Transparent and Service-Agnostic Monitoring of Encrypted Web Traffic

Pierre-Olivier Brissaud, Jérôme François, Isabelle Chrisment, Thibault Cholez, Olivier Bettan

Transparent and Service-Agnostic Monitoring of Encrypted Web Traffic

Pierre-Olivier Brissaud, Jérôme François, Isabelle Chrisment, Thibault Cholez and Olivier Bettan

***Abstract*—Nowadays, most of web services are accessed through HTTPS. While preserving user privacy is important, it is also mandatory to monitor and detect specific users' actions, for instance, according to a security policy. This paper presents a solution to monitor HTTP/2 traffic over TLS. It highly differs from HTTP/1.1 over TLS traffic what makes existing monitoring techniques obsolete. Our solution, H2Classifier, aims at detecting if a user performs an action that has been previously defined over a monitored web service, but without using any decryption. It is thus only based on passive traffic analysis and relies on random forest classifier. A challenge is to extract representative values of the loaded content associated to a web page, which is actually customized based on the user action. Extensive evaluations with five top used web services demonstrate the viability of our technique with an accuracy between 94% and 99%.**

***Index Terms*—Encrypted Traffic, HTTP2, HTTPS monitoring, Privacy, TLS, Traffic analysis**

I. Introduction

NOWADAYS, the Web provides access to a lot of online services while users expect privacy protection when accessing them. Indeed, recent public scandals about mass surveillance[1] have raised people awareness and expectation regarding privacy concerns. As a result, the number of web services protected by encryption with HTTPS [1] has hugely increased in the last years, as proven by the percentage of pages loaded through HTTPS (in the USA) for Chrome[2] and Firefox[3], with respectively an increase of 38% and 26% to reach the current proportion of 88% and 84%.

Although preserving and improving user privacy is necessary, monitoring, detecting and mitigating forbidden user actions[4] are also legitimate, such as tracking illegal activities undergone through the Internet for legal reasons. However, defending user privacy while assuring an efficient monitoring of the traffic are quite antagonist tasks. Usual monitoring techniques based on filtering such as port-based rules or deep packet inspection have become obsolete given the generalization of encrypted web applications. To provide accurate traffic monitoring within an enterprise network, decryption proxies are usually deployed but they break user privacy. The objective of a privacy-compliant monitoring of user actions, as proposed in this paper, is not to track every request of every user but rather to raise an alert, or to block user traffic when it does not comply with security rules (*e.g.* forbidden actions) that must be enforced. In addition, fully blocking the access to a remote service is not suitable as a given service can be used in both a legitimate and illegitimate manner.

We thus provide in this paper a solution to detect an inappropriate use of a HTTPS service based on pre-established rules without revealing any other user activity because the encrypted traffic is let untouched, and without entirely denying access to this service.

The solutions presented in this paper are (1) passive as they only collect encrypted HTTP (HTTPS) traffic, and (2) totally transparent as no specific software is supposed to be installed at the user side.

We will first present our previous work [2] dedicated to HTTP/1.1 using TLS. Second, we will propose a new technique for HTTP/2 as it is now widely used by modern browsers and servers. The huge differences between HTTP/1.1 and HTTP/2 on the encrypted traffic have motivated our new solution called H2Classifier. To the best of our knowledge, we are the first to propose a classifier for encrypted HTTP/2 traffic which is able to identify user actions considering a targeted service while being agnostic to the service, *i.e.* our method does not target specific service.

In this paper we present three main contributions about HTTP/2 based on our previous work on HTTP/1.1 [2]:

- An explanation of the root causes of the unsuccessful application of our previous technique with HTTP/2.
- A new method to classify user actions (e.g. searched keywords) when HTTP/2 is used over TLS.
- An application and evaluation with five major services: *Amazon*, *Instagram*, *Google Images*, *Google Maps* and *Google*, with the elaboration of a publicly available dataset containing more than 22,000 distinct requests per service[5].

The paper is organized as follows. Section II presents related work. We strictly define our classification problem in Section III followed by necessary technical background about HTTP and TLS in Section IV. The datasets and metrics for evaluation are described in Section V. Section VI gives an overview of our previous work on HTTP/1.1 traffic classification and Section VII shows its limitation when applied to HTTP/2 traffic. In Section VIII, our approach to classify HTTP/2 traffic is introduced and evaluated. Finally, we discuss

Pierre-Olivier Brissaud is with Universite de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France and Thales, Palaiseau, France {firstname.lastname@inria.fr}. Jérôme François, Isabelle Chrisment and Thibault Cholez are with Universite de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France {firstname.lastname@inria.fr}. Olivier Bettan is with Thales, Palaiseau, France {firstname.lastname@thalesgroup.com}.

[1] https://en.wikipedia.org/wiki/Golden_Shield_Project, https://en.wikipedia.org/wiki/PRISM_(surveillance_program)

[2] https://transparencyreport.google.com/https/overview?hl=en

[3] https://letsencrypt.org/stats/#percent-pageloads, accessed on 06/28/2019

[4] a user action is here a particular request performed on a web service

[5] only 1660 distinct requests for *Instagram*, see Section V.

practical implications of our new solution in Section IX before concluding the paper in Section X.

II. Related Work

Previous studies have shown that privacy is not guaranteed by encryption especially with HTTPS. In 1996, Wagner *et al.* [3] highlighted the vulnerabilities of the SSL protocol against traffic analysis. Then, Cheng *et al.* [4] followed by Sun *et al.* [5] and Hintz [6] have experimented the feasibility of a website recognition attack using resources' size with closed world assumptions[6].

Website fingerprinting over anonymous networks such as TOR[7] or JAP[8] have been extensively studied in literature [7]–[10] following initial work in this area [11]–[13]. All apply different clustering methods and features, mostly based on the packet size distribution, for their fingerprints. Website fingerprinting techniques tend to identify a few pages, from different websites whereas our goal is to monitor pre-defined user activities when they are using an online service, *i.e.* knowing which keyword has been typed, and so, what specific information has been searched.

Some particular methods like Blindbox [14] relies on searchable cryptography. This solution requires the deployment of non-standard encryption protocols and can suffer from scaling issues even if some improvement can be brought [15]. Our solution is transparent for users and can be more easily used at a larger scale.

Analysis of encrypted traffic can be related to different levels as traffic flow classification, service identification or specific application/protocol behavior detection for web or mobile communications. The first level aims to detect the application generating a flow as surveyed by Velan *et al.* [16]. At a higher level of identification, we quote Shbair *et al.* [17] with a framework for the detection of web services over HTTPS. Recently, Montieri *et al.* [18] achieve the classification of the different anonymized networks and detect some types of application traffic. Many research works have analyzed mobile traffic to identify the services installed on a mobile device. AppScanner from Taylor [19] can profile 110 mobile applications with an accuracy between 73% and 96%. In this area, particular activities or behaviors are detected within the flow generated by an application as for example Conti [20] (between 5 and 11 behaviors inside 7 apps), Liu [21] (8 activities for 3 apps) or Saltaformaggio [22] (between 1 and 3 activities for 22 apps for a total of 35 activities). Aceto *et al.* [23], [24] can detect more than 45 apps for both Android and IOS by using different techniques including deep learning. They provide guidelines and an infrastructure for such a solution [25]. In Fu *et al.* [26], encrypted traffic of mobile messaging applications is analyzed to discover usage patterns.

Application-specific encrypted traffic analysis has also been proposed, for Netflix [27] or Skype [28], [29]. These approaches are built on a precise understanding of the encoding techniques for the video or the audio streams and their aftereffects in the encrypted domain. In [30], Coull and Dyer are able to determine the language used in discussions in a text messaging application. They sort the type of messages and, based on the text message length, define a signature for each language. In [31], the authors propose timing attacks for detecting SSH inputs. Finally, IOACTIVE LABS [32] has found the location displayed to the client on Google Maps thanks to a meticulous learning of the size of satellite images and their contiguity. To identify the keywords searched on *Google Images* with HTTP/1.1, we have proposed a similar technique [2], based on images' sizes.

TABLE I: Related work classification

Level detection	description	Topics	
		Web	Mobile
Protocol	Protocol identification	[16]	
Service	Website Fingerprinting (from X to Y monitoring pages over VPN or TOR open or closed world)	[11] [12] [13] [7] [8] [9] [10]	
	Service / Application identification (from 3 to 110 application)	[17] [18]	[19] [20] [21] [22] [26] [23] [24] [25]
User Action	Video / Audio recognition	[27] [28] [29]	
	User Action detection for services (Web: up to 2000 distinct requests, Mobile: up to 11 distinct activities)	[2], this paper	[20] [21] [22]
	Other specific services	[30] [31] [32]	

Table I summarizes the aforementioned related work and positions our work accordingly. A notable difference is that our objective is to detect user actions like filling a search text field when using a monitored service. From the type of identified information, it is similar to a mobile user behavior provider but we focus on HTTPS to track a single user action. Moreover, our scale is larger as we can detect up to 2000 distinct requests.

Finally, to the best of our knowledge, we are the first to propose a method to identify user actions in HTTP/2, knowing that HTTP/1.1 will be obsolete in a near future.

III. Problem statement

Although ensuring the confidentiality of communications is primordial from a privacy point of view, encryption is also powerful to discretely convey malicious or illegal activities. Our technique is thus designed to detect such illegal uses while guaranteeing the privacy of normal users. Indeed, all traces that are not considered as illegal will be grouped into a single class by our technique and so cannot be differentiated afterwards, reminding that decryption never occurs.

A user action is atomic by nature, as for example clicking on a web page element, filling a text-field or toggling a button. For the rest of our experiment, we consider a user action as a request of a specific keyword[9] in a search text-field. These actions have a direct impact on the related pages to be loaded and thus should generate a specific traffic.

In other words, we want to fingerprint dynamic web-pages which generate content that is user action-dependent. We consider a range of popular services like *Amazon*, *Instagram*, *Google Search Engine*, *Google Images* and *Google Maps*.

[6]Closed world is opposed to open world as it only considers a finite sample for both the training and testing. The open world assumption involves a finite dataset to monitor and unlimited possibilities (unknown classes/labels) for the testing phase.

[7]https://www.torproject.org/

[8]https://anonymous-proxy-servers.net/index.html

[9]In the rest of the paper, keyword refers to the fully qualified user search which is actually a single or a set of keywords

Thus, when a user requests a specific keyword by filling a search field on a given web service, our technique aims at knowing if the traffic generated by this request and the associated response matches a signature learned from previous traces[10] produced by a monitored keyword, and possibly to determine this specific keyword.

Assuming L a dataset for the learning purpose and M the monitored keyword set, every single element $l \in L$ is a flow corresponding to a single keyword $w \in M$ and defined as a sequence of packets p_i:

$$\forall l \in L, \exists w \in M : l = \langle w, \{p_1, \ldots, p_i, \ldots, p_n\} \rangle \quad (1)$$

Based on that, our model is built as a function f that takes a new sequence of packets p as input and deduces whether it has been generated by a monitored keyword w. If not, the function returns u (*i.e.* unknown keyword):

$$\forall p = \{p_1, \ldots, p_i, \ldots, p_n\},$$
$$f(p) = \begin{cases} w \in M \\ u \end{cases} \quad (2)$$

The problem resides in defining f from the learning samples, *i.e.* all $l \in L$ and the associated keyword $w \in M$.

IV. Background

Since HTTPS is the combination of HTTP version 1.1 or 2 protected with TLS, this section gives an overview of TLS, HTTP/1.1 and HTTP/2 but limited to the relevant technical details for this paper.

A. Transport Layer Security (TLS)

TLS (Transport Layer Security) is carried by a Transport protocol (*e.g.* TCP) and is split in two parts: the TLS record protocol and the TLS handshake protocol. The handshake is not discriminative in regards of loaded content for a particular service and is so out-of-interest in our context.

There are four types of records but we only consider the application data records since the others manage only the connection itself and are thus not specifically dependent of the content our technique aims at identifying. The data encapsulation process for TLS 1.2 [33] is illustrated in Figure 1. The content to be sent is fragmented into blocks of 2^{14} bytes. Then, these blocks, called fragments, are compressed (optional, but the resulting size must be under $2^{14} + 1024$ bytes), authenticated (addition of the HMAC) and finally encrypted. The final resulting size of the encrypted fragment should be smaller than $2^{14} + 2048$ bytes. However, the compression must be disabled to be compliant with the HTTP/2 standard, as explained in Section IV-C .

[10]In the rest of the paper, a trace refers to a packet-level capture between the client and the web-server, when requesting a keyword until the full page with all objects is loaded.

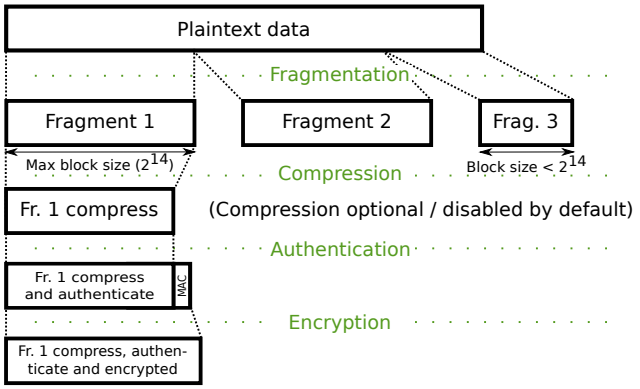


Fig. 1: TLS record layer

B. HTTP/1.1

HTTP/1.1 is the common version of HTTP (Hypertext Transfer Protocol) from 1997 (validation of the first HTTP/1.1 RFC) to 2015 (adoption of the RFC 7541 for HTTP/2). The whole protocol is in plain text ASCII (*e.g.* each request begins with one of the nine HTTP methods like GET, POST or DELETE).

The HTTP exchange pattern is simple: over a TCP connection the client sends one request and waits until the full response arrives before sending a new request. Figure 3(a) shows an example of this pattern. To improve performance, the pipelining concept was introduced as an extension of HTTP/1.1 to decrease the loading time of HTML pages by allowing the client to send new requests proactively.

However, it is limited by the head-of-line blocking effect[11] as the requests are processed in FIFO, like in Figure 3(b). As a result, due to this limitation and some compatibility issues, the pipelining feature has been abandoned [34], [35] and HTTP/1.1 clients usually open multiple connections in parallel (for example to load all the different objects of a page).

Regarding HTTP/1 without pipelining, it is possible to reconstruct the encrypted size of the HTTP requests and responses. In that case, assuming the TCP connection embedding HTTP traffic, all the response packets between two requests are related to the same HTTP object. The encrypted size of this object can be thus extracted. All retrieved encrypted sizes constitute the features used in our classification engine for HTTP/1 as explained in Section VI.

C. HTTP/2

HTTP/2 promotes binary headers instead of clear-text ones. The protocol header structure is described in Figure 2. The header introduces two mechanisms: the framing layer (with the header-field Frame Type) and the streams (with the Stream Identifier). First, the framing layer is binary and defines 10 structures for different types of data such as the message payload, HTTP headers or the priority system. The frame data is then placed after the headers in the Frame Payload. Second, the headers contain a stream identifier that allows the multiplexing feature. Indeed, this identifier associates a

[11]If the current processed request is postponed on the server, all remaining connections are blocked until all responses related to the current request are processed.

frame to a stream, which can be seen as an independent connection between two hosts. Through the streams, a single HTTP/2 connection can define multiple *virtual* connections. Each request is done in a new stream and all responses are sent with the stream id related to the request. The streams are managed with a priority system in order to inform the other host about the most important streams.

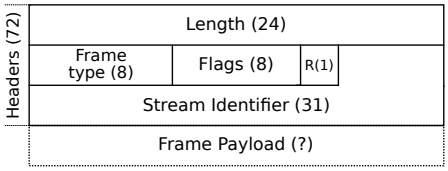


Fig. 2: HTTP/2 binary header structure (size in bits) encrypted over TLS 1.2

HTTP/2 was designed to be used with or without encryption and is identified with h2 (HTTP2 over TLS) or h2c (HTTP2 over TCP) respectively. However, modern web browsers only implement h2[12]. For the rest of the paper we thus only refer to h2. h2 must use TLS 1.2 or higher and the compression of TLS must be disabled [36]. During the handshake, TLS should be used with the Application-Layer Protocol Negotiation (ALPN) [37] extension to successfully begin a h2 session. This extension allows an HTTP/2 connection to be set in only one exchange of enriched hello messages.

A major difference with HTTP/1.1 is the HTTP header compression which has a dedicated algorithm: HPACK [38]. This algorithm is based on static and dynamic dictionaries and is resilient to the CRIME attack[13], which could give access to authentication cookies in HTTPS by exploiting data compression. Finally, the Server Push function allows a server to anticipate a client need. Before a client claims some content, the server can predict the future need and send data before the client request is even received (*e.g.* CSS file after HTML page request). Those two last features are core features for HTTP/2 but they have no direct impact on our classification work contrary to pipelining and multiplexing features.

The multiplexing provided by HTTP/2 is prone to break the FIFO processing pattern for requests. There is no more blocking situation because all requests can be processed, and answered back, in any order. Each HTTP request and the related response are sent in a separate stream within the same connection, as shown in Figure 3(c).

As a result, the request and response for an individual object cannot be distinguished by monitoring the encrypted traffic because the packets of a single given HTTP object are no longer sent in an continuous sequence. This makes our previous approach [2] ineffective (reported in Section VI-C).

V. Datasets and Metrics

A. Data collection overview

We have developed our own crawler to collect traces for both the training and testing stages of our classification model. It opens a page at a specific URL on the web-browser and collects all the packets from the generated traffic in a pcap file, from which all input and output TLS records sizes are extracted. The crawler is implemented in Python and use the Selenium[14] library coupled with geckodriver[15] in order to control the Firefox web browser; tcpdump[16] is used for the packet capture.

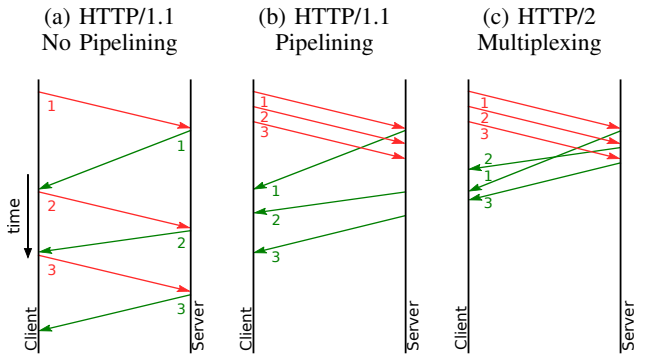


Fig. 3: Compare HTTP request and response with different configuration

We assume that only one request, for instance related to a given searched keyword, is captured at a time on a given service. Although one could consider that as an ideal and unrealistic scenario, distinguishing the use of a particular service is doable. Indeed, HTTPS traffic can be easily detected based on port number and headers [16]. Then to reconstruct a flow, Groleat *et al.* [39] or Xiong *et al.* [40] propose TCP reassembling method for HTTPS. Finally, it is possible to extract a pre-defined service based on IP addresses or domain names. However, it is known to have a limited applicability and better techniques exist such as using the TLS SNI field like in [17] or those already described for website fingerprinting in Section II.

A first dataset contains exclusively HTTP/1.1 (over TLS 1.2) traces of the Google Images search engine and was used in our prior work [2]. We remind that the pipelining option of HTTP/1.1 is not used. Our second dataset includes HTTP/2 (over TLS 1.2) traces for five services: Amazon, Instagram, Google search engine, Google Images and Google Maps.

To support reproducible research, a sample and a description of the datasets are available at **http://betternet.lhs.inria.fr/datasets/googleimg/** and at **http://betternet.lhs.inria.fr/datasets/h2classifier/** for HTTP/1.1 and HTTP/2 datasets respectively. Full access can be provided on demand.

B. HTTP/1 Dataset

For HTTP/1.1, the crawler was configured as follows:

- Client web-browser: Firefox 54.0.1 (64-bit) on Debian 8 (1 dedicated host),
- Browser settings: full screen display with a resolution of 1920×1080, cache disabled and HTTP/2 disabled to force HTTP/1.1 (TLS version 1.2),
- English version of *Google Images* (google.com[17]).

[12]https://bugzilla.mozilla.org/show_bug.cgi?id=1418832, https://caniuse.com/#feat=http2, accessed on 06/28/19

[13]http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2012-4929

[14]https://www.seleniumhq.org/

[15]https://github.com/mozilla/geckodriver

[16]https://www.tcpdump.org/

[17]https://www.google.com/ncr for disabling regional redirection

TABLE II: HTTP/1.1 datasets

ID	Origin of keywords	Number of keywords	Captures per keyword	Average packets number by trace	Average duration by trace
$data_1$	Dictionary (en)	10,500	5	1155 pkt/trace	2.94 s/trace
$data_2$	Wikipedia (en) titles pages	105,000	1		

For a realistic evaluation, *i.e.* in the case of an open world situation, the objective is to determine whether a new trace is significantly representative of a monitored keyword (from M) or not. Hence, some of the testing traces correspond to the monitored keywords ($\in M$), while others represent unknown user actions ($\notin M$).

Table II describes the composition of two sets of traces. A first set of experiments will be exclusively based on a restricted dataset $data_1$ of 10,500 keywords from the English Oxford dictionary. It will allow us to assess our technique in various conditions. A second dataset, made from a larger scale experiment, extends the first one with 105,000 other words/expressions from Wikipedia pages titles[18] composing $data_2$. Five traces of each keyword are automatically captured for $data_1$, four are reserved for training, *i.e.* to build meaningful signatures, and one for validation (testing). The $data_2$ traces will be used to verify whether our signatures are robust against new unknown keywords ($\notin M$), *i.e.* to consider a large-scale open world case. Thus, it only contains a single trace per keyword. HTTP/1.1 traffic has been captured during June and July 2017.

C. HTTP/2 Dataset

Below is the configuration of the crawler for the HTTP/2 dataset:

- Client web-browser: Firefox v.63.0 (64-bit) on Ubuntu 16.04 (different virtual machine with 2 CPU and 4GB of memory),
- Browser settings: full screen display with a resolution of 1920×1080, cache disabled.
- English version of the different services listed below

We consider the following widely used web services:

- *Amazon* (https://www.amazon.com): the web market of the largest retailer in the world.
- *Instagram* (https://www.instagram.com/): a photo and video-sharing social network.
- *Google Search* (https://www.google.com/): the most used web search engine.
- *Google Images* (https://images.google.com): a web search engine for images.
- *Google Maps* (https://maps.google.com): an advanced web mapping service.

We built two sets of traces as shown in Table III. *data_h2_2000* is composed of 12 traces for each of the 2000 monitored keywords (depending on the service) while a single trace is captured for each of the 20,000 legitimate keywords to constitute test traces for the open world scenario. *data_h2_500* is built from less keywords (500) but with more traces for each keyword (60 traces), these keywords are a subset of the 2000 monitored keywords. It is a complementary dataset to verify that our technique does not suffer from over-fitting in Section VIII-B1. The traffic was captured on September 17-25 2018 and March 22-28 2019 for *data_h2_2000* and *data_h2_500* respectively.

The keywords used for Google and Google Images are the ones used to create the $data_2$ HTTP/1 dataset from Wikipedia title pages (en). We searched the different keywords on both services independently. For Google Maps, a list of French cities[19] and of some big cities in the world[20] have been considered. The city names were selected in a random order and used in the search field to load the map of the city. Our Amazon keyword list is built from the dataset of J. McAuley [41] and specially the metadata file. This file contains a list of exact product names. Because we want to use the main search text field as a normal user, we mimic her behavior by selecting the most frequently 3-grams of words in the mentioned list. For Instagram, profile names were used. Some of them can be extracted from the Instagram profile section[21]. However, many of them cannot be publicly accessed and have been discarded or deleted during the capture. Thus, the Instagram dataset is only based on 1662 profiles and building an extensive dataset for the open world scenario was not possible in that particular case. For the sake of clarity, Instagram profiles will be also referred as keywords in the rest of the paper.

D. Classification metrics

In addition to usual metrics as TPs (True Positives), FPs (False Positives) and FNs (False Negatives), we introduce the Wrong Classification (WC) as illustrated in Figure 4.

A Wrong Classification happens when a trace related to the monitored keyword list is detected as such but the keyword predicted by our system is wrong. We make the distinction between TP and WC because our technique can be tested with two different granularities: to detect a monitored keyword has been searched without distinguishing that particular keyword ($TP + WC$) or to identify exactly the keyword that has been used (only TP). In the testing dataset, the number of traces related to a monitored or unknown keyword is denoted as $|m|$ and $|u|$ respectively. The following relative metrics are thus computed:

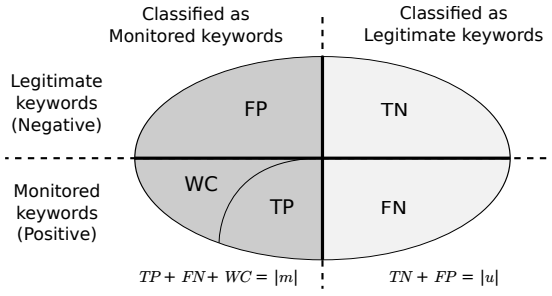


Fig. 4: Metric definition

TPR (True Positive Rate) The probability that a trace from a monitored keyword is classified as the correct monitored keyword.

[18] dumps.wikimedia.org/enwiki/latest/enwiki-latest-all-titles-in-ns0.gz

[19] http://www.nosdonnees.fr/wiki/images/b/b5/EUCircos_Regions_departements_circonscriptions_communes_gps.csv.gz

[20] https://datahub.io/core/world-cities, accessed on June 28 2019

[21] https://www.instagram.com/directory/profiles/

TABLE III: HTTP/2 datasets

Service	Origin	*data_h2_2000*		*data_h2_500*	
		Number of monitored / legitimage keywords	Captures per monitored / legitimate keyword	Number of monitored keywords	Captures per monitored keyword
Amazon	J. McAuley [41]	2000 / 20,000	12 / 1	500	60
Google Google Images	Wikipedia titles pages ($data_2$)				
Google Maps	French or international cities				
Instagram	Social media profiles	1662 / n.a.	12 / n.a.	n.a.	n.a.

FPR (False Positive Rate) The probability that a trace from a non-monitored keyword is incorrectly classified as a monitored keyword.

FNR (False Negative Rate) The probability that a trace from a monitored keyword is incorrectly classified as a non-monitored keyword.

WCR (Wrong Classification Rate) The probability that a trace related to a monitored keyword is classified as another monitored keyword.

Acc (Accuracy) The ratio of traces that have been correctly classified independently of their nature (monitored or unknown keyword).

VI. H1Classifier: classification of HTTP/1 flow

A. Rationale

In most web services that deliver a set of images as a result, like *Google Images* [2], several thumbnails are returned for a keyword. The number of thumbnails is not constant for all requests, but we can reasonably assume several dozens for each. Also, according to our experiment on Google Images, the results are indifferent to user profile but might be affected by some option like safe search (see IX-B). Each thumbnail is generated from a single image. Due to image compression techniques with formats like jpeg, the size (in bytes) of the images highly varies and is independent of the rendered size (in pixels). As a result, the encrypted size of the thumbnails differs from one image to another. A keyword is thus associated with multiple sizes of encrypted images that span over thousands of bytes. Having two keywords sharing the same set of image sizes is highly improbable as it is a high combinatorial problem.

Assuming n the average number of returned thumbnails, s the number of distinct sizes, s^n is the number of possible sets of thumbnail sizes leading to few collisions among two distinct keywords. Given a specific keyword and a uniform distribution of image sizes, a second keyword has a probability of $\frac{1}{s^n}$ to be associated with the same image sizes and thus to generate an identification error. This problem is equivalent to the birthday problem or paradox [42]. Thus, by considering a list of k keywords, the probability of a collision between at least two keywords is:

$$P(k) = 1 - \frac{s^n\ !}{(s^n)^k \times (s^n - k)!} \quad \text{or } P(k) \approx 1 - e^{-\frac{k(k-1)}{2 \times s^n}} \tag{3}$$

While this statement needs to be moderated because the uniformity of the size distribution is not guaranteed, our practical experiments have proven the viability of using encrypted thumbnails' sizes to build a keyword signature.

Using *Google Images* and 115,500 traces (cf. Table II), each of them generated from a different keyword, we have evaluated the average number of thumbnails and the distribution of their sizes (in bytes). First, around 50 thumbnails per keyword are loaded (the exact average is 50.58 with a standard deviation of 12.72). Second, the cumulative distribution in Figure 5 (computed from more than 5,800,000 thumbnails from Google Images) highlights that 60% of sizes are almost uniformly distributed (linear portion of the curve). They count for almost $s = 6000$ possible sizes. In the worst case, the probability of having at least one collision with 115,500 keywords is 8.25×10^{-180} using Equation (3). This ultra low probability demonstrates the validity of using the sizes of thumbnails as a signature for a given keyword. Assuming a whole English dictionary of $k = 230,000$ words[22] and only 10 thumbnails loaded ($n = 10$), this probability has a magnitude order of 10^{-28}.

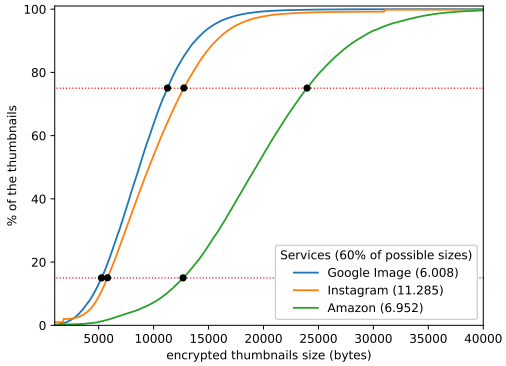


Fig. 5: Cumulative number of thumbnails depending on the size

Regarding other web services like Amazon or Instagram, we observe a similar distribution as shown in the Figure 5. 11,000 and 7,000 possible sizes are linearly distributed for Instagram and Amazon respectively. This proves the validity of our method on other services since the Google Images service is the worst case by having the smallest span in thumbnail sizes, which leads to a higher probability of collisions by nature.

B. Refined Model

Like the aforementioned image search services, the page content of any web-based application depends on user actions and requests. Therefore, the signature of a service use can be expressed through the observable characteristics of the HTTP objects despite the encryption, especially the different sizes. Indeed, the encrypted size of an object is highly related to its original size. Thus, we have refined our model based on

[22]https://en.wikipedia.org/wiki/List_of_dictionaries_by_number_of_words, accessed on 06/28/19

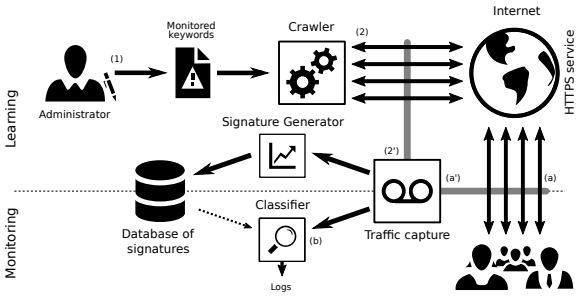


Fig. 6: Global architecture to track HTTPS (HTTP/1.1) application use

Equation (2), with s_i the size of each encrypted object from a trace, as follows:

$$\begin{aligned} \forall l \in L, \exists w \in M : l = \langle w, \{s_1, \ldots, s_i, \ldots, s_n\}\rangle \\ \forall s = \{s_1, \ldots, s_i, \ldots, s_n\}, \\ f(s) = \begin{cases} w \in M \\ u \end{cases} \end{aligned} \tag{4}$$

In online shopping websites, products are usually represented by thumbnails while social networks also use many images to illustrate text content. Hence, images are often present in web pages and we thus select images as significant objects to build signatures for H1Classifier.

C. Methodology

1) Overview: As highlighted in Figure 6, a learning phase is required so that the administrator can monitor user traffic. A database of signatures related to the keywords of the monitored list M is built by the learning module. The list M is provided by the *administrator* (1), the crawler then requests each keyword, a given determined number of times (5 times per monitored keyword for $data_1$), on a service (an image search engine in our case) (2) while the traffic is captured (2'). A profile from the traces related to a unique keyword (details in Section VI-C2) is then derived by the signature generator.

When a user accesses the search engine (a), the traffic (a') is captured and the part related to the loading of web objects is extracted. Data are then sent to the classifier (b) which is in charge of determining if a match exists with a signature from the database built during the learning stage (details in Section VI-C3). Then, the results of the classification engine can be used to raise alerts to the administrator in case of policy violation.

2) Signature generation: As discussed in Section VI-A the thumbnails' sizes are representative of a searched keyword. However, the HTTP answer containing the thumbnails also includes the HTTP response header whose size can slightly vary. Hence, relying on exact sizes is irrelevant. Thus, the designed method, H1Classifier, has to catch the variable sizes of the thumbnails and to automatically learn this variation when multiple samples are collected.

In this paper, we leverage the Kernel Density Estimation (KDE) technique [43]. Assuming a distribution for the thumbnail sizes of a keyword, KDE estimates the density function. Unlike Gaussian mixture models, KDE is non-parametric in the sense that it does not assume any specific type of the underlying probability function. Hence, the derived density function represents a signature of a keyword and any set of thumbnail sizes can be tested against to verify whether it matches. Let $X = (x_1, ..., x_n)$ be a random distribution, K a kernel function and h the bandwidth parameter of the kernel, the density function is estimated as follows:

$$\begin{cases} \mathbb{R} & \to [0,1] \\ y & \mapsto \frac{1}{nh}\sum_{i=1}^{n} K(\frac{y-x_i}{h}) \end{cases} \tag{5}$$

The bandwidth parameter sets the span of the smoothing of the density function around the provided sample points.

The smoothing shape is determined by the kernel function, which must fulfil some requirements:

$$\begin{aligned} \text{Definition} \quad & K : \mathbb{R} \to \mathbb{R}_+ \\ \text{Normalization} \quad & \int_{-\infty}^{+\infty} K(y)\,\mathrm{d}y = 1 \\ \text{Symmetry} \quad & \forall y \in \mathbb{R} : K(-y) = K(y) \end{aligned}$$

In our context, a distribution is composed of all the n thumbnail sizes s_i for a searched keyword k. As we have no prior assumption on data, we use an uniform kernel function defined in Equation (6):

$$K(y) = \frac{1}{2} \text{ if } \mid y \mid \leq 1 \text{ else } 0 \tag{6}$$

Assuming Equations (5) and (6), the signature function, denoted σ_w, is thus defined regarding the distribution of encrypted objects' sizes (*i.e.* the encrypted thumbnails' sizes: s_i):

$$\sigma_w(y) = \frac{1}{nh}\sum_{i=1}^{n}(\frac{1}{2} \text{ if } \mid y - s_i \mid \leq h \text{ else } 0) \tag{7}$$

Our database of signatures D is thus composed of all estimated density functions: $D = \{\sigma_w, w \in M\}$

3) Classification engine: According to Equation (4), H1Classifier processes a list of object sizes and should return a searched keyword w from the list M or u (for "unknown") in case the keyword was not in the monitored keyword list. The whole process is split into three parts: the extraction of thumbnail sizes, the signature scoring and the result filtering.

Considering the full packet capture of a single requested keyword, extracting the HTTP object size requires first to delimit the HTTP object in the encrypted payload. As no pipelining is available by default on modern web browser with HTTP/1.1 version[23], we know that, between two client requests sharing the same TCP connection, there are only packets corresponding to the HTTP response (header + payload) of the server [44], hence referring to a single HTTP object.

Therefore, by analysing the TCP and TLS headers, a list of sizes for all the encrypted HTTP requests and the corresponding responses can be determined.

By building the distribution of the request sizes we can empirically determine the request sizes related to thumbnails as they are the most frequent ones and so filter the returned objects' sizes to keep only the ones related to thumbnails.

[23]https://developer.mozilla.org/fr/docs/Web/HTTP/Connection_management_in_HTTP_1.x, accessed on 06/28/19

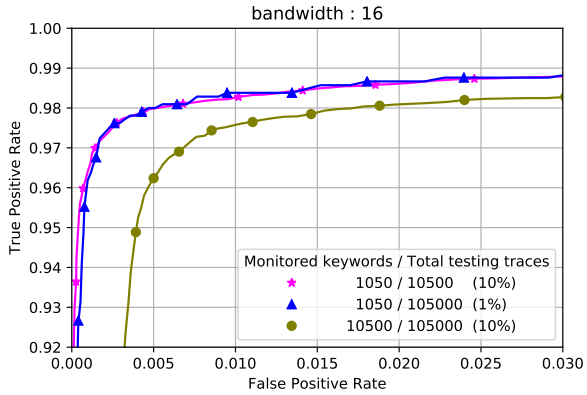


Fig. 7: ROC curve with different datasets assuming α varies

Actually, such an analysis has to be done for each monitored service. The classifier has thus as input a list of thumbnail sizes [24] for the trace subsequent to a queried keyword. This list is denoted as $s = \{s_1, \ldots, s_i, \ldots, s_n\}$.

The scoring consists in computing a single score for each signature (density function) of the database D. For each $\sigma_w \in D$, the score is calculated by evaluating the density of each size s_i with the function σ_w and summing over all results:

$$sc_w(s) = \sum_{i=1}^{n} \sigma_w(s_i) \tag{8}$$

A naive classifier would select the keyword associated with the maximum score. However, the classifier may decide that a keyword is unknown as highlighted by u in the classification function in Equation (4). The main issue is that there is no signature for u and so a maximum score can not be calculated for an unknown keyword. H1Classifier avoids as much as possible false positives and prefers to raise an unknown value if the confidence in the score is too low. Assuming the full list of calculated scores L_s, for a given capture, s, of image sizes, the maximum score is considered as relevant only if it reaches a threshold calculated from the mean, $mean(L_s)$, and the standard deviation, $std(L_s)$:

$$\begin{aligned} w_{max} &= \arg\max_{w \in M} sc_w(s) \\ f(s) &= \begin{cases} w_{max} \text{ if } sc_w(s) > mean(L_s) + \alpha \times std(L_s) \\ u \ otherwise \end{cases} \end{aligned} \tag{9}$$

The filtering parameter α multiplied by the standard deviation defines the minimum deviation from the mean.

D. Large-scale Open World experiment HTTP/1.1

This section summarizes the results obtained by our HTTP/1.1 identification framework [2]. This prior work notably includes a full evaluation of parameter settings. In this paper, we consider the best bandwidth parameter $h = 16$ (determined in [2]).

Figure 7 reports the results depending on the number of monitored keywords and the ratio of the latter over the total number of keywords used for testing. The ROC curve is calculated varying the filtering threshold α that determines if a trace belongs to a monitored keyword or not based on the obtained classification score.

With 10% of monitored keywords (1050 from $data_1$ dataset), we observe that it is possible to achieve $TPR = 0.968$ and $FPR = 0.001$. By considering more non-monitored keywords from the $data_2$ dataset and decreasing the ratio of monitored keywords to 1%, no major difference can be observed (curve 1050/105,000). Hence, our classification approach to profile user actions is robust against the noise induced by the traces of non-monitored keywords, so the signatures remain valid even when many legitimate requests occur (using allowed keywords). An update of the signature database is actually necessary when the list M is updated (new keywords added) or for trending keywords which results may change over time.

However, when the amount of monitored keywords increases, in our case with 10,500 keywords (curve 10,500/105,000 that represents both the full $data_1$ and $data_2$), a degradation of the classification performance is observed due to more collisions between signatures. Because the similarities between signatures from different keywords is very low, the degradation is highly limited. Figure 7 shows that the number of TPs decreases by around 2%. Meantime, the number of FPs slightly increase by 0.5%. Actually, as reported in Table IV, the maximum accuracy is 0.9918 with $TPR = 0.966$ and $FPR = 0.0056$.

VII. Classifying HTTP/2 flow with H1Classifier

This section presents the limitation of the previous method when applied to HTTP/2 traffic. We will first present features for the encrypted object sizes. Then, H1Classifier will be tested and discussed with real HTTP/2 traffic.

A. Finding equivalent features

The first problem is to find equivalent features to the image sizes. Applying the HTTP/1 classifier with all the TLS record sizes is impossible because the loading of individual objects cannot be distinguished anymore as noticed in Section IV-C. In order to adapt our prior HTTP/1.1 method to HTTP/2, new equivalent features must be thus defined.

The TLS fragmentation still leaks information about the size of objects. Considering a large object (*e.g.* an image) in the TLS input buffer (at the server side), it is first split into $n + 1$ fragments, with the first n ones having the same size MAX_RECORD_SIZE (MAX_RECORD_SIZE is lower or equal to 2^{14} and depends on the web server configuration). However, the last block contains the remaining bytes with a very low probability to be equal to MAX_RECORD_SIZE.

As no compression is performed at the TLS layer, the size of this last block depends directly on the content in the input buffer. Therefore, records' sizes with low frequency can be considered as relevant, *i.e.* representing objects sizes, by their remaining bytes, that are quite distinct from others. We remind the reader here that all objects can be mixed and so received successive MAX_RECORD_SIZE bytes blocks can be from different objects.

[24] for the rest of the paper we note the thumbnails' or images' sizes refer to the encrypted HTTP responses' sizes related to the thumbnails

TABLE IV: Results: KDE-based method, HTTP/1.1, large dataset ($data_1$ & $data_2$ with 10,500 monitored / 105,000 unknown)

α \ h	1	4	16	50	1	4	16	50	1	4	16	50	1	4	16	50
	TPR				FPR				Accuracy				WCR			
3.5	0.820	0.979	0.986	0.927	0.1042	0.0908	0.0811	0.0739	0.8889	0.9207	0.9250	0.9262	0.017	0.001	0.001	0.001
3.9	0.814	0.973	0.980	0.790	0.0219	0.0201	0.0162	0.013	0.9631	0.9788	0.9834	0.9691	0.004	0	0	0
4.1	0.805	0.965	**0.966**	0.557	0.0052	0.0069	**0.0056**	0.0026	0.9775	0.9884	**0.9918**	0.9574	0.001	0	**0**	0
4.25	0.780	0.932	0.871	0.170	0.0009	0.0038	0.0026	0.0003	0.9792	0.9857	0.9858	0.9242	0			
4.35	0.448	0.253	0.009	0	0	0.0004	0	0	0.9498	0.8762	0.9099	0.9091	0			

Based on Equation (1), we can reconstruct the sequence of TLS records' sizes and only keep the relevant values, *i.e.* sizes with a frequency lower than a threshold $frequency\_threshold$. It will also exclude very common sizes due to non discriminative objects that are loaded in many pages.

The threshold is defined as the product of the average frequency (count on our full dataset) and a factor, β, to be empirically determined.

$$frequency\_threshold = \beta \times mean(all\_sizes\_frequency) \tag{10}$$

B. Closed World experiment HTTP/2

To evaluate KDE-based signatures when applied to HTTP/2, we assume a best case with a closed world assumption as all keywords are known and monitored. As expected, the classification is very less effective than for HTTP/1.1 and would be even worse in case of an open world evaluation. This last scenario is thus not considered when using the KDE approach for HTTP/2 (but will be evaluated with the H2Classifier in Section VIII-B). Moreover, for a fair comparison with HTTP/1.1, only 5 random traces are kept for each keyword composing the HTTP/2 dataset (see Section V-C).

In the following evaluation, we are particularly interested in the impact of the parameter tuning, *i.e.* h the bandwidth parameter of KDE, and β the filtering frequency used to discard non relevant TLS record sizes. Table V summarizes the results obtained with the best values found for the parameters based on 5 independent evaluations. As highlighted, the best value is $h = 0.5$. Hence, the variability of encrypted sizes for the same content is lower for HTTP/2. However, the optimal value of β depends on the service showing also the necessity to optimize the parameters for each service to be monitored.

As shown in table V, results are only acceptable for Google Images and Maps.

C. Limitations by design

We describe here the limitations of the HTTP/2 signature model leading to the poor results obtained in the previous section. First, the TLS input buffer may not only contain the data of a single HTTP object. It can contains other HTTP data frames or other types of frames too. When fragmentation happens, different frames (and so objects) can also be merged together in the same record. Figure 8 shows an example

TABLE V: HTTP/2 KDE results closed world (mean 5 evaluation)

Source	Amazon	Instagram	Google	Google Images	Google Maps
β, h	1, 0.5	0.5, 0.5	0.1, 0.5	1, 0.5	1, 0.5
Accuracy	0.8475	0.5865	0.259	0.9914	0.924
Accuracy std	0.0076	0.0052	0.0071	0.0013	0.0137

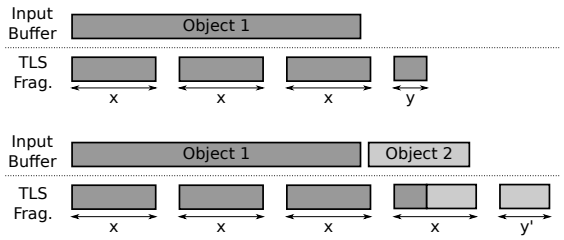


Fig. 8: Fragmentation with different numbers of object in the input buffer

when the buffer contains one or two objects. As a result, the remaining sizes used as features may be representative of the remaining bytes of an aggregation of several objects. The sizes of objects are collected with a coarser granularity, that logically degrades the performance of the classification. In addition, there is no guarantee that the same objects are always merged together in a TLS record. That is why more traces would be necessary to collect the traces representative of different object aggregations.

Second, the MAX_RECORD_SIZE value varies from one web service to another and over the connection time as explained in [45]. Actually, some services may limit the MAX_RECORD_SIZE value during connection setup and increase it later over time to be aligned with the TCP slow start. MAX_RECORD_SIZE values impact on two different parameters. On the one hand, it defines the range of the possible sizes and so the number of collisions when building the KDE signatures as explained in Section VI-A. On the other hand, the larger the MAX_RECORD_SIZE value, the higher the probability to have multiple merged objects in the TLS buffer.

To summarize, the main problems are the aggregation of objects due to multiplexing and a lower span in size values, both leading to more collisions when building signatures with KDE.

D. Discriminating factor of the signatures

To demonstrate our previous observation in practice, we evaluated the discriminating factor of the signatures, *i.e.* quantify how much a single signature is representative of a single keyword. A pair-wise dissimilarity between the signatures is so calculated. As the signatures are functions, the dissimilarity of two signatures can be evaluated by calculating the area between the curves of these two functions:

$$\phi(w_1, w_2) = \int_0^\infty \mid \sigma_{w_1}(x) - \sigma_{w_2}(x) \mid \mathrm{d}x \tag{11}$$

where σ_{w_1} and σ_{w_2} are the signature functions of the keywords w_1 and w_2 respectively.

The dissimilarity score ϕ is between 0 and 2 because $\forall w : \int_0^\infty \sigma_w(x)\,\mathrm{d}x \leq 1$ as σ_w is a density function as described in

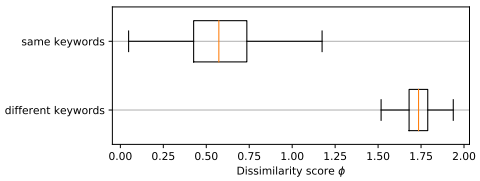


Fig. 9: Dissimilarity of the signature function for HTTP/1.1

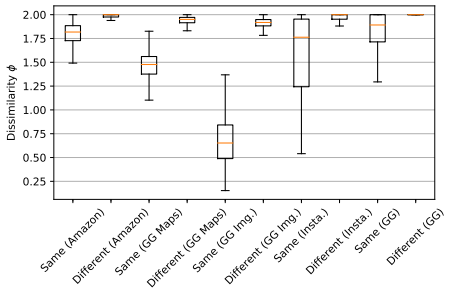


Fig. 10: Dissimilarity of the signature function HTTP/2

Section VI-C2. The higher the score, the lower the similarity between the signatures.

Due to the intensive pair-wise comparison, 1000 keywords were randomly selected before applying Equation (11). We compare the dissimilarity computed between two signatures from the same keyword (built from different traces) and between two signatures from different keywords. Figure 9 reports the result for HTTP/1.1. As shown, signatures are highly discriminative among two different keywords because there is no overlap between the dissimilarity scores of the signatures from the same and from different keywords, even for the extreme scores. In addition, the dissimilarity for a same keyword is a bit more scattered. This is because the object responses do not have always the exact same size in the encrypted domain as explained in Section VI-C2, even for the exactly same request. Using multiple traces allows us to smooth the variations in our signature generation thanks to KDE.

For the HTTP/2 in Figure 10, some services, such as Google Images and even Google Maps, exhibit disjointed dissimilarity scores when assuming the same or different keywords while other services show a high overlapped in computed scores. This is fully aligned with the previous results given in Section VII-B.

E. A need for another classification technique

Our KDE-based technique introduced with our previous work for HTTP/1.1 [2] is not so appropriate for HTTP/2 for the following reasons. First, the bandwidth h set to 0.5 shows that TLS record sizes do not vary (for a same keyword) as much as for HTTP/1.1. One reason of using KDE, was indeed to smooth these variations. Second, a new parameter (β) has been introduced and depends on the monitored service, reducing so the practicability of the approach. Third, because the HTTP/2 multiplexing avoids to distinguish individual requests and loaded objects, we cannot reconstruct the entire size of these objects. More values are thus used to build the KDE signatures reducing its computational efficiency for both learning and testing (as the resulting calculation in Equation 5 always depends on all original samples used for learning). Finally, the TLS records' sizes we rely on have been filtered a priori despite they could contain relevant information for classification purposes. However, taking all of them, including high frequency sizes, leads to degrade results as well.

VIII. H2Classifier: HTTP/2 classification refinement

Due to the aforementioned limitations, the model was extended by including additional features. Because the KDE method does not fit with multivariate models, we rely on a multi-class classification machine learning algorithm for HTTP/2. Although there are multiple candidates (neural networks, naive bayes, support vector machines, decision trees, *etc.*), we leveraged random forest as it has been proved to provide good results in many traffic analysis problems even in earlier works on HTTPS classification [17]. Furthermore, as demonstrated in the following section, results are very satisfactory.

The random forest learning technique [46] builds an ensemble (call 'forest') of decision trees, each one based on a random part of the training data. For classifying a new data sample, each tree gives the resulting label and the most selected label (among all decision trees) will be returned by the algorithm.

As five services are considered, five independent sets of decision trees (*i.e.* five forests) are learned. Each of them is a classifier predicting the following classes:

- one class for each monitored keyword;
- one unknown class for the other keywords. It corresponds to legitimate traffic that should not raise any alert. This class mixes traces from different keywords and could be assimilated to noise. Such a methodology has been successfully leveraged in [10] to analyse anonymized network traffic.

The H2Classifier[25] prototype is based on Python scikit-learn, which implements the CART (Classification And Regression Tree) decision tree algorithm.

A. Features extraction

As input of the random forest algorithm, a trace is represented as a vector of values. We voluntary exclude all time-based features to reduce the impact of the network environment (connection stability and latency) on the classification. We consider features usually described in related work about (encrypted) traffic classification, but also new features to fit with HTTP/2 protocol. The final vector of features is presented in Figure 11 and detailed hereafter.

1) Usual features from related work: We have selected some of the features used for encrypted traffic analysis [7], [10].

(a) Connection statistics (3 features): Number of incoming and outgoing TLS records, total number of bytes exchanged at the TLS layer.

[25]https://gitlab.inria.fr/pbrissau/h2classifier.git

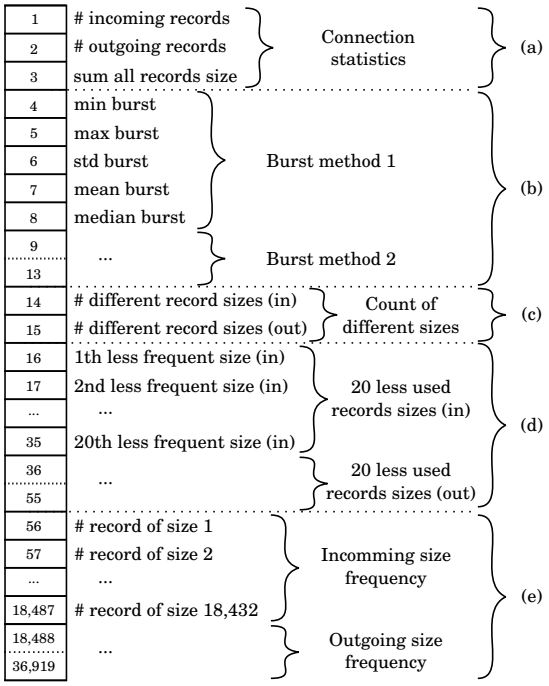


Fig. 11: H2Classifier features

(b) Burst statistics (10 features): Two methods have been defined to measure the burst of incoming packets (from the server). First, we collect the amount of data sent by the server between two consecutive packets sent by the client. The different observed values are stored in a list from which we extract the minimum, maximum, standard deviation, mean and median. The second method to evaluate the burst of data sent by the server counts every 20 TLS records the number of records sent by the server. The same statistical properties are then derived.

(c) Number of different sizes (2 features): For the incoming and outgoing data, this feature is the number of different TLS records' sizes encountered.

2) Other features: In order to fit with the particularity of HTTP/2, we extend the previous feature set with the following ones:

(d) Top 20 most representative sizes of TLS records (2×20): based on the rationale presented in Section VII-A, the 20 TLS record sizes with the lowest frequencies, from input and output packets, are extracted as features. They thus compose a set of 40 individual values (size in bytes) in our vector-based representation of the trace.

(e) Size distribution (up to 2×18,432): it corresponds to all TLS record size frequencies for both the request and response. To have fixed-size vector for all traces, one could consider all potential TLS record sizes. As explained in Section VII-A the largest possible size is $2^{14} = 16,384 + 2048$ bytes for a record. While it is a theoretically highly dimensional vector, in practice, most values are never observed and so not considered as a feature. Finally, the number of effectively used sizes (for both directions) is for example 28,699 for *Amazon*, 5073 for *Instagram*, 2165 for *Google*, 14,769 for *Google Images* and 17,710 for *Google Maps*.

B. Evaluation

To perform the evaluation we first fit the model by tuning both the hyperparameters and the model parameters. Then we test the H2Classifier with the selected parameters and assess the classification performance.

1) Hyperparameters tuning: model fitting is based on a 5-cross validation. To avoid over-fitting, we need to have a large number of samples for the learning. *data_h2_500* is thus adequate with 60 traces per keyword. For this first step we only use 52 traces, the other 8 traces are reserved for model parameters tuning (4/8) and testing phases (4/8). As *data_h2_500* only provides monitored keyword traces, *data_h2_2000* is also used for providing legitimate keyword traces. We note that unknown keywords form a single class. Individual traces of such keywords must be used jointly to build this class rather than collecting multiple traces of the same unknown keyword. We thus consider 200 random traces of legitimate keywords from *data_h2_2000* to be representative of this class for learning purpose.

In Figure 12(a), accuracy is the mean of the 5-cross validation. The maximum tree depth and the number of trees vary between 30 and 70 and between 50 and 500 respectively. For the sake of clarity, the curve for the maximum depth is not always reported. However, no noteworthy improvement can be obtained with a depth higher than 50. According to Figure 12(a), using more than 400 trees is not beneficial whereas it increases the size of the model by design, and so leads to a higher runtime. We thus set the number of trees as 400 and the maximum depth as 50 for the rest of our evaluation.

2) Model parameters tuning: our goal is to evaluate the impact of the model parameters on accuracy. These parameters are the number of legitimate traces and the number of features to use for training. For learning the monitored keywords, we use all the 52 traces of *data_h2_500* (for each keyword) from the previous step. Legitimate traces (from unknown keywords) are selected from *data_h2_2000*. The evaluation traces for the monitored keywords are composed of four unused traces (4 of the 8 traces remaining for each of the 500 monitored keywords, so 2000 traces in total) from *data_h2_500*. The evaluation traces for the legitimate keywords are 5000 unused traces of legitimate keywords (each trace is related to a different keyword) captured in *data_h2_2000*.

The results highlighted in Figure 12(b) correspond to the accuracy calculated over the evaluation traces when the number of legitimate traces used for the training vary between 50 and 200. In the figure, accuracy is stable for Google Maps, Amazon and Google. Hence, identifying user actions for these services do not require a huge number of traces for learning. However, we observe a strong growth when increasing this number up to 80 and a slight improvement between 80 and 130 for Google images. To keep our technique as generic as possible, we consider 130 as the appropriate number of traces from legitimate keywords to be used for learning independently of the targeted services.

Another important aspect of H2Classifier is the number of useful features to be used. While a full set has been defined, it is possible to reduce this set and thus the complexity of

TABLE VI: Final results for H2Classifier
(The values are the mean of 5 evaluations, Standard deviation is not specified because always under 0.3%)

Dataset	*data_h2_500* + partial *data_h2_2000*				Full *data_h2_2000*				
Source	Amazon	Google	Google Images	Google Maps	Amazon	Google	Google Images	Google Maps	Instagram
TPR	0.949	0.896	0.975	0.964	0.838	0.645	0.987	0.927	0.938
TNR	0.986	0.999	0.998	1	0.995	0.999	0.999	1	n.a.
FPR	0.066	$3.8.10^{-3}$	0.011	0	0.053	3.10^{-3}	5.10^{-3}	0	n.a.
FNR	0	0	0	0	3.10^{-5}	1.10^{-4}	1.10^{-4}	0	n.a.
WCR	0.050	0.103	0.024	0.036	0.161	0.354	0.011	0.073	n.a.
Accuracy	0.981	0.982	0.994	0.994	0.981	0.967	0.998	0.993	0.938

the classifier and the risk of over-fitting. To achieve that, we compute the importance of each feature for the classifier (based on the Gini criterion). Each feature has a score between 0 and 1; the higher the score, the more important the feature. Figure 12(c) is based on varying the number of the most important features that are kept. While there are some subtle differences between services, there is no major observable improvement for any of them with more than 300 features.

3) Testing: through the previous experiments, H2Classifier can be properly tuned (number of trees, maximum depth, features and number of legitimate traces). Although accuracy is a synthetic metric to assess the impact when parameters vary, Table VI shows more detailed results considering the parameters inferred previously (130 traces to model the legitimate traffic, 400 trees, a maximum depth of 50 and limiting the number of features to 300).

The results presented in this section were obtained considering an open world situation (except for Instagram as explained in Section V-C). As a reminder, the problem to address is to detect whether a trace has been generated by a monitored keyword or not, and to possibly identify this keyword. All the traces were collected with a real web-browser and we did not apply any filtering on traces before the classification. Thus, these traces are representative of real applications, and so may be erroneous because of connection issues for example.

The first experiment reported in the left part of the Table VI relies on the training set derived from *data_h2_500* and *data_h2_2000* (see model parameters tuning). The testing set is composed of the last four traces of *data_h2_500* per keyword (*i.e. 2000 traces*) as well as 10,000 traces from legitimate keywords of *data_h2_2000*, which have not been used yet. The second experiment (right part of Table VI) is performed with the full dataset (data_h2_2000) always with the previously tuned parameters. In this experiment a single random trace of each monitored keyword is used for the testing. Additionally 20,000 traces for the legitimate traffic except the 130 traces randomly selected for the training are added to the testing set.

On the one hand, a major difference between the two experiments is about the TPR. Except for Google Images, which has a slightly better result with the large dataset, the results are better for the smallest one. It is due to the number of traces available in each dataset (12 vs 52) and the number of different classes (500 vs 2000), that make thus the prediction problem more difficult with the large dataset. It is confirmed by the WCR values showing the same trend.

On the other hand, our model is really efficient in order to detect a monitored keyword. Indeed the FNR is often null or at least very low for every service while the TPR is quite high even with the large dataset. The major exception is for Google. It is explained by a rather limited variability in objects loaded as the requests mostly return text and hyperlinks only. FPR is also very low in most of cases. However, for the Amazon service, the FPR is not negligible as it is between 5.3 and 6.6% contrary to the other services (0-1.1%).

The Instagram service has been evaluated in a closed world situation. With a TPR of 93.8%, the results are aligned with the other observations.

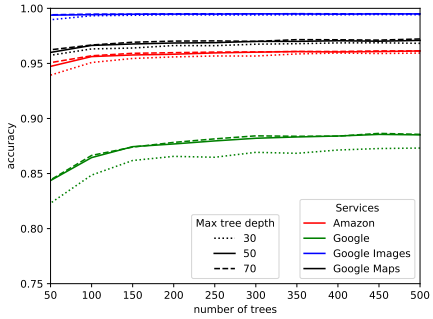
Finally, the high scores achieved by our solution for both datasets regarding our accuracy metrics prove the efficiency of H2Classifier for its envisioned application.

IX. Discussion

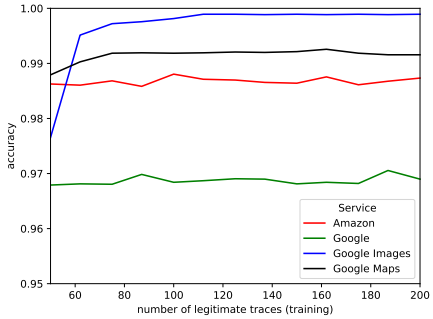
A. Temporal evaluation

Because web content evolves over time, models learned by Random Forest would be obsoleted after a certain period of time. Moreover, the service itself might be updated (*e.g.* how the content is loaded and pre-fetched). As a result, the traces produced by the same page might differ. In our case, the HTTP2 datasets have been collected with a 6 months

(a) Results of the 5-cross-validation (eval. number tree and max. depth)



(b) Evaluation of the number of traces need for the training of the legitimate class



(c) Evaluation of the number of feature based on the accuracy

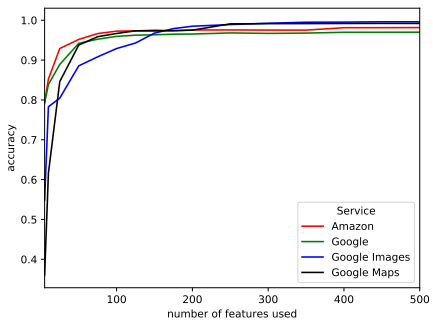


Fig. 12: Parameters evaluation

TABLE VII: Environment Impact Results
(5 experiments mean)

Options	Default	F1A	F1B	F2	F3	G1
Acc.	1	0.62	0.58	0.63	0.04	0.46
Std.	0	0.02	0.09	0.03	0	0.04
Options	G2A	G2B	G3A	G3B	G4A	G4B
Acc.	0.13	0.07	0.49	0.09	0.07	0.64
Std	0.03	0.02	0.02	0.01	0.01	0.03

difference. Leaning the model with *data_h2_500* to detect monitored keywords present in *data_h2_2000* leads to poor results that are out of any practical application. While it confirms our expectation, the learning is very efficient as only 5 minutes are necessary to build the model for 500 keywords. Crawling services and updating frequently the models can be thus easily automated. In our future work, we will investigate the period of time a model remains valid.

B. Configuration Impact

The configuration of the service or the web browser impacts on the loaded content and so probably on the H2Classifier accuracy. We consider the following possibilities:

- Firefox options:
  - User Agent[26] Firefox (default), Android (F1A) or Safari (F1B).
  - Firefox languages: French (default) or German (F2).
  - Screen resolution: FullHD (default), HD (F3).
- Google Images options:
  - Safe search: deactivate (default), activate (G1).
  - Image size criterion: all images (default), filtering large images (G2A) or small images(G2B).
  - Image colours criterion: all images (default), images with colour (G3A) or black and white images (G3B).
  - Date criterion: no date filtering (default), last week (G4A) or last year (G4B) filtering.

For this experimentation, we captured traces of 25 keywords on Google Images (50 times) for each of the configurations above. Those 25 keywords have been randomly selected from the set of keywords used in *data_h2_500*. This dataset is leveraged for learning purposes while our objective is to evaluate the ability of our technique to classify the new traces generated by the different configurations. The results shown in Table VII highlight different cases. Traces produced by some configurations (like F1A, F2 or G4B) can be classified to some extent but with a significant decrease of the accuracy. However, in other cases, the classification becomes totally ineffective (like F3, G3B or G4A) exhibiting an accuracy below 0.1. This drawback can be mitigated by using a mix of different configurations as learning traces or by creating a class for each couple keyword-configuration (and so learn the model accordingly). The impact of the environment can also depend on the keywords. For example the safe-search option only impacts very specific keywords.

[26]Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:63.0) Gecko/20100101 Firefox/63.0 ; Mozilla/5.0 (Linux; Android 7.0; PLUS Build/NRD90M) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.98 Mobile Safari/537.36 ; Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1 Safari/605.1.15

During the crawling, we deactivated the cache to avoid altering page content based on historic activities. However, if a user visits a cached page of a monitored keyword, the latter has been already detected on the first load. Furthermore, the cache induced by another page is limited because the content generated by two different requests are often very different.

Finally, although our experimentation relies on traffic protected by TLS 1.2, the solution is compatible with TLS 1.3 because the record layer, which is leveraged by our method, did not evolve from 1.2 to 1.3.

C. Resources consumption

Assuming the dataset with 500 keywords, *data_h2_500*, we consider in this section the resources consumption of H2Classifier. There are two important metrics. First, the size of the learned model needs to be kept in memory. It corresponds to the trees, which have been constructed. Second, the time to test a trace that measures the ability to analyze a new trace, assuming in our case a 3.2GHz CPU.

TABLE VIII: Memory and time consumption of the algorithm

Service	Google	Google Images	Google Maps	Amazon
Memory used for learning (in GB)	21.5	3.2	5.4	8.5
Time for testing (per trace in ms on 1 CPU)	15.1	110	18.9	18.9

The results are presented in Table VIII. We notice that the amount of memory used for the different services highly differs especially for google search engine. It is due to a lower heterogeneity in content sizes returned by google search compared to a page full of images (for example with Google Images). Therefore, to make distinction between pages, every single feature is subject to numerous conditional tests, knowing that our implementation with scikit-learn allows creating multi-variable conditions at each node of the decision trees.

X. Conclusion

This paper proposes a solution to classify different user actions when using a web service over TLS. In this paper, user actions mainly refer to, but are not limited to, keyword searches with a particular engine. Based on our prior work on HTTP/1.1 over TLS [2], we introduced the main differences introduced with HTTP/2 that particularly impact the classification methodology. We thus described our new solution H2Classifier, which addresses the HTTPS protocol when referring to HTTP/2 over TLS. The core idea resides in combining well-known features from related work and a new set of features reflecting the content that is downloaded when accessing a web page, *i.e.* the significant objects that compose the latter. Our results highlight the viability and the practicability of our technique with five widely used web services using large realistic datasets, which are made publicly available to support reproducible research. The achieved overall accuracy is between 94% and 99% depending on the monitored services.

To summarize, H2Classsifier is a service-agnostic method, which is able to efficiently analyze HTTP2 traffic to identify predefined user actions.

Although user actions are atomic operations, they can support a behavioral (*i.e.* multi-steps) modeling to provide advanced security policy verification. For example, denying an access to a service could be envisioned if a user repeats inappropriate actions. It will thus reduce the effect of false positives. Therefore, our future work will be focused on user behavior modeling over HTTP/2, supporting the different user configurations and automating the update of the classifier to maintain its efficiency over time.

ACKNOWLEDGEMENT

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreements No 830927 (project CONCORDIA) and No 830892 (project SPARTA). We thank the High Security Lab (https://lhs.loria.fr) for hosting our dataset an J. McAuley providing valuable metadata for building our experimentation [41].

REFERENCES

[1] E. Rescorla, "Http over tls," RFC 2818, 2000.
[2] P.-O. Brissaud, J. François, I. Chrisment, T. Cholez, and O. Bettan, "Passive monitoring of https service use," in *14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018.
[3] D. Wagner, B. Schneier *et al.*, "Analysis of the ssl 3.0 protocol," in *The Second USENIX Workshop on Electronic Commerce Proceedings*, 1996.
[4] H. Cheng and R. Avnur, "Traffic analysis of ssl encrypted web browsing," 1998.
[5] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Security and Privacy*. IEEE, 2002.
[6] A. Hintz, "Fingerprinting websites using traffic analysis," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2002.
[7] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. ACM, 2011.
[8] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.
[9] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting." in *USENIX Security Symposium*, 2014.
[10] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique." in *USENIX Security Symposium*, 2016.
[11] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*. ACM, 2006.
[12] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009.
[13] L. Lu, E.-C. Chang, and M. C. Chan, "Website fingerprinting and identification using ordered feature sequences," in *Computer Security – ESORICS 2010*, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds. Springer Berlin Heidelberg, 2010.
[14] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," in *ACM SIGCOMM Computer Communication Review*. ACM, 2015.
[15] Y.-H. Lin, S.-H. Shen, M.-H. Yang, D.-N. Yang, and W.-T. Chen, "Privacy-preserving deep packet filtering over encrypted traffic in software-defined networks," in *Communications (ICC)*. IEEE, 2016.
[16] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, 2015.
[17] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A multi-level framework to identify https services," in *Network Operations and Management Symposium (NOMS)*. IEEE, 2016.
[18] A. Pescape, A. Montieri, G. Aceto, and D. Ciuonzo, "Anonymity services tor, i2p, jondonym: Classifying in the dark (web)," *IEEE Transactions on Dependable and Secure Computing*, 2018.
[19] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, Jan 2018.
[20] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing android encrypted network traffic to identify user actions," *IEEE Transactions on Information Forensics and Security*, Jan 2016.
[21] J. Liu, Y. Fu, J. Ming, Y. Ren, L. Sun, and H. Xiong, "Effective and real-time in-app activity analysis in encrypted internet traffic streams," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017.
[22] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, and J. Qian, "Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic," in *10th USENIX Workshop on Offensive Technologies*. USENIX Association, 2016.
[23] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescap, "Multi-classification approaches for classifying mobile app traffic," *J. Netw. Comput. Appl.*, 2018.
[24] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescap, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, 2019.
[25] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescap, "Know your big data trade-offs when classifying encrypted mobile traffic with deep learning," 05 2019.
[26] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service usage classification with encrypted internet traffic in mobile messaging apps," *IEEE Transactions on Mobile Computing*, Nov 2016.
[27] A. Reed and M. Kranch, "Identifying https-protected netflix videos in real-time," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 2017.
[28] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot me if you can: Uncovering spoken phrases in encrypted voip conversations," in *Security and Privacy*. IEEE, 2008.
[29] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson, "Language identification of encrypted voip traffic: Alejandra y roberto or alice and bob?" in *USENIX Security Symposium*, 2007.
[30] S. E. Coull and K. P. Dyer, "Traffic analysis of encrypted messaging services: Apple imessage and beyond," *ACM SIGCOMM Computer Communication Review*, 2014.
[31] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on ssh." in *USENIX Security Symposium*, 2001.
[32] "Traffic analysis on google maps with gmaps-trafficker," white paper, 2012. [Online]. Available: http://blog.ioactive.com/2012/02/ssl-traffic-analysis-on-google-maps.html
[33] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," RFC 5246, 2008.
[34] "Remove h1 pipeline support," bugzilla, 02-2017. [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=1340655
[35] "Http pipelining," documentation, 2015. [Online]. Available: https://www.chromium.org/developers/design-documents/network-stack/http-pipelining
[36] R. Peon and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," RFC 7540, 2015.
[37] S. Friedl, A. Popov, A. Langley, and S. Emile, "Transport layer security (tls) application-layer protocol negotiation extension," RFC 7301, 2014.
[38] R. Peon, "Hpack: Header compression for http/2," RFC 7541, 2015.
[39] T. Groleat, S. Vaton, and M. Arzel, "High-speed flow-based classification on fpga," *International journal of network management*, 2014.
[40] B. Xiong, C. Xiao-su, and C. Ning, "A real-time tcp stream reassembly mechanism in high-speed network," *South West Jiaotong University*, 2009.
[41] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015.
[42] F. Mosteller, "Understanding the birthday problem," in *Selected Papers of Frederick Mosteller*. Springer, 2006.
[43] E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, 1962.
[44] IETF, "Hypertext transfer protocol – http/1.1," RFC 7230-7237, 2014.
[45] I. Grigorik, "Optimizing tls record size & buffering latency," Blog, 2013. [Online]. Available: https://www.igvita.com/2013/10/24/optimizing-tls-record-size-and-buffering-latency
[46] L. Breiman, "Random forests," *Machine learning*, 2001.