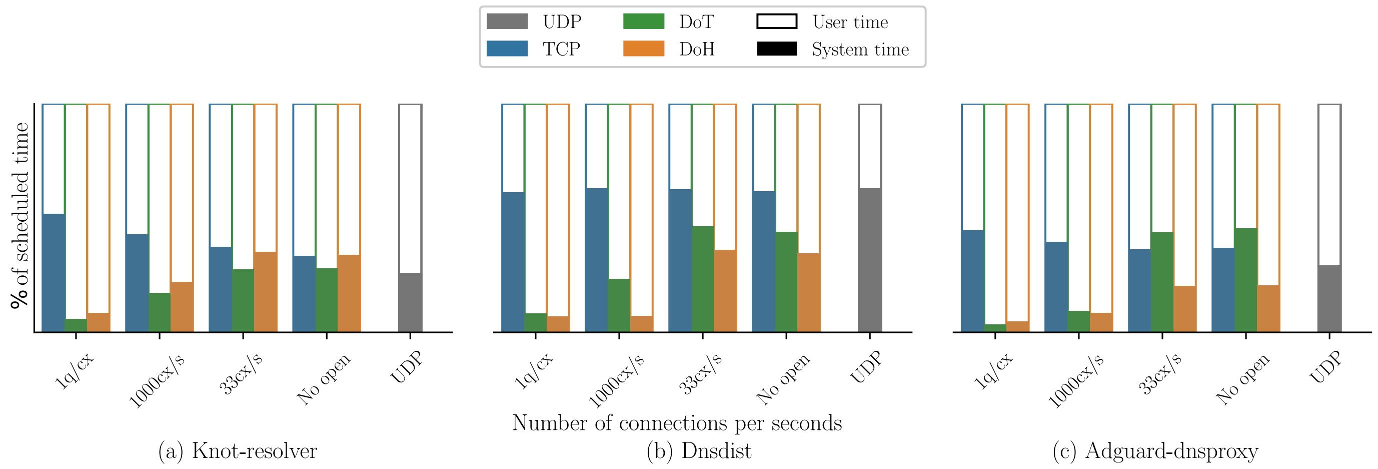


Fig. 6 – Percentage of its scheduled time the resolver process spent in each mode

Table 3 – Estimation, **in kWh** of the energy consumed by running 10k QPS for one day using the measured profiles

	Knot-resolver			Dnsdist			Adguard		
Protocol	1 q/cx	1000cx/s	33cx/s	1 q/cx	1000cx/s	33cx/s	1 q/cx	1000cx/s	33cx/s
UDP	0.12			0.05			0.19		
TCP	3.1	0.34	0.33	3.08	0.08	0.08	1.87	0.29	0.2
DoT	8.97	0.52	0.28	9.46	0.3	0.12	17.48	2.42	0.24
DoH	9.67	0.77	0.46	9.85	0.73	0.19	19	3.27	0.97

of aggressive connection opening (0.05 kWh versus 9.85 kWh).

Despite its worse overall power consumption, Adguard-dnsproxy distinguishes itself on TCP connection establishment, as it shows a 30% performance gain over the other two.

6. CONCLUSION

DNS is still at the core of today's Internet. Originally designed for performance, using an un-encrypted connection-less protocol, growing concerns about security have led to the standardization of secured protocols. In this article, we studied the resolver-side cost of transitioning to such protocols.

We benchmarked public resolvers using various DoH clients to gather profiles. We found that all entities tried to maintain connections opened but that, when faced with a high load, browsers became unstable and we observed a very high number of small connections. Then, we measured the additional cost of each step of the connected protocols (DNS over TCP, DoT, DoH). We observed that transitioning from the legacy protocol to a secured one lead to, at minimum, a division of the performances by two, due to TCP connection establishment, TLS key exchange and message encryption, and that the performance loss was even more noticeable when considering the switch to DoH because of the protocol layers added by HTTP/2 . When considering the fact that the use of DoH seems to be pushed by the industry, one can wonder if the complexity added by the HTTP/2 protocol is compatible with DNS's initial goals of efficiency and scalability. Furthermore, in the case in which connections are short, performances take an even bigger hit, which can lead to a large increase in the number of resolvers as well as their energy consumption, leading in turn to a higher operating and environmental cost. As it is, we believe that switching 100% of the client-to-resolver DNS traffic to DoH is not sustainable. To realise this transition, it will be necessary to ensure that clients can keep their connections alive as much as possible, and to use less costly protocols than HTTP/2, that still retain its ability to go through firewalls. Therefore, it could be interesting in the future to measure the performances of QUIC-based protocols, such as DNS over QUIC or DNS over HTTP/3. or to experiment with mechanisms to reduce HTTP/2's complexity in the context of DNS over HTTPS.

APPENDIX

Table 4 – Table presenting the number of connections, 1st quartile (Q1), median (Q2) and 3rd quartile (Q3) of the distribution of connections length and messages per connection measured when recording the behaviour of popular DNS clients' and servers' implementations when faced with varying load.

	Cloudflare							Google							Quad9						
	# conns	Messages per conn			Conn duration			# conns	Messages per conn			Conn duration			# conns	Messages per conn			Conn duration		
		Q1	Q2	Q3	Q1	Q2	Q3		Q1	Q2	Q3	Q1	Q2	Q3		Q1	Q2	Q3	Q1	Q2	Q3
50ms	485	1	4	8	0.38	0.72	2.05	1063	1	2	7	0.08	0.20	0.76	1005	0	1	3	0.16	0.36	1.22
1000ms	7	0	0	87	0.07	0.10	75.37	9	0	0	42	0.18	0.19	30.07	11	0	0	143	0.11	0.12	132.27
60000ms	13	0	0	0	1	2.0	2.2	7	0	0	6.5	0.17	0.21	15.02	30	1	1	2.75	30.1	30.25	56.72

(a) Client : Firefox

	Cloudflare							Google							Quad9						
	# conns	Messages per conn			Conn duration			# conns	Messages per conn			Conn duration			# conns	Messages per conn			Conn duration		
		Q1	Q2	Q3	Q1	Q2	Q3		Q1	Q2	Q3	Q1	Q2	Q3		Q1	Q2	Q3	Q1	Q2	Q3
50ms	847	1	3	10	0.16	0.41	1.14	1540	1	3	6	0.08	0.20	0.76	693	1	2	6	0.12	0.23	0.68
1000ms	3	2.5	4	905	427.66	455.20	1135.30	3	2.5	4	905	267	295	1047	9	0	1	3	0.40	30.12	30.26
60000ms	3	2.5	4	18.5	427.80	455.25	1135.31	3	2.5	4	18.5	267.69	292.23	1055.29	33	1	1	1	30.1	30.17	30.42

(b) Client : Chromium

	Cloudflare							Google							Quad9						
	# conns	Messages per conn			Conn duration			# conns	Messages per conn			Conn duration			# conns	Messages per conn			Conn duration		
		Q1	Q2	Q3	Q1	Q2	Q3		Q1	Q2	Q3	Q1	Q2	Q3		Q1	Q2	Q3	Q1	Q2	Q3
50ms	3	9953	9998	9999	600.3	608.5	612.1	1		29960			1813.5		1		29959			1813.6	
1000ms	1		1787			1816		1		1783			1815.7		1		1792			1815.8	
60000ms	30		1		5.03	5.09	5.19	30		1		5.04	5.05	5.16	30	1	1	1	5.07	5.09	5.18

(c) Client : DNSCrypt-proxy

ACRONYMS

CPU Central Processing Unit
DNS Domain Name System
DoH DNS over HTTPS
DoQ DNS over QUIC
DoT DNS over TLS
DoUDP DNS over UDP
HTTP/2 HyperText Transfer Protocol version 2
IETF Internet Engineering Task Force
IP Internet Protocol
ISP Internet Service Provider
NIC Network Interface Controller
NSS Network Security Services
QPS Queries Per Second
TCP Transmission Control Protocol
TLS Transport Layer Security
TOR The Onion Router
UDP User Datagram Protocol
URL Uniform Resource Locator

GLOSSARY

FIN TCP flag
GET HTTP request method
GOAWAY HTTP/2 control message
HEAD HTTP request method
HPACK HTTP/2 header compression algorithm
POST HTTP request method
QUIC Transport-layer network protocol
RST TCP flag

ACKNOWLEDGEMENTS

This work is supported by the ENE5AI project (DOS0185314 / DOS0185315 / DOS0185931 / DOS0185932), the FogSLA-Antillas project (DOS0168403/00-DOS0168405/00) and the ANRT (CIFRE n°2022/0178)

REFERENCES

[1] Zi Hu et al. *RFC 7858: Specification for DNS over Transport Layer Security (TLS)*. 2016.

[2] Paul E. Hoffman et al. *RFC 8484: DNS Queries over HTTPS (DoH)*. 2018.

[3] Eric Rescorla. *RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3*. 2018.

[4] Mike Belshe et al. *RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2)*. 2015.

[5] S. Bortzmeyer. *RFC 7626: DNS Privacy Considerations*. Tech. rep. 7626. 2015.

[6] S. Siby et al. "Encrypted DNS ==> Privacy? A Traffic Analysis Perspective". In: *The Network and Distributed System Security Symposium (NDSS '20)*.

[7] Bushart et al. "Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS". In: *10th USENIX Workshop on Free and Open Communications on the Internet, FOCI 2020*.

[8] I. M. M. Dissanayake. "DNS Cache Poisoning: A Review on its Technique and Countermeasures". In: *National Information Technology Conference (NITC '18)*.

[9] G. Lowe et al. "The great DNS wall of China". In: *MS, New York University* (2007).

[10] D. Vekshin et al. "DoH Insight: Detecting DNS over HTTPS by Machine Learning". In: *15th International Conference on Availability, Reliability and Security (ARES '20)*. Virtual Event, Ireland, 2020.

[11] Austin Hounsel et al. "Comparing the Effects of DNS, DoT, and DoH on Web Performance". In: *Web Conference 2020 (WWW '20)*. Taipei, Taiwan, 2020.

[12] Austin Hounsel et al. "Analyzing the Costs (and Benefits) of DNS, DoT, and DoH for the Modern Web". In: *Proceedings of the Applied Networking Research Workshop (ANRW '19)*. Montreal, Quebec, Canada, 2019.

[13] Timm Böttger et al. "An Empirical Study of the Cost of DNS-over-HTTPS". In: *Internet Measurement Conference (IMC '19)*. Amsterdam, Netherlands, 2019.

[14] Rishabh Chhabra et al. "Measuring DNS-over-HTTPS Performance around the World". In: *Proceedings of the 21st ACM Internet Measurement Conference (IMC '21)*. 2021.

[15] Mike Kosek et al. "DNS Privacy with Speed? Evaluating DNS over QUIC and Its Impact on Web Performance". In: *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22)*. Nice, France, 2022.

[16] S. Garcia et al. "Large scale measurement on the adoption of encrypted DNS". In: *arXiv preprint arXiv:2107.04436* (2021).

[17] Gerald Combs et al. *Tshark v3.6*.

[18] Google. *chromium*. Version 101.0.4. URL: `chromium.googlesource.com/chromium/src.git`.

[19] statcounter. *Browser Market Share Worldwide*. 2022. URL: `gs.statcounter.com/browser-market-share`.

[20] Mozilla foundation. *firefox*. Version 91.5.0esr. URL: `hg.mozilla.org/mozilla-central/`.

[21] DNS OARC. *DNS sample queries file*. 2012. URL: `github.com/DNS-OARC/sample-query-data`.

[22] Etienne Le Louët. *doh-client-analysis-scripts*. URL: `github.com/etienne-lelouet/doh-client-analysis-scripts`.

[23] D. Balouek et al. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In: *Cloud Computing and Services Science*. 2013.

[24] cz.nic. *knot-resolver*. Version 5.5. 2022. URL: `www.knot-resolver.cz/`.

[25] PowerDNS. *Dnsdist*. Version 1.7.3. 2022. URL: `dnsdist.org/`.

[26] adguard. *dnsproxy*. Version 0.54. 2023. URL: `github.com/AdguardTeam/dnsproxy`.

[27] ns1labs. *flamethrower*. URL: `github.com/DNS-OARC/flamethrower`.

[28] Etienne Le Louët ns1labs. *flamethrower*. URL: `github.com/etienne-lelouet/flamethrower`.

AUTHORS

Etienne Le Louët graduated from Sorbonne Universite in 2021 with a master's degree in computer science from Sorbonne Universite (Paris, France). He is currently working as a PhD student in the research and development department of Gandi (Paris, France), co-supervised by both Antoine Blin at Gandi and Julien Sopena in the DELYS team at LIP6 (Sorbonne Universite, Paris, France).

Antoine Blin obtained his PhD from Sorbonne Universite (Paris, France) in 2017. He is currently head of research and development at Gandi (Paris, France). He has authored multiple papers in the domain of operating systems kernels.

Julien Sopena received a PhD in computer science in 2008, at Sorbonne Universite. He is currently associate professor in computer science at Sorbonne Universite, and carries out his research within the LIP6 lab, in the DELYS team more specifically. His research interests include large scale distributed systems such as computing grids, cloud computing and multicore architectures.

Kamel Haddadou received an engineering degree in computer science from INI in 2000, an MS degree in data processing methods for industrial systems from the University of Versailles in 2002, and a Ph.D. in computer networks from Pierre and Marie Curie University in 2007. In 2001, he was a research assistant at the Advanced Technology Development Centre (CDTA), Algiers, Algeria. He is currently Chief Revenue Officer at Gandi SAS, France. Since 2003, he has been involved in several projects funded by the European Commission and the French government (RAVIR, ADANETS, Adminroxy, GITAN, OGRE, ADANETS, MMQoS, SAFARI, and ARCADE). His research interests are focused primarily on cloud computing and resource management in wired and wireless networks. He is equally interested in designing new protocols and systems with theoretical concepts, and in providing practical implementations that are deployable in real environments. He is an author of more than 30 papers published in leading conference proceedings and journals. He has served as the TPC member for many international conferences, including IEEE ICC, GLOBECOM, and reviewer on a regular basis for major international journals and conferences in networking. He is a member of the IEEE.

Ahmed Amamou received an Engineering degree in computer science and an M.S. degree in network and computer science from the National School of Computer Science, Tunisia, in 2009 and 2011, and a Ph.D. in network and computer science from the Sorbonne Universite, Paris, France, in 2013. He is currently Chief Technical Officer at GANDI SAS. His research interests are cloud computing and virtualization technologies.