DNS Over HTTPS Traffic Analysis and Detection

Carlos López Romera
Master's Degree in Information and Communication Technology Security
Systems hacking area

Thesis director: Carlos Hernández Gañán
Area supervisor: Víctor García Font

2nd June, 2020

THESIS DATASHEET

Project title:	DNS Over HTTPS Traffic Analysis and Detection
Author:	Carlos López Romera
Thesis director:	Carlos Hernández Gañán
Area supervisor:	Víctor García Font
Date (mm/yyyy):	06/2020
Degree:	Master's Degree in Information and Communication Technology Security
Knowledge area	Hacking
Language:	English
Keywords	DNS over HTTPS, traffic analysis, machine learning

Abstract

The Domain Name Service (DNS) is a prevalent protocol used in computer communications, used to translate domain names to addresses that can be routed to via de Internet Protocol (IP). One of the main characteristics of DNS is the use of plaintext requests and responses, leaking information even in traditional secure communications; a client might resolve a server's IP address using plaintext messages, and then cryptographically protect its exchange with the server itself.
DNS over HTTPS (DoH) is a protocol specification introduced in the IETF RFC 8484 (2018), which provides a mapping of regular DNS requests and responses over TLS-encapsulated HTTP messages. TLS (Transport Layer Protocol) and HTTP (HyperText Transfer Protocol), known in conjunction as HTTPS, are the two most common methods of communication with web servers, each providing security and structure respectively. DoH, then, provides not only the cryptographic benefits of TLS, but also the masquerading of DoH communications as regular web traffic.
Although recent work has aimed to identify the content of DoH communications by using different fingerprinting techniques, distinguishing regular TLS-encapsulated HTTP traffic from DoH remains an unsolved challenge.
In this thesis, passive analysis of DoH traffic is presented, as well as a method and implementation for its detection.

INDEX

1. Introduction

1.1. Context

The Domain Name Service (DNS) is a predominant protocol in computer communications. In the current landscape, mostly every connection between two endpoints is preceded by a domain resolution query and response to translate a known, text-based domain name, to a numerical Internet Protocol (IP) address.

In the recent years, the privacy of this protocol has become a topic of interest in the field of information security research, as it can potentially leak data about an user's activity on the World Wide Web, among other Internet services [1], [2].

The reason for these issues is that, traditionally, all exchanges using the DNS protocol are done without any form of encryption or authentication. To further illustrate the workings of DNS, a usual sequence of events for a domain name resolution is explained below. In a DNS resolution procedure there are three types of endpoints: a client, a recursive resolver, and several iterative resolvers. Iterative resolvers consist of the root, TLD and authoritative resolvers.

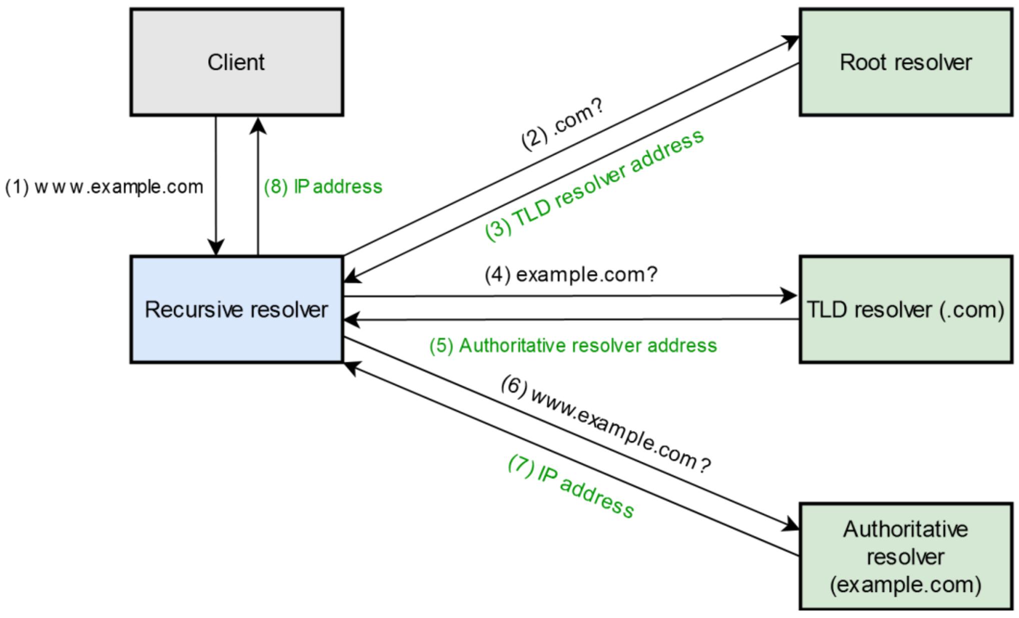


Figure 1

To follow the usual sequence of events, let us suppose that a client wants to access the domain *www.example.com* through a web browser. To establish the connection over which the web data will be transmitted, the client needs to translate said domain to an IP address. The process to perform this translation is illustrated in Figure 1[1].

[1] QNAME minimization [81] is used in the figure for simplicity.

First, the client sends its request to the recursive resolver, which in turn will perform requests against several iterative resolvers. It must send a query to the root resolver for the *.com* TLD resolver. The same process is repeated against the TLD resolver, asking for the *example.com* authoritative resolver. Finally the recursive resolver can ask for the for the *www* subdomain within the *example.com* domain and return the IP address to the client.

The description above matches the workings of the DNS protocol for over the last 30 years [3], [4]. As previously mentioned, the exchange between the client and the recursive resolver is performed in plaintext (UDP port 53), which can serve malicious actors for several purposes.

1.2. DNS threats

The lack of protection mechanisms inherent to the DNS protocol incur in several threats, from the traffic analysis perspective, related to authenticity, integrity and privacy. For these threats to be valid, we assume an attacker situated in the path between the client and the recursive resolver, either locally (for example, an adversary connected to the same Wi-Fi network) or on the path through the public network an (an ISP). Figure 2 displays the described threat model.

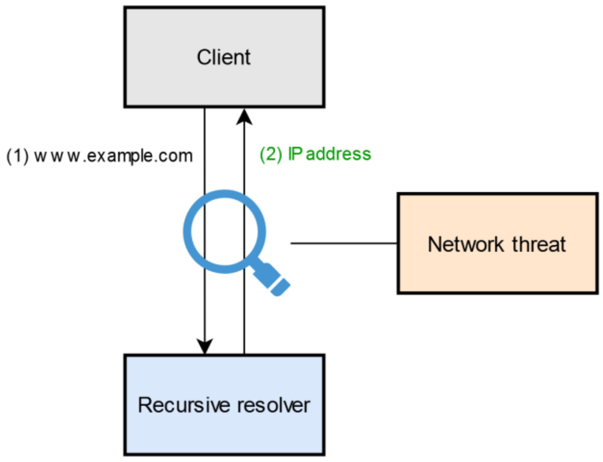


Figure 2

Firstly, the data sent over the network has no integrity mechanisms, meaning that it might be altered on the fly by a third party, and the client would have no way of noticing. Additionally, due to the lack of authenticity, an attacker can masquerade as the original DNS resolver, by means of what is known as a man-in-the-middle attack (MITM) [5]. If an adversary is able to deploy such an attack, it can serve selected IP addresses to the client, redirecting its traffic to malicious servers.

Additionally, attackers can exploit the lack of privacy in the protocol. If an adversary controls the DNS traffic's pathway, it can selectively choose to block said traffic to perform certain types of censorship. Another more passive technique would be to capture and analyze DNS traffic to collect data about the user's activity. Given the pervasiveness of DNS traffic, this could result in a total loss of privacy for a World Wide Web user.

As a remainder, note that there are other threats related to DNS, such as the privacy of the data collected by a resolver about its users, that are not taken into account from a purely traffic analysis-based perspective.

1.3. DNS security mechanisms

To solve DNS's security problems, a standard for several security extensions, was proposed as far back as 1999 [6]. DNSSEC, the name given to this proposal, addresses data integrity and endpoint authenticity, but does not aim to solve problems with privacy. These additions to the protocol allow for responses indicating the IP address for a certain domain (or even the non-existence of it [7]) to have cryptographic integrity. Therefore, it aims to solve issues with resolver impersonation and traffic manipulation, either with censorship or traffic redirection purposes. However, as stated, DNSSEC leaves privacy concerns untouched, and moreover, it has seen low adoption rates [8].

HTTP (HyperText Transfer Protocol) is a protocol that has faced similar challenges in the past. It is the predominant protocol for web communications since the 1990s [9], but as DNS, lacks the adequate security mechanisms. Thus, the kind of concerns described above used to be present for this protocol too, before the introduction and widespread adoption of SSL (Secure Sockets Layer) [10] and then TLS (Transport Layer Security) [11], its successor. The mapping of HTTP over one of these two protocols is referred to as HTTPS (HTTP Secure) [12]. TLS, and previously SSL, act as an intermediate layer, providing encryption, integrity and authenticity.

It follows, then, the use of TLS as an intermediate security layer for the Domain Name System protocol: DNS over HTTPS (DoT) [13]. Much like in HTTPS, DNS queries are encapsulated using the TLS protocol, which provides a straightforward specification to introduce the encryption and authentication properties of TLS to DNS with a minimal amount of modification to the original protocol; other than the switch from UDP port 53 to TCP port 853, and the TLS encapsulation, DNS remains mostly the same.

However, adoption has not been unified, as DNS over HTTPS (DoH) has also been proposed as an alternative for DNS security [14] in 2018. Although the explicit cryptographic properties remain the same as in DoT, DoH is sent to TCP port 443, the default port also for HTTPS, which helps the masquerading of DoH as regular HTTPS traffic. The main goal, however, is to provide a simpler interface for web applications (i.e. browsers).

Briefly, DoT and DoH propose the network stacks illustrated in Figure 3 for transmitting DNS queries.

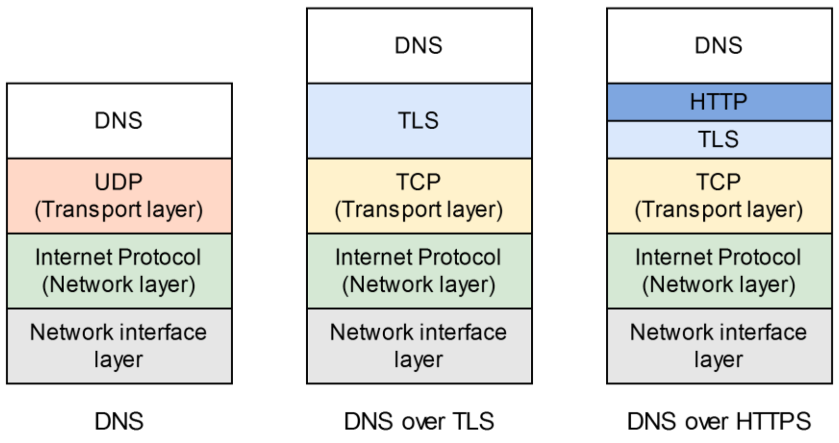
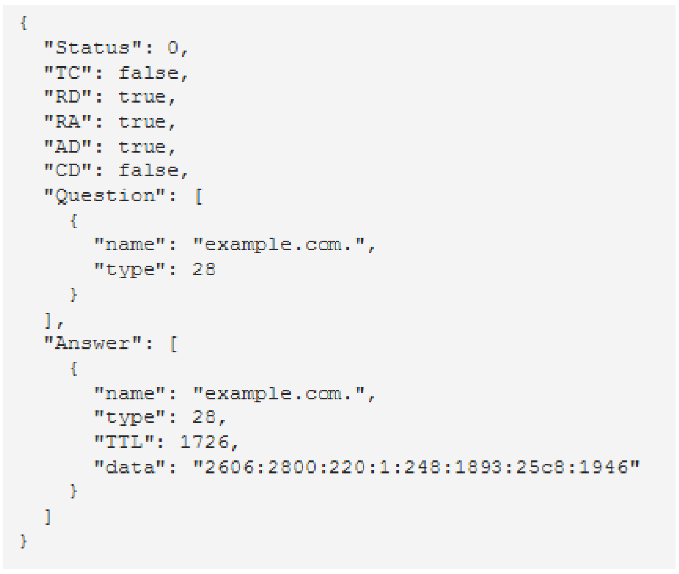


Figure 3

Unlike DoT, DoH introduces several alternatives when it comes to sending DNS data. The first way, known as wire format, consists on simply encapsulating a regular DNS query in an HTTP message, either a GET or POST request, meaning that the resolver must be able to parse and understand the HTTP protocol. A GET request will usually take the following form:

```
https://dns-resolver.com/?dns=<base64-encoded-query>
```

While in a POST request, the query will be sent in the body of the HTTP message. Additionally, a second format which uses JSON structures was introduced by Google and later standardized [15] adopted by other providers [16]. This second way of DoH communications, although not included in the official IETF RFC, allows for more simplicity, avoiding the need for a DNS format parser and allowing requests and responses to be text-based. Several major browsers introduce the option of using the JSON format [17], [16], despite the fact that it is not enabled by default. Figure 4 shows a sample response for a type AAAA query for the *example.com* domain [16].



```
{
  "Status": 0,
  "TC": false,
  "RD": true,
  "RA": true,
  "AD": true,
  "CD": false,
  "Question": [
    {
      "name": "example.com.",
      "type": 28
    }
  ],
  "Answer": [
    {
      "name": "example.com.",
      "type": 28,
      "TTL": 1726,
      "data": "2606:2800:220:1:248:1893:25c8:1946"
    }
  ]
}
```

Figure 4

DNS over HTTPS introduces several other differences from DNS over TLS, such as the use of HTTP/2 mechanisms (server push, header compression, stream parallelism, etc.), the use of TLS 1.2 or higher as a consequence, and the integration with the overall HTTP ecosystem, such as caches, proxying and authentication. The key difference, however, is that, although both alternatives have seen adoption in different areas, it is DoH that has been selected as the security mechanism for DNS in web browsers [18], [19], [20], [21], [22], mainly due to the easier interoperability browsers have with HTTP APIs. This fact alone, along with the integration already in place, puts this protocol in the spotlight for network security research.

Although research on both DoT and DoH fingerprinting is available [23], [24], [25], the problem of detection remains poorly researched. While DoT traffic can be identified due to the use of a specific TCP port, DoH shows no apparent distinction to regular HTTPS traffic. Therefore, the goal of this work is centered around DoH identification.

1.3. Motivation and objectives

From an adversary's perspective, it is of interest to detect DoH traffic. The intents might include user monitoring, analytics or censorship. In the simplest scenario, an attacker might just be interested in detecting DoH traffic to block it, preventing a client accessing a certain domain, or forcing the client to fall back to plaintext DNS. In more elaborate attacks, the adversary might want to fingerprint encrypted DNS requests to monitor the user's activity, as exposed in the research mentioned previously. The kind of attacks and techniques described above, such as censorship or advanced traffic analysis, are not only used by individuals, but also state-sponsored agencies and ISPs [26], [27], [28], [29].

Therefore, the goal of this work is to demonstrate DNS over HTTPS detection, independently of per-case variables such as destination IP addresses, with the intent of offering a general overview of the protocol's weaknesses, rather than proposing a solution specific to one certain situation. The proposal is to do so through the obtainment of a traffic capture dataset, extraction of macroscopic features on a per-connection basis, and using machine learning classifier algorithms to perform detection.

By exposing the characteristics of a DNS over HTTPS connection from the network analysis perspective, the final objective is to provide a reference for privacy enhancement in DNS security.

2. State of the art

This section is structured as follows: first, a basic overview of the relationship between traffic analysis and machine learning is given. Then, general research on DNS over TLS and DNS over HTTPS is reviewed in order to establish a relationship to our analysis. Finally, existing solutions to the problem of DoH detection are reviewed.

2.1. Traffic analysis and machine learning

Traffic classification is a common problem in the field of information technology and computer science. The application of machine learning to this task is no longer a new field of research, with studies on this subject dating back as far as 2005 [30].

In the field of machine learning, there are two main methods to train an algorithm: supervised and unsupervised. In both types, an algorithm's task is to classify a set of input samples into a set of output classes based on the samples' features. In a supervised model, these classes are already appended to the input samples, meaning that the algorithm will need to adapt to match each sample with its output class. On the other hand, in an unsupervised model there are no predefined class, and thus the algorithm will classify the samples into new classes (also known as clusters) autonomously. Additionally, other hybrid methods that combine supervised and unsupervised training exist. Although unsupervised traffic classification is a field with existing research [31], [32], [33], we focus on supervised learning, as, for our current problem in hand, the output classes are already known: HTTP and DoH.

In an early study [34], several classic classifier algorithms (Naïve Bayes Tree, C4.5 Decision Tree, Random Forest and KNN) are compared in a task of traffic classification for very different types of applications (DNS, HTTP, POP3, FTP, several P2P applications, etc.), with good results for Random Forest C4.5 and KNN. This research serves as an early reference, but the types of traffic differ widely in purpose and characteristics.

Another early reference can be found in [30], where a survey on different machine learning techniques for traffic classification is carried out. This paper does not focus on a specific methodology to classify data, but rather offers an overview on the different methods to accomplish it (supervised/unsupervised, packet-level/connection-level/multi-flow level samples, common metrics and other techniques). An important element mentioned in this survey is the use of correlation-based feature selection techniques; an example of this type of selection is the Pearson's correlation coefficient method, which is explained in section 3.2. The authors also provide a general high level schema for supervised traffic classification experiments, shown in Figure 5.

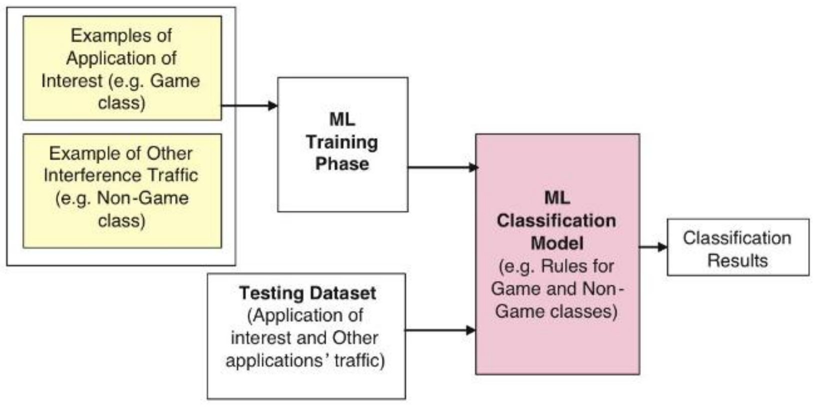

Examples of Application of Interest (e.g. Game class)
Example of Other Interference Traffic (e.g. Non-Game class)
ML Training Phase
ML Classification Model (e.g. Rules for Game and Non-Game classes)
Classification Results
Testing Dataset (Application of interest and Other applications' traffic)

Figure 5

A final conclusion to draw from this survey is that up until that point in time, most classification efforts were centered around the identification of traffic coming from very different applications (web, Telnet, FTP, mail services, P2P, etc.). Additionally, the lack of ensemble algorithms (machine learning techniques that encompass a number of other known algorithms to yield a more accurate prediction) such as AdaBoost and Random Forest is a highlight.

In [35], the authors set out to detect different HTTPS-encapsulated services through the use of traffic fingerprinting, where traffic for these different services is generated, and then the data extracted is use to detect said services in the wild. This paper does not aim to detect the behavior of different protocols per se, but rather the differences in several web services such as Google Maps, Google Drive or several Dropbox applications. This problem, by design, is more open, and the classification tags (extracted from the TLS SNI extension, explained in section 2.3) can be arbitrarily big, depending on the number of services one aims to detect. They use the Naïve Bayes Classifier, Random Tree, C4.5 Decision Tree and Random Forest.

In [36], the authors face a very similar problem to the one in hand: binary classification of encrypted HTTP streams. In their case, however, their goal is to characterize HTTP/1 and HTTP/2 flows. Each sample (a TCP connection) is labeled according to the TLS APLN extension (explained below in section 3.2). As in previous studies, they employ Random Forest, C4.5 and Naïve Bayes Tree, as well as a Bayesian network, being Random Forest the one with the best results across all metrics. This study serves as a methodology reference for dataset treatment (separating each network connection as a sample), as well as algorithm evaluation.

On top of the mentioned research, it is necessary to mention other, more advanced approaches. In [37] the authors employ correlation between flows to identify webmail access from other types of traffic. In [38], a Fast Fourier Transform (FFT) is applied to a compiled sequence of packet sizes, ignoring flow-level statistical characteristics, and obtaining very high accuracy results with a Random Forest classifier. Finally, several approaches make use of deep learning networks to classify streams[39], [40], [41]. The deep-learning approach does not have a significant difference in methodology, as a set of features needs to be generated to feed into the network; however, deep learning techniques can allow the detection of more non-linear relationships, and even variable-length inputs in the case of recursive neural networks.

Overall, although research on traffic classification using machine learning has been present for over 15 years, few studies focus on a similar case as the one in hand, especially taking into account that DNS over HTTPS is not a completely separate protocol from HTTPS, but rather the addition of a different payload to HTTPS in order to transport different data. However, these studies provide a solid reference for metrics (such as confusion matrixes, accuracy or F-measure, explained in section 3.3) and the selection of proven algorithms such as K-Nearest Neighbors and different decision trees. Additionally, these studies can provide an idea towards the kind of features to extract from a network flow, but it is also assumed that this selection is problem-specific, i.e., relevant features for solving one problem might not apply directly to other types of classification.

2.2. DoT and DoH fingerprinting

Research on DNS over TLS and DNS over HTTPS is scarce, and even then, efforts mainly focus on traffic fingerprinting.

One of the main sources for research on DoH is found in [25]. They focus their efforts on fingerprinting the traffic patterns triggered when visiting a set of known websites. They characterize this pattern with a series of packet sizes, grouping them in n-sized groups, which they call n-grams; part of their research focuses on comparing this technique to other fingerprinting methods such as k-Fingerprinting [42] , CUMUL [43] and DF [44]. They perform tests on only-DoH, only-HTTPS and mixed traffic to recognize fingerprints, although they filter DoH traffic by destination IP address. Their results score over 91% precision with the Random Forests algorithm. As a countermeasure, they propose the elimination of packet size information by padding to constant-size messages.

In [45], authors take a similar approach to the previous study, in that they take a sequence of packet sizes to analyze and fingerprint, this time working with the security padding proposed for DoH in 2018 [46]. There are two additional differences in their approach: they examine only downstream packets (from resolver to client), and they take into account the gaps between messages, meaning that their sequences consist of interleaved message sizes and time gaps, not just the first. However, the goal is the same: to accurately identify a client visiting certain websites.

Through the use of the k-Nearest Neighbors algorithm, they achieve high accuracy results in several scenarios, up to a maximum of 95%. They conclude that the addition of padding does not completely solve the issue of DoH and DoT fingerprinting. As they express it, packets can be analyzed in the dimensions of counts (number packets), sizes and time - padding only addresses sizes. This is not sufficient because sequences can be unique enough for different websites despite of message padding. They propose constant-rate sending as a countermeasure.

The final study we look at is found in [23]. Here, the authors focus on DoT rather than on DoH, but it is still deemed relevant. Again, requests and responses are treated as a sequence of packets with three main characteristics: timestamp, size and direction. From here, several statistics are compiled for each website (mean, median, cumulative values, time intervals between packets), as well as other metrics such as time elapsed until the reception of N bytes and total transmission time. They compare a variety of classifiers (Simple Logistic, Naïve Bayes, Random Forest, AdaBoost and other less known algorithms as SMO and J48 Decision Tree). They find that both Random Forest and AdaBoost yield high accuracy scores (false negative rates below 7% and false positive rates below 5%). They also measure the loss of accuracy with the introduction of padding.

While DNS query deanonymization is an important issue, traffic identification is a preceding step which these studies take for granted. However, these studies display several characteristics particular to DoT and, more interestingly, to DoH, such as its burst-like size nature, and the importance of packet sizes and inter-arrival times.

2.3. DoH detection

For DNS over HTTPS detection, very little research is available. This is attributed to the fact that it is a young protocol, and that solutions do not seem trivial enough to implement with basic heuristics.

The main resource for DoH detection is found in [47]. In this study, several methods for DoH detection are proposed, and their feasibility is discussed below.

TLS inspection is the first method mentioned. This is, evidently, the most invasive technique for detection, as it requires complete traffic decryption and inspection to find out which network flows carry DNS data. A basic setup would consist in the installation of a middlebox that would decipher and inspect traffic to perform detection. Figure 6 illustrates this configuration in a very basic manner.

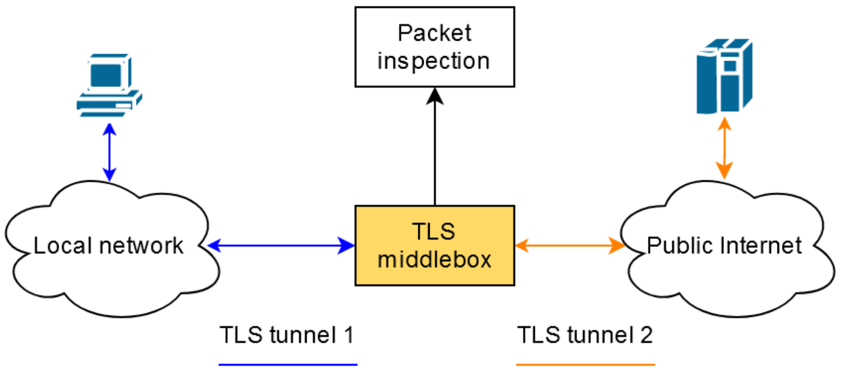


Figure 6

TLS decryption requires both technical and bureaucratic control over the network, since all traffic is going to be inspected. It most likely requires control over the involved clients inside the network too: as a TLS middlebox is installed, clients need to accept it as a valid endpoint, as its hostname will not match the one the clients are trying to access.

Either clients are forced to accept the middlebox's TLS certificate manually (usually through an option given by web browsers), or the new certificate needs to be added as trusted on the client machine. Even then, this will not solve issues when dealing with pinned certificates [48]. Using this mechanism, the client application is bundled with certain trusted authority certificates, and it expects to find these authorities in the verification chain. If pinned certificates are in use, the connection will fail, since, even if the middlebox's certificate is trusted, it will not match what the application is expecting.

Overall, this technique is only feasible under certain conditions such as complete control of the network, and clients being willing to have their traffic completely decrypted and inspected.

The next solution proposed is application logging. Mozilla Firefox, for example, offers the option to log every DNS request (whether encrypted or not) [49]. As with the previous alternative, it requires administrative control over the client performing the requests, which might not be possible; personal mobile devices and laptops are harder or impossible to monitor this way, and even then, every single client inside the network needs to run an application that can log DNS queries, and each one needs to be configured. Overall, this technique requires a great amount of administrative work to establish and maintain.

Finally, two open source tools are proposed. The first one is Zeek [50], an open source network analysis and security monitoring tool. The approach here is to analyze Zeek's logs to detect sites visited for which there have been no regular DNS requests. This presents two problems: the difficulty of identifying visited sites when encryption

(HTTPS) is in use, and the fact that the actual DoH connections are still not identified, even if its presence can be confirmed. To supply this, the use of JA3 fingerprints [51] is suggested. The way this kind of fingerprint works is through the computation of a hash string, employing parameters observed during the TLS handshake.

This approach is useful when dealing with malware campaigns; the main idea here is to block fingerprinted command and control servers so that malware cannot connect to them an receive instructions. These servers are usually kept static, as malware needs to find them autonomously, but DoH servers may not be kept the same by the client. An user might switch to a new DoH resolver, or might simply employ an HTTPS proxy.

The second tool suggested is RITA (Real Intelligence Threat Analytics) [52]. This tool makes use of Zeek logs to perform detection on beaconing activity amongst other suspicious traffic activities. While the author shows that RITA is able to mark DoH connections as beaconing activity, no quantitative metrics are given regarding accuracy or false positives.

It is mentioned in the original document that the tests consist on a web browsing session with a duration of 5 minutes, which is lower than what is expected from a regular user[53]. As we explain below, beaconing activity is defined by constant times between packets over time and constant packet sizes over time. Therefore, the question of whether DoH connections still present beacon-like characteristics over longer periods of time can be asked.

The result given for RITA is interesting nonetheless, as it yields a hints towards the characteristics of DoH traffic, at least at the time scale of the experiment presented in this study. By analyzing the tool's source code[2], the characteristics deemed beacon-like by RITA are, among others:

- Small packet sizes.
- Low skewness on packet sizes.
- Low skewness on time between packets (constant throughput).

Finally, on top of the proposed methods proposed in the aforementioned research, there are two other approaches that could be taken to detect DNS over HTTPS traffic. Both of these consist on keeping updated blacklists of both IP addresses and SNI (Server Name Indication) values [54].

For <u>IP addresses</u>, it would be trivial to compile a list of known DoH resolvers and monitor connections towards it. While this could be a very simple to implement approach, right now there are over 40 known resolvers [55], and it is reasonable to

[2] https://github.com/activecm/rita/blob/master/pkg/beacon/analyzer.go

expect this number to increase over time, given how young the protocol is and the recent adoption by major browsers, so this solution might not scale over time. Moreover, a simple HTTPS proxy would completely defeat this approach.

As for the SNI value, it is a TLS extension which indicates the hostname the client is trying to connect to, so when it is sent unencrypted, it leaks information about the communication; the string sent could be used to blacklist DoH resolvers. On top of having the same issue as with IP addresses, where an updated database needs to be maintained, other problems arise, such as the possible absence of this field, or it being encrypted, as it has been recently proposed [56]. Even in its presence, the modification of this field for mischievous purposes is already a state of the art technique [57].

Overall, none of the previous solutions seem to be viable, complete or clarifying enough. The first two approaches, TLS inspection and application logging, involve a violation of user privacy and excessive administrative control, which might not be possible due to technical or bureaucratic reasons. On the other hand, server fingerprinting, IP address collection and SNI monitoring are reactive solutions to a young and evolving problem, and do not seem to scale on the long run. Moreover, these can be circumvented easily with the use of HTTPS proxies, which conveniently work with DoH traffic. Finally, a more pure traffic analysis solution is proposed through the use of Zeek and RITA, but no quantifiable metrics are given.

3. Methodology

DNS over HTTPS is a mechanism intended for use in web browsers, given their easier interoperability with web APIs, allowing for a self-contained domain name lookup mechanism, as opposed to relying on the underlying operating system. Therefore, using a web browser to generate DNS over HTTPS traffic not only the easiest, but the most adequate technique to obtain this kind of traffic. The characteristics of both HTTPS and DoH connections generated this way will match that of a regular user.

Web browsers employ TCP (Transmission Control Protocol) to open connections to web servers, with the intent of sending and receiving HTTP requests and responses. These requests are cryptographically protected by TLS. When using DoH, some of these connections will transport HTTP traffic to and from regular web servers, whereas others will transport HTTP-encapsulated DNS data.

The proposed methodology is to automate a web browser to visit a sequence of websites, generating in the process both HTTPS and DoH traffic. From this traffic, each connection is analyzed, and several macroscopic features will be extracted, hoping that they will be of use to characterize the difference between DoH and regular HTTPS. This is, therefore, a binary classification problem.

Section 3.1. Experiment design explains the tools and techniques used to generate encrypted traffic through the use of web browsers. Section 3.2. Feature extraction and selection offers an overview of the selected traffic features and the reasoning behind their selection. Section 3.3. Classification algorithms and evaluation methodology gives an explanation of the selected machine learning classification algorithms, as well as the metrics used for their evaluation.

3.1. Experiment design

As previously explained, the most accessible and adequate method to obtain user-like web traffic is by using a browser. We make use of the Selenium library [58] to automate a DNS over HTTPS-supporting web browser, Mozilla Firefox version 68.8.0esr. During the experiment, the browser will visit 1500 randomly selected web sites from the first 10.000 sites [3] in the Alexa Top ranking [59] as of March 29th, 2020.

Selenium is a popular library available in several programming languages that allows control and automation over web browsers. It acts as an interface between the user and a binary executable which called WebDriver [60], hiding the internal workings of the specific browser implementation through a standardized interface. Figure 7 shows the

[3] https://gist.github.com/chilts/7229605

aforementioned schema. Several browser developers provide a WebDriver binary along with their product, like Chromium [61], Mozilla Firefox [62], Opera [63] and Internet Explorer [64]. We use Selenium 3.141.0 with geckodriver (Firefox) version 0.26.0.

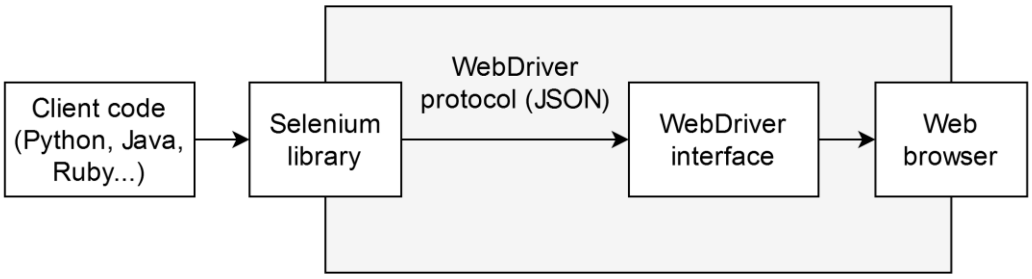

Client code (Python, Java, Ruby...)
Selenium library
WebDriver protocol (JSON)
WebDriver interface
Web browser

Figure 7

To activate the use of DNS over HTTPS, Mozilla Firefox has several configuration options under the `network.trr` section [65] (TRR stands for Trusted Recursive Resolver). Of those options, two need to be specifically configured:

```
network.trr.mode = 2
network.trr.uri = https://mozilla.cloudflare-dns.com/dns-
query
```

The mode option specifies to use DoH whenever possible, which is the default value when a user enables DoH through the graphical interface. The URI parameter allows us to select the specific recursive resolver; in this case Cloudflare is selected as it was the first one to be added to Firefox [18].

During the initial experiments, it became apparent that web browsers actually keep DoH connections open for longer periods of time than usual HTTPS connections. This is presumably done to avoid closing and reopening a TLS session and the underlying TCP connection, which incur in a time overhead due to protocol handshakes. Web browsers then try to maintain these connections active for long periods of time, so that when a DNS request needs to be sent, the channel is immediately ready. While this behavior is characteristic of DoH connections, it hinders our ability to obtain a significant amount of samples to characterize them. If the experiment consisted on visiting sequentially a number of web sites, the number of DoH connections would be substantially low, especially when compared to the number of regular HTTPS samples.

To avoid this effect during our experiments, but without limiting the duration of these connections substantially, a waiting mechanism was introduced. After visiting a web site, the browser will halt for a period of time, selected from a discrete uniform distribution between 0 and 150 seconds before visiting the next site. This will cause

DoH connections to be closed at random times due to inactivity, and therefore increasing the number of final samples.

To reduce the time taken to perform the experiment, two browsers were launched in parallel, dividing the workload between them. Each of them is independent, and therefore will not share any DoH connections to perform queries; this would not be true if tabs within the same browser were employed.

Note that this waiting mechanism does not intend to be a realistic behavior model. While a user might display a similar load-then-browse pattern, the waiting parameter is chosen only as a means to increase the number of DoH connection samples, while at the same time allowing them to have their usual above average time durations.

Once the browser is ready to be launched, traffic needs to be captured. To do this, the Tcpdump [66] utility is used with a capture filter, as we focus on traffic directed to port 443. Note that, since traffic can be filtered with a simple capture filter, even in a real world scenario, there is no need to introduce interfering traffic as suggested in [30]. The output file has a PCAP format that can be analyzed by other tools. Specifically, for traffic processing, a closed-source tool internal to the Telecommunications Networks and Services research group at the Public University of Navarre is used. This tool reads the network trace file, reconstructs TCP streams and outputs information in text format.

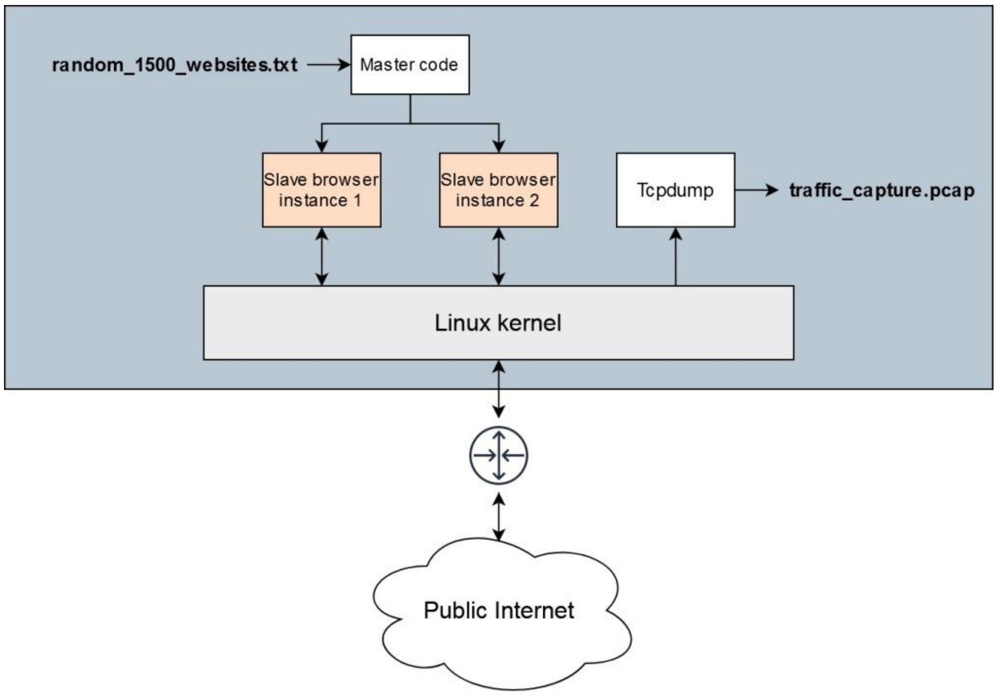


Figure 8

Figure 8 shows a high level schema for the proposed experiment. As explained, two browsers are driven in parallel to request a total of 1500 sites between both, pausing for random periods of time. At the same time, Tcpdump will capture all of the traffic generated to be processed afterwards.

After processing the network trace, each connection is taken as a sample, and labeled according to its destination IP address. Each sample can have one of two labels: DNS over HTTPS (the Cloudflare resolver's destination IP address) or HTTP (any other destination address).

3.2. Feature extraction and selection

In total, 29 features are extracted for each sample in the dataset, listed below:

- Number of requests
- Number of responses
- Mean, standard deviation (STD) and skewness of request sizes
- Mean, STD and skewness of response sizes
- Mean, STD and skewness of request durations
- Mean, STD and skewness of response durations
- Request time delta (time between requests), skewness
- Response time delta (time between responses), skewness
- Number of concurrent connections to the same host at the time of start
- Connection duration (seconds)
- Number of data packets from client to server
- Number of data packets from server to client
- Average number of data packets per second from client to server
- Average number of data packets per second from server to client
- Average number of bytes per second from client to server
- Average number of bytes per second from server to client
- Client to server channel occupation percentage
- Server to client channel occupation percentage
- Average number of bytes per packet from client to server
- Average number of bytes per packet from server to client
- Negotiated ALPN

As an approximation, it is assumed that every uninterrupted burst of packets from client to server is a request, and that every burst from server to client is a response, as seen in [67]. Figure 9 illustrates this concept. With each line representing a data packet, a burst is comprised of an uninterrupted sequence of packets in each direction. For each burst, its size (the sum of the sizes of all of its packets) and duration is collected; then, the mean, standard deviation and skewness for each of these values are computed.

Using this very same approximation for requests and responses, a time delta (time between messages) for requests and responses can also be extracted, as shown in Figure 9. By calculating the skewness of these values we expect to distinguish beacon-like patterns as seen in section 2.3.

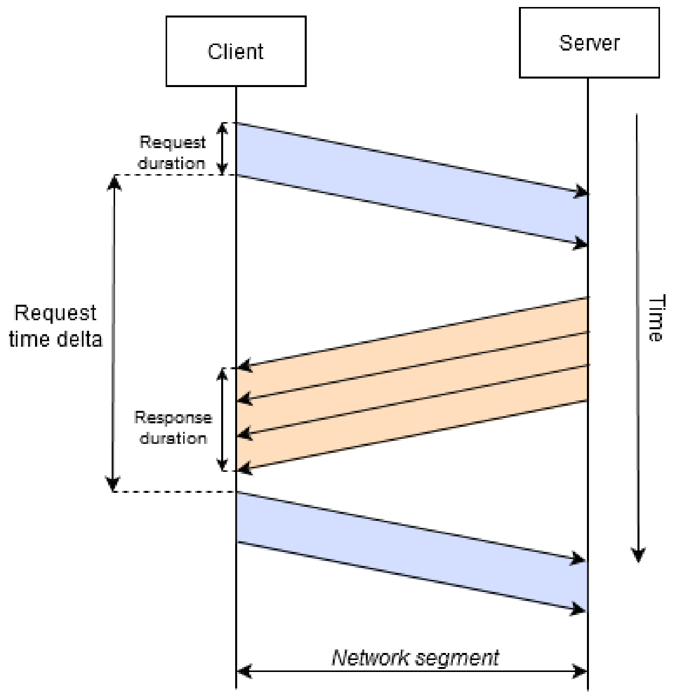


Figure 9

Each sample has an associated start and end timestamp, corresponding to the first and last packet seen. For any given sample, the number of connections open to the same server at the time of start can be calculated. This value is expected to be relevant, as it makes sense for browsers to keep one or two DoH connections open at maximum to send requests over; this number usually grows bigger for HTTP traffic, as a browser needs to request a number of resources in the parsed HTML document for the web page it is visiting.

An schema for the channel occupation metric is given in Figure 10. For both connection endpoints, the time spent sending data is calculated. This is done by taking the first and last timestamp of each packet burst seen in each direction. Then, the total time spent sending by each endpoint is summed and normalized to the connection duration. Therefore, channel occupation is given as a percentage.

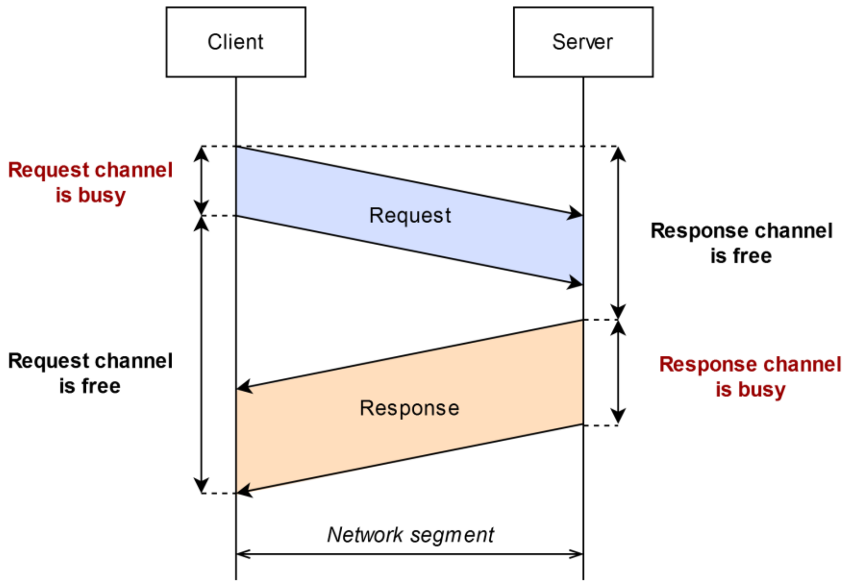


Figure 10

Note also that, since measurements are done at the client machine, the occupation for the response channel is not perfectly estimated, as effects along the network pathway could alter the time and order of the data packets.

Finally, the last characteristic extracted for each sample is the negotiated ALPN (Application-Layer Protocol Negotiation Extension)[68]. This extension is announced by both the client and the server during the TLS handshake (Figure 11) to establish the application protocol to be used over TLS; in web environments, this means that either HTTP/1.0, HTTP/1.1 or HTTP/2 will be announced, if anything at all. This parameter is relevant since, per DoH's specification, HTTP/2 is the minimum recommended version to be used, as to use its benefits such as parallelism, header compression and server push.

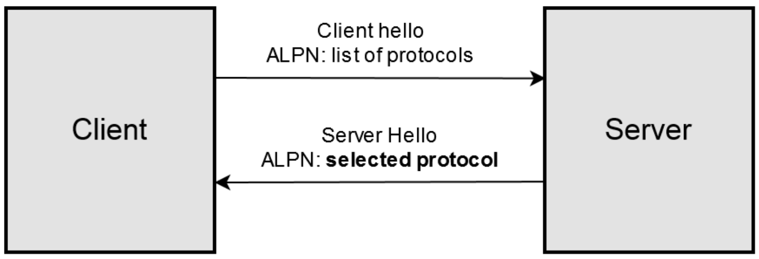


Figure 11

Once the feature list above is extracted for each sample, feature selection needs to be performed as to filter out less representative features and improve the classification algorithms' performance. One of the most proven methods to do so is through the use of Pearson's correlation coefficient [69] [70]. This metric evaluates the linear association of two variables, given samples for both. The use of this tool is double: first, to find out which features have a low correlation to the output, this filtering the ones that are not relevant to the current problem, and second, to detect which extracted features are highly correlated between them, so as to not feed redundant data to the classification algorithm.

Pearson's correlation coefficient for two variables, when applied to a population of *n* samples takes the following form:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where:

- r_{xy} represents the correlation between variables *X* and *Y*.
- n is the sample size.
- $\bar{x}$ and $\bar{y}$ are the values for the sample mean for variables *X* and *Y*.
- x_i and y_i are the i-th samples for variables *X* and *Y*.

The final result, for m variables, is an $m \times m$ matrix that holds the correlation values between each possible pair of variables. This matrix is symmetric, with the diagonal having always a value of 1, as it represents the comparison of a variable to itself. Given that the value $r_{i,j}$ is the correlation between variables *i* and *j*, *r* will have a value of 1 if $i = j$.

Based on this matrix, we can filter out features that have a high correlation with others, as well as features that have a low relationship with the output.

3.3. Classification algorithms and evaluation methodology

The selection of machine learning algorithms is varied across existing literature, as seen in sections 2.12.1. Traffic analysis and machine learning and 2.2. Some of the most common algorithms in the field of traffic analysis are Naïve Bayes, k-Nearest Neighbors and Random Forest [71] and are chosen for this experiment for that reason. Additionally, the AdaBoost classifier [72] is also included in the tests, as it is an ensemble classifier, like Random Forest. We proceed now to give an overview of the characteristics of these algorithms, keeping in mind that we are dealing with a binary classification problem. Multiclass classification (more than two output labels) is not discussed.

The first two algorithms mentioned are arguably the less complex. Naïve Bayes classifier specifically is one of the most simple algorithms in machine learning. It makes use of Bayes's theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where P(H|E) is the probability of H (hypothesis), given E (evidence). Given a set *X* of *n* features, the probability for output label *y* can we expressed as:

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y) \dots P(x_n|y)}{P(x_1) \dots P(x_n)}$$

With the previous expression, a probability can be calculated for each possible output, for each sample, by substituting *y* and $x_1 \dots x_n$ respectively.

K-nearest neighbors, on the other hand, works by calculating distances in an *n*-dimensional space, where *n* is the number of input features. For example, for n=1, the output label of a sample is assigned by voting between the k closest points along that single dimension (a straight line). The weight of each neighbor's vote is *1/k*, meaning that the closest neighbor has a weight of 1, the second closest a weight of 0.5, etc. *k*, of course, is a configurable parameter for the algorithm.

A different type of algorithm that appears all along machine learning literature, including the one related to traffic analysis, is decision trees, and in particular, classification trees. There are several types of decision trees, and we will not go into their differences, but some common variants are ID3, C4.5 and CART [73].

Decision trees are formed by a root node, a series of branches and other nodes, and a series of outputs or leafs. Each node splits into a *n* branches based on a feature, and tries to form *n* groups, both as homogenous as possible. Figure 12 shows a sample decision tree of two levels.

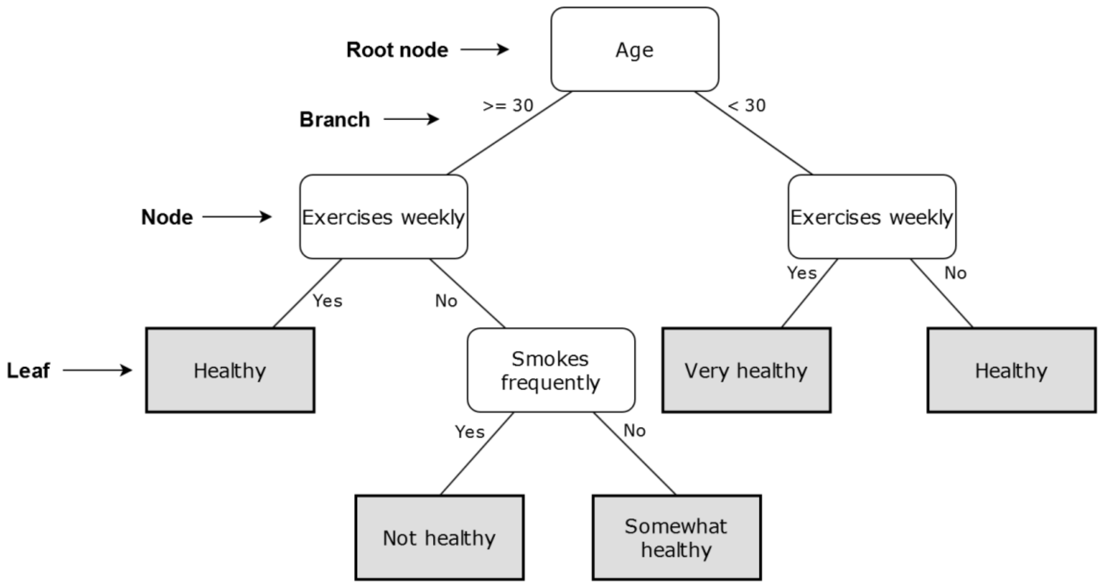


Figure 12

The homogeneity of the new *n* groups after a split is measured with one of two criteria, either the Gini impurity or the entropy coefficient. For *j* output labels, the Gini impurity of a sample group is determined by:

$$g = 1 - \sum_j p_j$$

The entropy coefficient, on the other hand, is obtained with the following expression:

$$e = \sum_j p_j \log_2 p_j$$

p_j represents the probability of randomly selecting a sample with label *j*. For example, in an evenly distributed dataset with two labels, the probability of choosing a sample of each label is 0.5. If, after a node split, one of the resulting groups is formed only by samples with the same label, that group becomes a leaf with perfect purity (a value of 0 with both Gini and entropy criteria). This obviously means that future prediction attempts that end up at this leaf should be always correctly classified, based on training data. On the other hand, low purity branches are likely to split again to increase purity.

Despite their ease for interpretation, the use of decision trees as single classifiers does not seem to yield the best results, as seen in the literature reviewed in sections 2.1 and 2.2. However, their use in ensemble classifiers is very common, as explained below.

Ensemble classifiers aggregate a number of base classifiers and provide predictions by performing some calculation with the results from the base classifiers. The goal of grouping base classifiers is to add flexibility and at the same time increase robustness towards bias. Despite this similarity, both ensembles use a different strategy; Random Forest uses what is known as bagging, while AdaBoost uses boosting.

Bagging is the simpler approach: each base classifier is trained in parallel, independently of the others. During the prediction stage, the output of the algorithm is decided by choosing the most common answer among all estimators, that is to say, an unweighted vote is carried out.

In boosting, however, each base estimator is trained taking into account the error rate of the previous trained classifier, in hopes to learn from past mistakes by other estimators. This is done in an adaptive way: initially, all training samples have the same associated weight. For each base estimator trained, the weights of wrongly classified samples are summed and normalized to the total number of samples (also known as weighted error rate, or *e*). Then, the weight of the current base classifier, *cweight*, is computed as follows:

$$cweight = lrate * \log\left(\frac{1-e}{e}\right)$$

Where *lrate* is the learning rate parameter for the algorithm. A high classifier weight implies more decision power when predicting the output of new samples. Next, each wrongly classified sample has their weight, *sweight*, increased according to the following expression:

$$sweight = sweight^{cweight}$$

This, of course, implies that wrongly classified samples will increase in weight, in turn increasing the error rate of subsequent classifiers that wrongly classify it as well. During the final prediction stage, the output of each classifier is multiplied by its weight to obtain a final answer (weighted vote).

Another key difference is that, in Random Forest, the base classifiers are decision trees trained with a random subset of the input features. In AdaBoost, these can be any generic estimator, although usually it is formed also by decision trees. The decision trees used for AdaBoost are normally very shallow, with likely two or even one layer of depth (also known as stumps).

Once the algorithms are chosen, evaluation methods need to be selected. In the existing research, the prevalent metrics for evaluation revolve around the confusion matrix [74], and metrics calculated from it: precision, recall and F-measure [75].

A confusion matrix takes the following shape:

$$\begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix}$$

After an algorithm classifies N samples from a verification set, these outputs can be verified against their actual label. If the label given by the algorithm matches the original one, the sample is correctly classified. In binary classification problems, one of the output labels is called "positive" and the other "negative". With this in mind, TP stands for True Positives, FP for False Positives, TN for True Negatives and FN for False Negatives. Given the previous explanation, these values measure the number of correctly and incorrectly classified samples for each of the two classes.

By normalizing the confusion matrix values to the total number of samples in the verification set for each label, the following values can be computed:

$$TPR = \frac{TP}{N_p};\ FNR = \frac{FN}{N_p};\ TNR = \frac{TN}{N_n};\ FPR = \frac{FP}{N_n};$$

Where N_p and N_n are the number of positive and negative samples in the set. TPR stands for True Positive Rate, FPR for False Positive Rate, and so on.

The precision, recall and F-measure metrics are based on these previous values as well:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$f\text{–}measure = \frac{(1+\beta^2) \times recall \times precision}{recall + precision}$$

The precision metric offers a quantifiable measure of the classifier's ability to discern false positives from true ones. A low precision metric indicates that the algorithm is identifying a great amount of negative samples as positive. Recall, on the other hand, measures the classifier's ability to find positive samples. A low recall indicates that a great amount of positive samples are being overlooked.

Finally, the f-measure can vary depending on beta (β). It gives a weighted harmonic mean of recall and precision, weighing recall more than precision by a factor of beta. In our case, we set beta to 1, meaning that both recall and precision are equally important.

As a visual aid, an additional metric is the Receiver Operating Characteristic (ROC) curve, mainly used to measure the quality of a classifier. However, this type of curve does not seem to be representative enough with binary classification problems where the output labels are unbalanced [76]. Our output classes are DoH (1, or positive) and HTTPS (0, or negative), and, as will be later shown, the number of negatives outweigh the number of positives.

As an alternative, the literature suggests the use of the precision-recall (PR) curve, which plots pairs of these two metrics for different probability thresholds. When a classifier algorithm takes a sample, it assigns a probability for each of the output labels. A probability of 1 for a label indicates that the current sample can definitely be mapped to that label. By default, the threshold to separate between both output classes is 0.5 (probabilities below 0.5 are assigned to one class, and above 0.5 to the other). However, for highly imbalanced datasets, where one class outweighs the other in number, this threshold might not be beneficial. The precision-recall curve plots points with coordinates for different threshold values, with precision and recall as their coordinates [77].

An unskilled classifier will be plotted as a straight line, with a Y-coordinate proportional to the number of positive samples in the dataset (as the precision is plotted on the Y axis). A perfect classifier, on the other hand will be plotted as a straight line at Y=1.

For each curve, a metric known as average precision (AP) [78] can be calculated as a weighted mean of precisions as each decision threshold. The weight for each precision value can be obtained from the increase in recall since the previous threshold:

$$AP = \sum_{n} (R_n - R_{n-1}) P_n$$

Where R_i and P_i are the recall and precision values for the *i*-th decision threshold.

In order to get an accurate measurement of the skill of a machine learning model without falling into biases, a technique known as k-fold cross validation is used [79]. Using this technique, the initial dataset is divided into k groups (or folds) of the same size. Then, *k-1* folds are used for training, and the remaining one is used for verification. This is done *k* times until every fold has been used as the testing set. For each iteration, the metrics described above (precision, recall, f-measure, AP) are calculated and averaged, and one PR curve can be plotted. Using this technique, the selected classifiers can be compared so the best one can be selected to use in a hypothetical real world scenario.

On top of quantifying the quality of each of the proposed classifiers, speed benchmarks are taken for each algorithm, measuring the time taken to classify a number of samples, in order to evaluate feasibility for real-time classification.

4. Results

In this section, the result of the exposed experiment is presented. First, general statistics about the obtained traffic capture are given. Then the results for feature selection are presented, as well as a list of the resulting features used for stream classification. Next, an analysis on the selected features is shown, and finally classification algorithms are evaluated over the exposed data through the use of metrics described in section 3.3.

4.1. Dataset analysis and feature selection

After executing the experiment presented in the previous section, a traffic trace of approximately 5.98 GB is obtained, amounting to a total of 54955 TCP connections transporting data. 198 of them are identified as DoH (0.36%), and 54757 are regular HTTP (99.64%). Note that, even with the addition of the browser halt mechanism introduced in the previous chapter, the number of DoH samples is low. However, as will be shown later, the sample size is enough to analyze and perform detection.

As mentioned in section 3, Pearson's correlation coefficient is used to perform feature selection, both by examining feature-to-feature correlation and feature-to-output correlation. To do so, we examine the correlation matrix presented in a color-coded manner in Figure 13. Note that the absolute value of the correlation coefficient is used for simplicity, as values closer to |1| indicate stronger correlation, independently of sign.

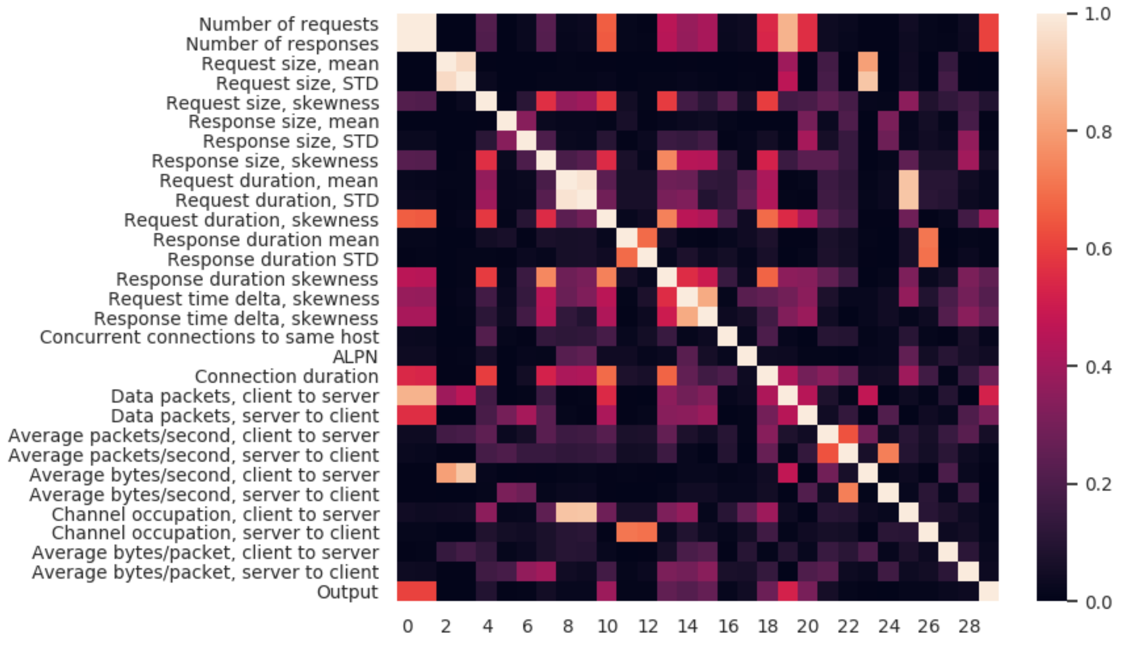


Figure 13

First, we aim to reduce the number of features by examining the correlation of each variable to the output. Once a list of the most significant variables is acquired, the objective is to filter out redundant variables.

The correlation of each variable to the output (the last column in Figure 13) is shown in Figure 14.

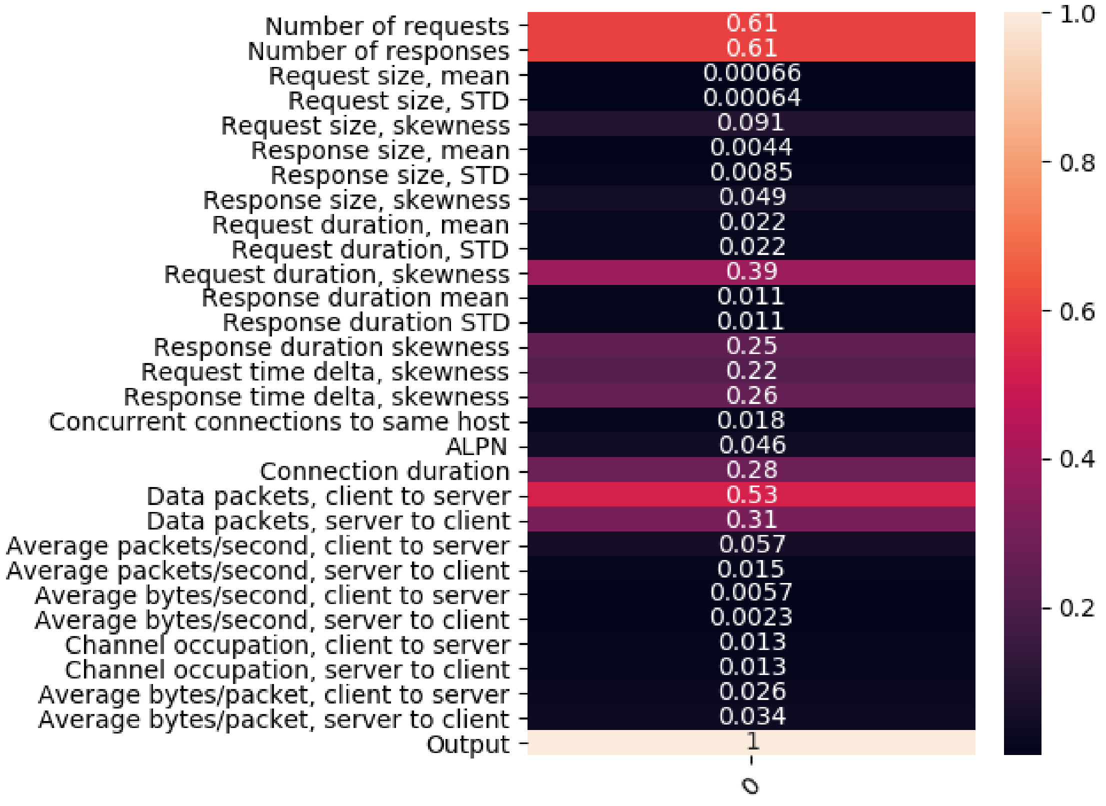


Figure 14

Clearly the variables related to the length of the connection stand out: number of requests and responses, connection duration, number of data packets from client to server and server to client, etc. Additionally, other variables less related to total duration, as request and response duration skewness seem relevant.

Of these first 29 features, the best 15 are selected, reducing the number of features to roughly half of the original ones. These 15 features are listed below, ordered by their correlation to the output in absolute value:

1. Number of responses
2. Number of requests
3. Data packets, client to server
4. Request duration, skewness
5. Data packets, server to client
6. Connection duration
7. Response time delta, skewness
8. Response duration skewness
9. Request time delta, skewness

10. Request size, skewness
11. Average packets/second, client to server
12. Response size, skewness
13. ALPN
14. Average bytes/packet, server to client
15. Average bytes/packet, client to server

With the filtered set of features, the correlation matrix results in the one shown in Figure 15.

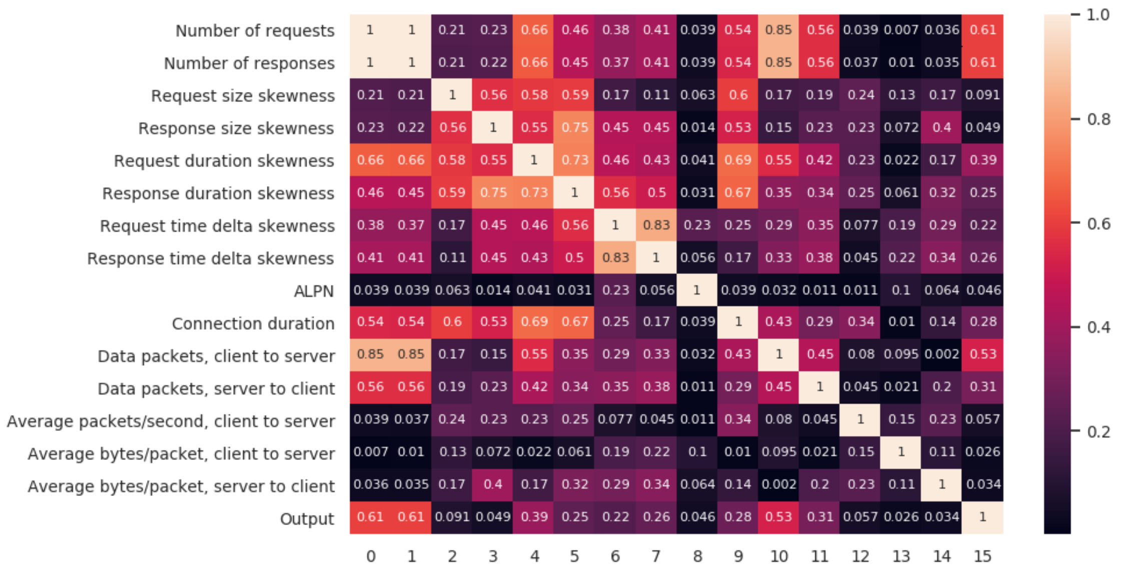


Figure 15

An obvious pair of correlated variables are the number of requests and responses, which share a correlation coefficient of 1. This is due to the methodology estimating requests and responses, explained in chapter 3. However, since both DNS and HTTP are request-response protocols, this is an expected result even in the absence of the mentioned approximation.

Another pair of highly correlated features are the number of requests and data packets, with a value of 0.85. The first one is computed from the second one, so this result is also expected.

Finally, the time delta skewness of requests and responses are highly correlated as well. Given that the reception of a response usually triggers the next request in the client, and that the round trip time (RTT) of the connection should remain mostly stable, the time between the reception of responses and sending of requests should be highly similar, as the first triggers the second; a delay in the reception of a response incurs in a delay in the transmission of a request, and thus, both becoming correlated.

After filtering these three features with high correlation values with others (number of responses, number of data packets from client to server, response delta skewness), the final set of selected features is, ordered by output correlation:

1. Number of requests
2. Request duration skewness
3. Data packets, server to client
4. Connection duration
5. Response duration skewness
6. Request time delta skewness
7. Request size skewness
8. Average packets/second, client to server
9. Response size skewness
10. ALPN
11. Average bytes/packet, server to client
12. Average bytes/packet, client to server

4.2. Feature analysis

As explained in previous sections, lengthy connections are characteristic to DoH. To investigate this fact, we plot a histogram of for connection duration in Figure 16.

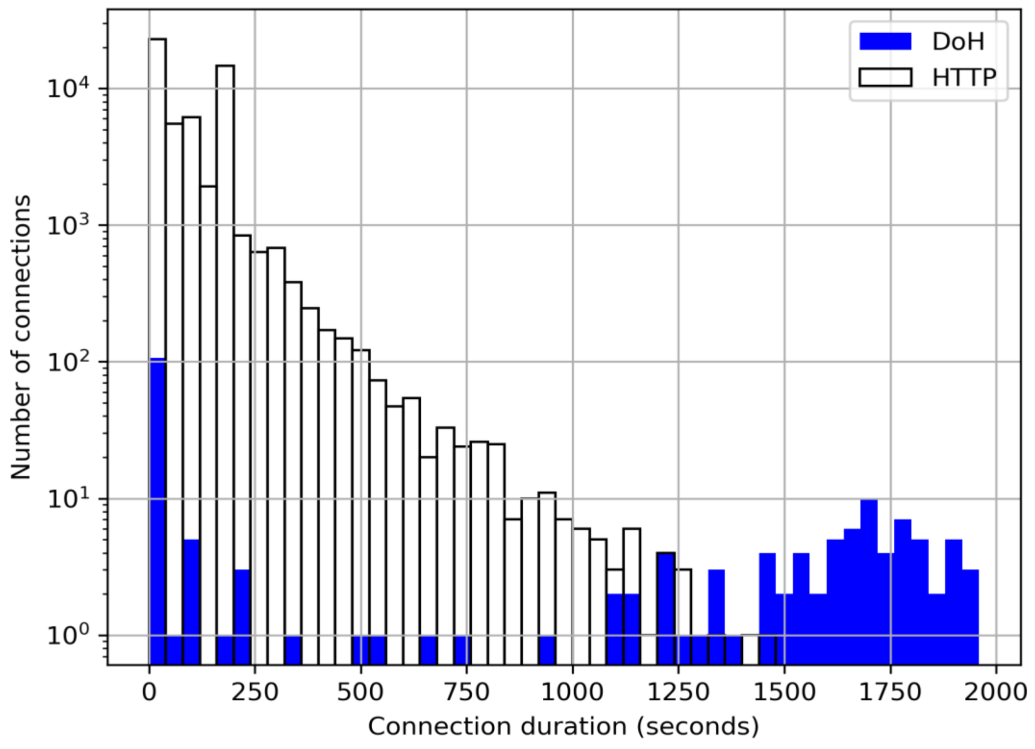


Figure 16

Note that the Y-axis is shown in a logarithmic scale, as there are much more HTTP connections, especially around the first values of the X-axis. Therefore, to be able to show both distributions in the same graph, this scale is used. However, the key piece of information is the distribution of DoH connections to the right of the graph. To more accurately display the distribution of connection lengths, the cumulative distribution function (CDF) is shown for the same variable in Figure 17.

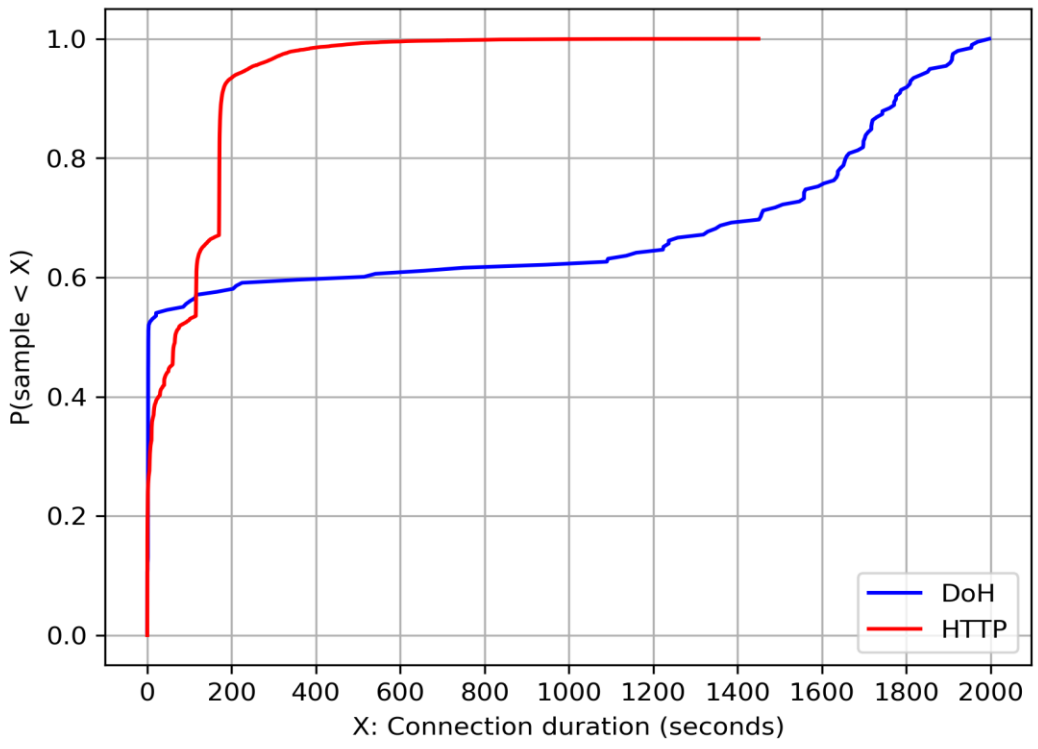


Figure 17

The CDF shows that over 90% of the HTTP connections last less than 200 seconds, while only around 58% of the DoH samples meet this boundary. On average, an HTTP connection lasts for 94 seconds, while a DoH stream lasts for 644 seconds. As a TCP stream containing TLS data is kept for longer and longer periods of time, it will be more likely that it transports DoH data.

The next most prominent feature is the request duration skewness. On average, a DoH stream has a request duration skewness of 8.70, while this value drops to 0.69 for HTTP. In fact, 43.43% (86) of DoH samples have their request duration skewness above 5, while only 0.289% (158) of the HTTP samples match this criterion. The difference regarding this feature becomes clear when plotted against the connection duration (Figure 18).

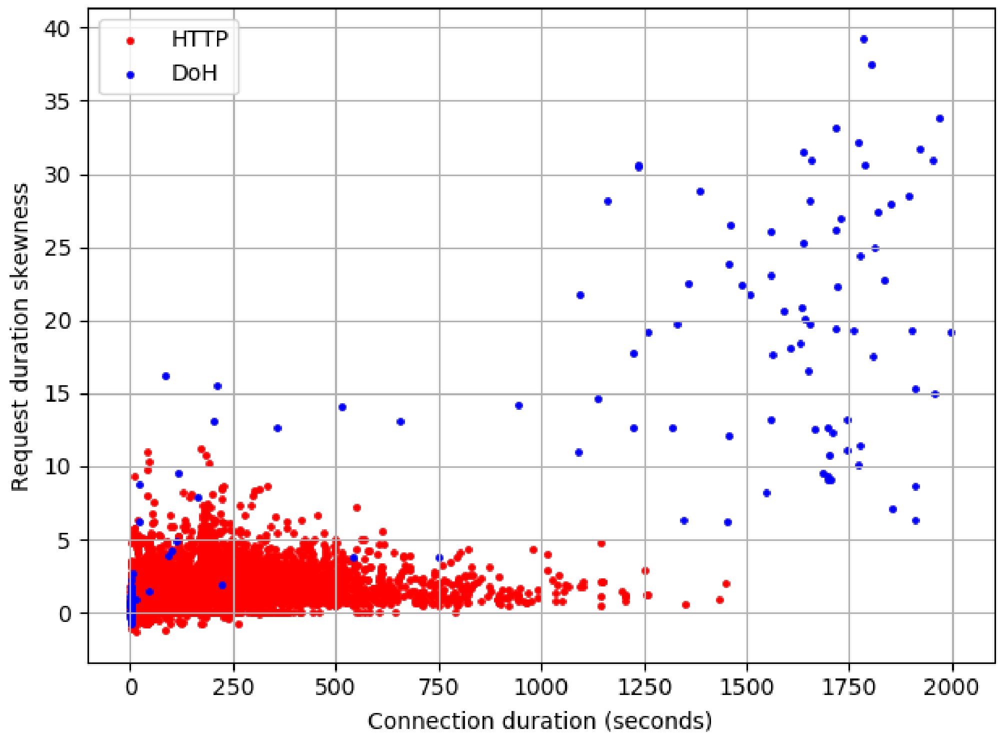


Figure 18

The request time delta (time between packet bursts from client to server) seems to be relevant as well. When plotted against the number of requests (Figure 19), a tendency becomes clear: as new requests appear in the connection, the skewness grows and then stabilizes. This is most likely because the first packets are TLS control messages, which are smaller than the transmitted data, and therefore the distribution skews to the right of the median. On average, a DoH connection displays a 3.416 request time delta skewness, while an HTTP connection only averages to 0.189.

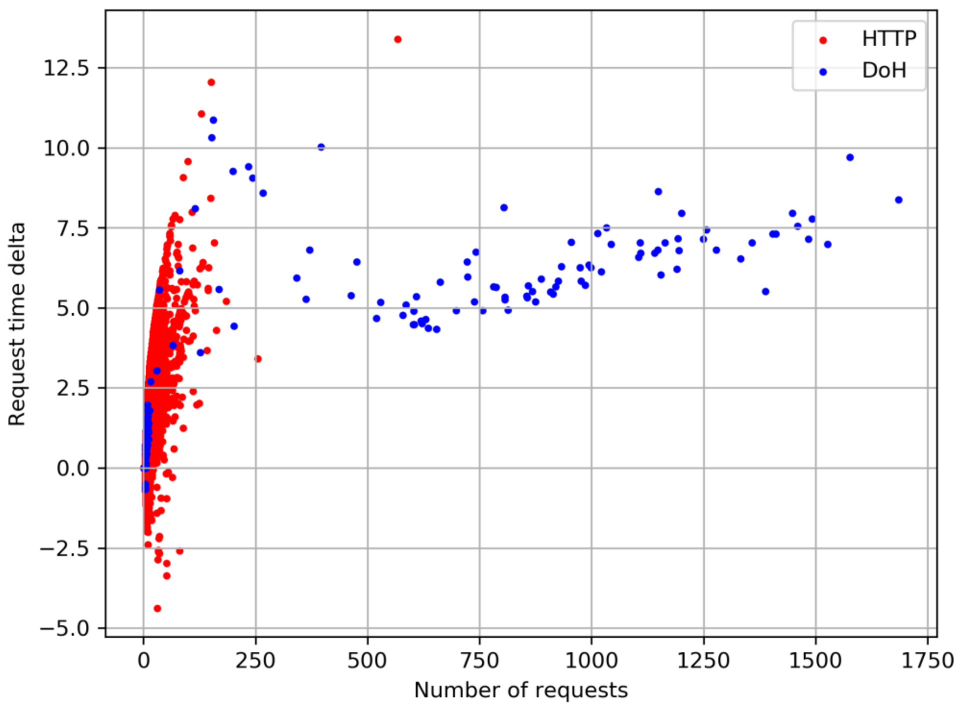


Figure 19

Another interesting feature is the ALPN extension. This value is encoded as 2 (HTTP/2), 1 (HTTP/1) or 0 (not seen). The browser automatically uses HTTP/2 to connect to Cloudflare's resolver, so the ALPN extension is never seen in DoH connections. This can clearly be distinguished if plotted against the number of requests, as Figure 20 shows.

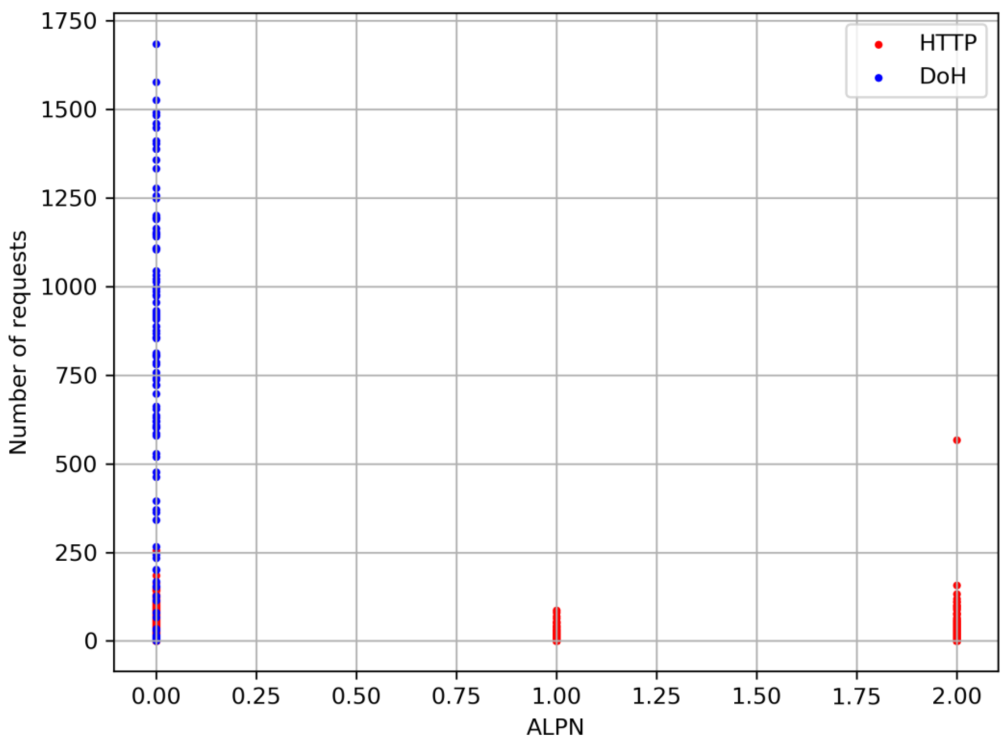


Figure 20

A common issue when dealing with a high number if features is obtaining a complete representation of all of them. A common technique to bypass this problem is the use of Principal Component Analysis (PCA) [80] to reduce the dimensionality of the original data. The details of the algorithm are technical and not relevant at the moment. The key piece of information is that PCA allows the representation of n-dimensional variables in a lower number of dimensions, usually 2 for easier representation. The new dimensions represented in such plots are components that hold enough information about the original variables.

When PCA is applied to a n-dimensional dataset, it can allow a more clear identification of output classes, in order to verify that they are actually distinct. Figure 21 shows the PCA representation for our current features.

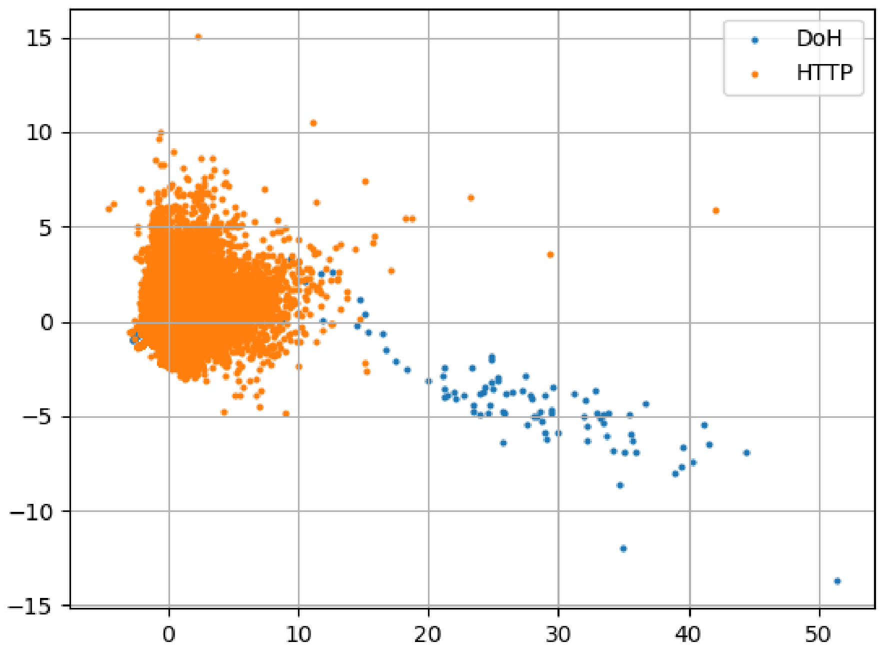


Figure 21

In a scenario where bad feature selection is made, or the output does not properly correlate with the input, visual identification of the output labels is not possible. However, in our PCA plot both classes are clearly identifiable, as HTTP forms a dense cluster and DoH spreads to the right of the plot.

4.3. Algorithm evaluation

As explained in section 3.3, a set of classifiers will be tested using k-fold validation. Since there are a total of 198 DoH samples, using 5 folds (k=5) seems appropriate, as every fold will have, on average 39.6 DoH samples. For these tests, we consider DoH a positive sample, and HTTPS a negative one.

In total, the list of classifiers to be tested is listed below. For the AdaBoost ensemble, classic decision trees of depth one are chosen as base estimators.

- Naïve Bayes Classifier
- K-Nearest Neighbors (k=3, 5, 15)
- AdaBoost (100, 200, 400, 600 estimators, entropy criterion)
- AdaBoost (100, 200, 400, 600 estimators, Gini criterion)
- RandomForest (100, 200, 400, 600 estimators, entropy criterion)
- RandomForest (100, 200, 400, 600 estimators, Gini criterion)

Gini and entropy criteria are explained in section 3.3. For each test, (taking 4 folds as training and 1 as testing data), a confusion matrix is generated. Instead of displaying the 5 confusion matrixes, the values for each one are summed, as each holds the results of classifying 1/5th of the total data:

$$M = M_0 + M_1 + M_2 + M_3 + M_4$$

Where M_i is the confusion matrix for the fold i. Each member of the resulting matrix is normalized to the number of samples of that class, giving a percentage or rate; true positives and false negatives are divided by the number of total DoH samples (198), giving the true positive rate (TPR) and the false negative rate (FNR); the same is done with the true negatives, false positives and total HTTPS samples (54757).

For the precision, recall and f-measure metrics, an average over the five tests is shown.

Classifier	**TPR**	**FNR**	**TNR**	**FPR**	**Precision**	**Recall**	**F-measure**
Naïve Bayes	56.061	43.939	99.836	0.164	0.5568	0.5604	0.5572
KNN (k=3)	92.929	7.0710	99.991	0.009	0.9742	0.9294	0.9508
KNN (k=5)	92.929	7.0710	99.991	0.009	0.9742	0.9294	0.9508
KNN (k=15)	88.384	11.616	99.991	0.009	0.9725	0.8841	0.9257
AB (n=100, entropy)	95.455	4.545	**99.991**	**0.009**	**0.9758**	0.9546	**0.9642**
AB (n=200, entropy)	95.455	4.545	99.985	0.015	0.9628	0.9546	0.9572
AB (n=400, entropy)	94.949	5.051	99.973	0.027	0.9316	0.9495	0.9384
AB (n=600, entropy)	95.455	4.545	99.978	0.022	0.9456	0.9545	0.9482
AB (n=100, Gini)	94.949	5.051	99.989	0.011	0.9703	0.9495	0.9588
AB (n=200, Gini)	95.455	4.545	99.976	0.024	0.9407	0.9545	0.9457
AB (n=400, Gini)	95.455	4.545	99.98	0.020	0.9466	0.9545	0.9499
AB (n=600, Gini)	**95.960**	**4.040**	99.982	0.018	0.9511	**0.9595**	0.9547
RF (n=100, entropy)	91.919	8.081	**99.998**	**0.002**	**0.9944**	0.9191	0.9550
RF (n=200, entropy)	**92.929**	**7.071**	99.996	0.004	0.9895	**0.9292**	**0.9580**
RF (n=400, entropy)	92.424	7.576	99.996	0.004	0.9895	0.9242	0.9554

RF (n=600, entropy)	91.919	8.081	99.998	0.002	0.9944	0.9191	0.9550
RF (n=100, Gini)	89.394	10.606	99.984	0.016	0.9548	0.8940	0.9223
RF (n=200, Gini)	90.909	9.091	99.984	0.016	0.9551	0.9091	0.9305
RF (n=400, Gini)	89.394	10.606	99.993	0.007	0.9784	0.8940	0.9340
RF (n=600, Gini)	89.899	10.101	99.989	0.011	0.9692	0.8991	0.9321

As expected, Naïve Bayes and KNN underperform against the ensemble classifiers, due to the complexity of the task. The best values for each column are highlighted for both AdaBoost and RandomForest.

When the recall is plotted (Figure 22) a clear superiority is seen for the entropy criterion within RandomForest. For AdaBoost, however, entropy seems better only for lower number of base estimators, with two maximums at x=100 and x=200.

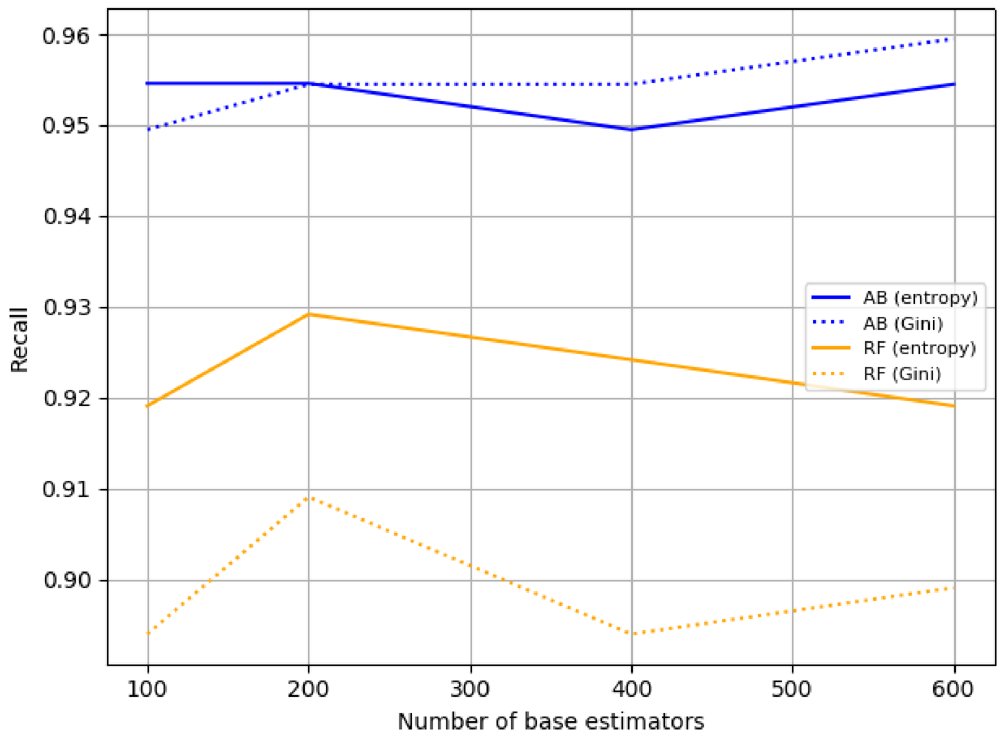


Figure 22

Looking back at the previous table, AdaBoost, there are two possible best choices, the first one being a classifier with 100 base estimators using the entropy criterion, and the second one having 600 base estimators using the Gini criterion. Even though the first one has a slightly lower true positive score (189 vs. 190 correctly detected DoH connections), it has half as much false positives (5 vs. 10), and takes a considerably smaller time to train (13.911 seconds vs. 62.476). The differences between both can even be considered random noise, due to their small magnitude, and therefore the first classifier is chosen.

As for RandomForest, the best configuration (200 base estimators and entropy criterion) is way safer on the false positive score (just 2 samples), while still having a high true positive value (184 out of 198 detected samples).

As mentioned in section 3.3, a PR (precision-recall) curve can be plotted for each iteration during the k-fold validation. Along with the datapoints for this curve, the AP (average precision) metric can be obtained. The PR curves along with the AP values for the best AdaBoost and RandomForest configurations are shown in Figure 23 and Figure 24 respectively.

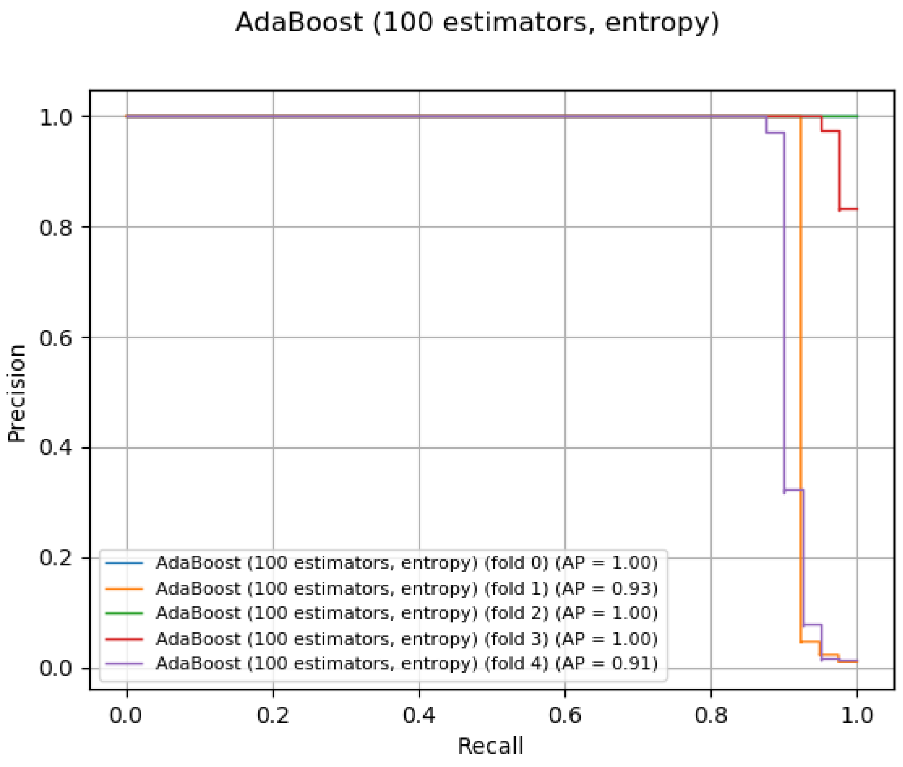


Figure 23

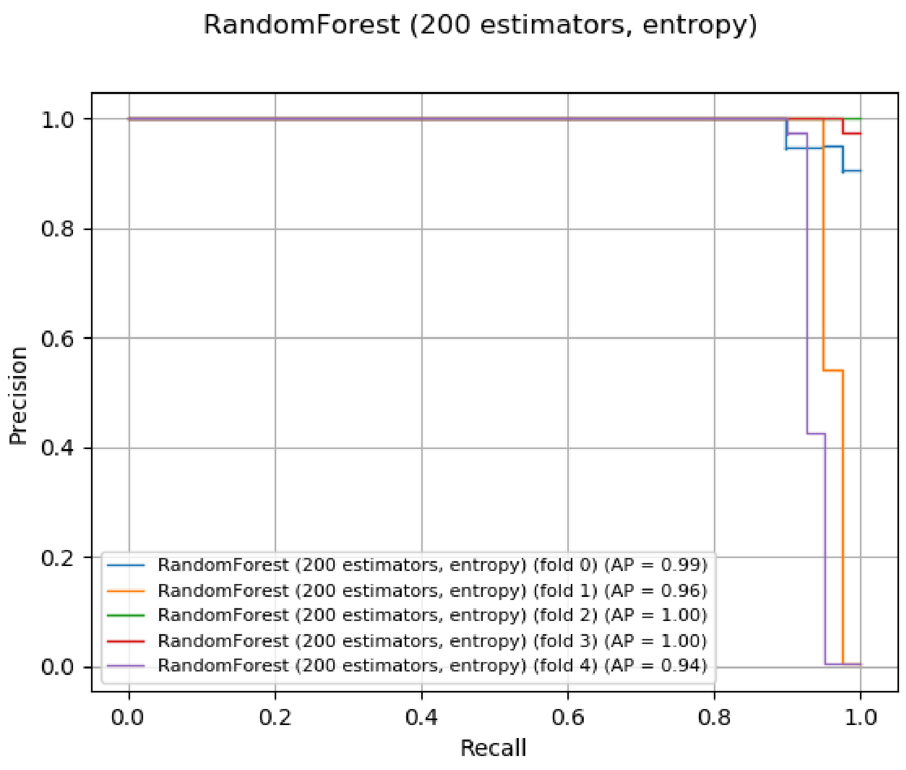


Figure 24

Clearly, folds with AP=1 are represented as straight lines, i.e., the classifier behaved perfectly for that subset of data. AdaBoost averages on a 0.968 precision score, while RandomForest has a slightly higher 0.978 average precision score.

Regarding performance, both algorithms have high classification throughput. Given that each fold has 10991 samples, and that one fold is used to classify:

Classifier	**Average time elapsed**	**Estimated throughput**
AdaBoost, 100 estimators, entropy	0.3640 seconds	30195 samples/second
RandomForest, 100 estimators, entropy	0.1418 seconds	77510 samples/second

Therefore, in a real time scenario, sample classification will not likely be the bottleneck in the pipeline.

5. Conclusion

5.1. Overview

In this thesis, DoH and HTTPS traffic was analyzed jointly, and characterized through a number of features, reduced to a total of 12. Treating each TCP stream as a single sample, through the review of existing literature and using the proposed machine learning algorithms, Random Forest and AdaBoost, true positive rates between 91 and 95.6% have been consistently achieved, along with false positive rates below 0.01%.

In conclusion, DoH traffic is highly detectable among regular HTTPS traffic, mainly due to lengthy connections and variety of message sizes and durations; this is specifically notable when looking at the asymmetry of the distributions of these features.

5.2. Further work

As further work, the proposal is to obtain more varied datasets, using different target websites, DNS resolvers, browsers (Chrome, due to its popularity) and DoH formats (GET, JSON, although they cannot be set from the basic configuration interface). There is no a priori reason to think that any of these variables will affect the results presented directly, other than slight variations in message sizes.

Furthermore, tests could be made where avoidance techniques are employed, such as shortening DoH connections, adding a constant sending rate or modifying the payload padding.

Finally, a real time system proof of concept is interesting. This system would need to follow a pipeline with the following shape:

```
traffic capture > traffic processing > feature extraction >
sample classification
```

A traffic processing intermediate is needed to reconstruct TCP streams and parse TLS messages, although omitting this element and working with raw packets could lead to a faster system with perhaps similar results. Of course, this traffic processor would need to dump regularly the state of opened connections in order to classify them before they are closed. As mentioned in section 4.3, the selected algorithms already support a high classification throughput in a single threaded environment, and so they would likely not be a performance bottleneck.

6. Bibliography

[1] Spanish Data Protection Agency (AEPD). (2019, November) DNS Privacy. [Online]. https://www.aepd.es/sites/default/files/2019-12/nota-tecnica-privacidad-dns-en.pdf [Accessed May 13, 2020]

[2] IETF, "RFC 7626: DNS Privacy Considerations," 2015.

[3] IETF, "RFC 1034: Domain Names - Concepts and Facilities," 1987.

[4] IETF, "RFC 1035: Domain Names - Implementation and Specification," 1987.

[5] Mauro Conti, Nicola Dragoni, and Viktor Lesyk, "A Survey of Man In The Middle Attacks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2027 - 2051, March 2016.

[6] IETF, "RFC 2535: Domain Name System Security Extensions," 1999.

[7] IETF, "RFC 7129: Authenticated Denial of Existence in the DNS," 2014.

[8] Information Technology Laboratory (National Institute of Standards and Technology). (2020, June) Estimating IPv6 & DNSSEC Deployment SnapShots. [Online]. https://fedv6-deployment.antd.nist.gov/snap-all.html [Accessed June 1, 2020]

[9] IETF, "RFC 1945: Hypertext Transfer Protocol - HTTP/1.0," 1996.

[10] IETF, "RFC 6101: The Secure Sockets Layer (SSL) Protocol Version 3.0," 2011.

[11] IETF, "RFC 2246: The TLS Protocol Version 1.0," 1999.

[12] IETF, "RFC 2818: HTTP Over TLS," 2000.

[13] IETF, "RFC 7858: Specification for DNS over Transport Layer Security (TLS)," 2016.

[14] IETF, "RFC 8484: DNS Queries over HTTPS (DoH)," 2018.

[15] IETF, "RFC 8427: Representing DNS Messages in JSON," 2018.

[16] Cloudflare, Inc. DNS over HTTPS. [Online]. https://developers.cloudflare.com/1.1.1.1/dns-over-https/ [Accessed May 13, 2020]

[17] Google Developers: JSON API for DNS over HTTPS (DoH). [Online]. https://developers.google.com/speed/public-dns/docs/doh/json [Accessed May 21, 2020]

[18] The Mozilla Blog. (2020, February) Firefox continues push to bring DNS over HTTPS by default for US users. [Online]. https://blog.mozilla.org/blog/2020/02/25/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/

[19] Chromium Blog. (2019, September) Experimenting with same-provider DNS-over-HTTPS upgrade. [Online]. https://blog.chromium.org/2019/09/experimenting-with-same-provider-dns.html

[20] Kenji Baheux. (2020, May) Chromium Blog: A safer and more private browsing experience with Secure DNS. [Online]. https://blog.chromium.org/2020/05/a-safer-

and-more-private-browsing-DoH.html [Accessed May 20, 2020]

[21] Opera Blog. (2019, September) Opera 65.0.3430.0 developer update. [Online]. https://blogs.opera.com/desktop/2019/09/opera-65-0-3430-0-developer-update/

[22] Brave Browser. (2020, April) Release Notes for 1.7.x (Release Channel). [Online]. https://github.com/brave/brave-browser/issues/9109

[23] Rebekah Houser, Zhou Li, Chase Cotton, and Haining Wang, "An Investigation on Information Leakage of DNS over TLS," in *CoNEXT '19: Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, 2019.

[24] Bushart Jonas and Christian Rossow, "Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS," *CoRR*, vol. abs/1907.01317, July 2019.

[25] Marc Juarez, Sandra Siby, Claudia Díaz, Vallina-Rodriguez Narseo, and Carmela Troncoso, "Encrypted DNS --> Privacy? A Traffic Analysis Perspective," in *NDSS Symposium*, 2020.

[26] Anonymous, "The collateral damage of internet censorship by DNS injection," in *ACM SIGCOMM*, 2012.

[27] Stéphane Bortzmeyer. (2015, December) DNS Censorship (DNS Lies) As Seen By RIPE Atlas. [Online]. https://labs.ripe.net/Members/stephane_bortzmeyer/dns-censorship-dns-lies-seen-by-atlas-probes [Accessed May 20, 2020]

[28] Bruce Schneier. (2013, October) The Guardian: Attacking Tor: how the NSA targets users' online anonymity. [Online]. https://www.theguardian.com/world/2013/oct/04/tor-attacks-nsa-users-online-anonymity

[29] Khaled Al-Naami et al., "Adaptive Encrypted Traffic Fingerprinting With Bi-Directional Dependence," in *ACSAC '16: Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016.

[30] Thuy T.T. Nguyen and Greenville Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56 - 76, 2008.

[31] Jun Zhang, Yang Xiang, Wanlei Zhou, and Yu Wang, "Unsupervised traffic classification using flow statistical properties and IP packet payload," *Journal of Computer and System Sciences*, vol. 79, no. 5, pp. 573-585, August 2013.

[32] Höchst Jonas, Lars Baumgärtner, Matthias Hollick, and Bernd Freisleben, "Unsupervised Traffic Flow Classification Using a Neural Autoencoder," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, 2017.

[33] Hardeep Singh, "Performance Analysis of Unsupervised Machine Learning Techniques for Network Traffic Classification," in *2015 Fifth International Conference on Advanced Computing & Communication Technologies*, 2015.

[34] Li Jun, Zhang Shunyi, Lu Yanqing, and Zhang Zailong, "Internet Traffic Classification Using Machine Learning," in *2007 Second International Conference on Communications and Networking in China*, 2007.

[35] Wazen M. Shbair, Thibault Cholez, Jerome Francois, and Isabelle Chrisment, "A multi-level framework to identify HTTPS services," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016.

[36] Jawad Manzoor, Idilio Drago, and Amin Sadre, "How HTTP/2 is changing web traffic and how to detect it," in *2017 Network Traffic Measurement and Analysis Conference (TMA)*, 2017.

[37] Dominik Schatzmann, Wolfgang Mühlbauer, Thrasyvoulos Spyropoulos, and Xenofontas Dimitropoulos, "Digging into HTTPS: flow-based classification of webmail traffic," in *Proceedings of the 10th annual conference on Internet measurement, IMC '10*, 2010.

[38] Liu Chang, Cao Zigang, Li Zhen, and Xiong Gang, "LaFFT: Length-Aware FFT Based Fingerprinting for Encrypted Network Traffic Classification," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018.

[39] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé, "Mobile Encrypted Traffic Classification Using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445-458, February 2019.

[40] Li Rui, Xiao Xi, Ni Shiguang, Haitao Zheng, and Xia Shutao, "Byte Segment Neural Network for Network Traffic Classification," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018.

[41] Liu Chang, He Longtao, Xiong Gang, Cao Zigang, and Li Zhen, "FS-Net: A Flow Sequence Network For Encrypted Traffic Classification," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*.

[42] Jamie Hayes and George Danezis, "k-fingerprinting: a robust scalable website fingerprinting technique," in *SEC'16: Proceedings of the 25th USENIX Conference on Security Symposium*, 2016.

[43] Andriy Pachenko et al., "Website Fingerprinting at Internet Scale," in *Network & Distributed System Security Symposium (NDSS)*, 2016, pp. 1-15.

[44] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright, "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning," in *CCS '18: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[45] Jonas Bushart and Christian Rossow, "Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS," *arXiv e-prints*, pp. arXiv:1907.01317, July 2019.

[46] IETF, "RFC 8467: Padding Policies for Extension Mechanisms for DNS (EDNS(0))," 2018.

[47] Drew Hjelm. (2019, September) SANS Institute - A New Needle and Haystack: Detecting DNS over HTTPS Usage. [Online]. https://www.sans.org/reading-room/whitepapers/dns/paper/39160

[48] IETF, "RFC 7469: Public Key Pinning Extension for HTTP," 2015.

[49] Mozilla. (2020, March) MDN web docs - HTTP logging. [Online]. https://developer.mozilla.org/en-US/docs/Mozilla/Debugging/HTTP_logging [Accessed May 18, 2020]

[50] The Zeek Network Security Monitor. [Online]. https://zeek.org/ [Accessed May 21, 2020]

[51] Salesforce. GitHub: JA3 - A method for profiling SSL/TLS Clients. [Online]. https://github.com/salesforce/ja3 [Accessed May 21, 2020]

[52] Active Countermeasures. GitHub: RITA (Real Intelligence Threat Analytics). [Online]. https://github.com/activecm/rita [Accessed May 21, 2020]

[53] (2010, December) Mozilla Blog: Browsing Sessions. [Online]. https://blog.mozilla.org/metrics/2010/12/22/browsing-sessions/ [Accessed May 21, 2020]

[54] IETF, "RFC 6066: Transport Layer Security (TLS) Extensions: Extension Definitions," 2011.

[55] GitHub: DNS over HTTPS - curl/curl Wiki. [Online]. https://github.com/curl/curl/wiki/DNS-over-HTTPS [Accessed May 21, 20]

[56] IETF, "Issues and Requirements for SNI Encryption in TLS," draft-ietf-tls-sni-encryption-09, 2019.

[57] Wazen M. Shbair, Thibault Cholez, Antoine Goichot, and Isabelle Chrisment, "Efficiently bypassing SNI-based HTTPS filtering," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.

[58] Selenium HQ Browser Automation. [Online]. https://www.selenium.dev/ [Accessed May 22, 2020]

[59] Alexa - Top sites. [Online]. https://www.alexa.com/topsites [Accessed May 22, 2020]

[60] W3C. WebDriver - W3C Working Draft 21 May 2020. [Online]. https://www.w3.org/TR/webdriver/ [Accessed May 22, 2020]

[61] ChromeDriver - WebDriver for Chrome. [Online]. https://chromedriver.chromium.org/ [Accessed May 22, 2020]

[62] GitHub: Releases - mozilla/geckodriver. [Online]. https://github.com/mozilla/geckodriver/releases [Accessed May 22, 2020]

[63] GitHub: Releases - operasoftware/operachromiumdriver. [Online]. https://github.com/operasoftware/operachromiumdriver/releases [Accessed May 22, 2020]

[64] Microsoft Download Center - Download IE WebDriver Tool. [Online]. https://www.microsoft.com/en-us/download/details.aspx?id=44069 [Accessed May 22, 2020]

[65] MozillaWiki: Trusted Recursive Resolver. [Online]. https://wiki.mozilla.org/Trusted_Recursive_Resolver [Accessed May 30, 2020]

[66] The Tcpdump Group. Manpage of TCPDUMP. [Online].

https://www.tcpdump.org/manpages/tcpdump.1.html [Accessed March 31, 2020]

[67] Carlos López, Daniel Morató, Eduardo Magaña, and Mikel Izal, "Effective Analysis of Secure Web Response Time," in *2019 Network Traffic Measurement and Analysis Conference (TMA)*, 2019.

[68] IETF, "RFC 7301: Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension," 2014.

[69] Zheng Yuefeng et al., "A Novel Hybrid Algorithm for Feature Selection Based on Whale Optimization Algorithm," *IEEE Access*, vol. 7, pp. 14908 - 14923, 2018 November.

[70] Rania Saidi, Waad Bouaguel, and Nadia Essoussi, "Hybrid Feature Selection Method Based on the Genetic Algorithm and Pearson Correlation Coefficient," *Machine Learning Paradigms: Theory and Application. Studies in Computational Intelligence*, vol. 801, pp. 3-24, December 2018.

[71] Leo Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5-32, October 2001.

[72] Yoav Freund and Robert E. Schapire, "A Short Introduction to Boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771-780, September 1999.

[73] Bhumika Gupta et al., "Analysis of Various Decision Tree Algorithms for Classification in Data Mining," *International Journal of Computer Applications*, vol. 163, no. 8, April 2017.

[74] Mehrdad Fatourechi et al., "Comparison of Evaluation Metrics in Classification Applications with Imbalanced Datasets," in *2008 Seventh International Conference on Machine Learning and Applications*, 2008.

[75] Naeem Seliya, Taghi M. Khoshgoftaar, and Jason Van Hulse, "A Study on the Relationships of Classifier Performance Metrics," in *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, 2009.

[76] Takaya Saito and Marc Marc Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," *PLoS One*, vol. e0118432, March 2015.

[77] Jason Brownlee. (2020, January) Machine Learning Mastery: ROC Curves and Precision-Recall Curves for Imbalanced Classification. [Online]. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/ [Accessed June 1, 2020]

[78] Zhang Peng and Su Wanhua, "Statistical Inference on Recall, Precision andAverage Precision under Random Selection," in *2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2012)*, 2012.

[79] S. Yadav and S. Shukla, "Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification," in *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, 2016, pp. 78-83.

[80] S. Sehgal, H. Singh, M. Agarwal, V. Bhasker, and Shantanu, "Data analysis using

principal component analysis," in *2014 International Conference on Medical Imaging, m-Health and Emerging Communication Systems (MedCom)*, 2014.

[81] (2019, July) 360 Netlab Blog - An Analysis of Godlua Backdoor. [Online]. https://blog.netlab.360.com/an-analysis-of-godlua-backdoor-en/ [Accessed May 18, 2020]

[82] IETF, "RFC 7816: DNS Query Name Minimisation to Improve Privacy," 2016.

[83] Charlie Osborne. (2019, September) ZDNet - PsiXBot malware upgraded with Google DNS over HTTPS. [Online]. https://www.zdnet.com/article/psixbot-malware-upgraded-with-google-dns-over-https-sexploitation-kit/ [Accessed May 18, 2020]