How HTTP/2 is changing Web traffic and how to detect it

Jawad Manzoor
Université Catholique de Louvain
Email: jawad.manzoor@uclouvain.be

Idilio Drago
Politecnico di Torino
Email: idilio.drago@polito.it

Ramin Sadre
Université Catholique de Louvain
Email: ramin.sadre@uclouvain.be

***Abstract*—HTTP constitutes a dominant part of the Internet traffic. Today's web traffic mostly consists of HTTP/1 and the much younger HTTP/2. As the traffic of both protocols is increasingly exchanged over encryption, discerning which flows in the network belong to each protocol is getting harder. Identifying flows per protocol is however very important, e.g., for building traffic models for simulations and benchmarking, and enabling operators and researchers to track the adoption of HTTP/2.**

This paper makes two contributions. First, using datasets of passive measurements collected in operational networks and Deep Packet Inspection (DPI), we characterize differences in HTTP/1 and HTTP/2 traffic. We show that the adoption of HTTP/2 among major providers is high and growing. Moreover, when comparing the same services over HTTP/1 or HTTP/2, we notice that HTTP/2 flows are longer, but formed by smaller packets. This is likely a consequence of new HTTP/2 features and the reorganization of servers and clients to profit from such features.

Second, we present a lightweight method for the classification of encrypted web traffic into appropriate HTTP versions. In order to make the method practically feasible, we use machine learning with basic information commonly available in aggregated flow traces (e.g., NetFlow records). We show that a small labeled dataset is sufficient for training the system, and it accurately classifies traffic for several months, potentially from different measurement locations, without the need for retraining. Therefore, the method is simple, scalable, and applicable to scenarios where DPI is not possible.

***Index Terms*—HTTP, Characterization, Machine Learning**

I. Introduction

HTTP has occupied a dominant position among Internet traffic, and it is expected to maintain such position in the future as more and more applications are getting migrated to the Web. The constant evolution of the Web has resulted in various updates to HTTP. Since its first version, the behavior of HTTP in the wild has been intensively studied and the gained insights have been used, for example, by web developers to improve the performance of web applications [1], by researchers to build traffic models for simulation and benchmarking [2], and by operators and service providers to track the adoption of new protocol versions [3].

With the advent of new HTTP versions (e.g., HTTP/2), it is expected that the traffic share of coexisting protocols will change. Not a surprise, the biggest Internet companies are usually the early adopters. As a consequence, our knowledge about the behavior of web traffic has to be updated, since these protocols may differ significantly in terms of key metrics, such as request/response sizes, flow duration and number of connections. However, identifying the used HTTP version in large-scale traffic measurements is not trivial. First, since HTTP/2 traffic is mostly encrypted, and HTTP/1 is fast being migrated to HTTPS as well, any practical identification method has to operate without payload inspection. Second, whereas fields in TLS handshakes still provide hints about the used HTTP version as we will discuss later, this information is not available in basic flow-level statistics [4] (e.g., NetFlow records) usually available in real deployments. Thus, operators and researchers have no means to monitor the protocol adoption at large scale, or to study practical differences of the protocols from the network point-of-view.

This paper makes two contributions. First, using datasets of flow-level statistics collected in operational networks, augmented with features extracted by means of Deep Packet Inspection (DPI), we provide a comparison of flow-level characteristics of HTTP/1 and HTTP/2 traffic. We confirm [3] a fast increase in HTTP/2 adoption among top Internet players. Furthermore, we show that new features introduced in HTTP/2, such as header compression, together with likely reorganizations of sites and servers to profit from such features, result in clear differences in the network fingerprints of the protocols. Indeed, HTTP/2 flows are on average 36% longer in duration than HTTP/1 flows, while HTTP/2 request and response packets are on average 40% and 29% smaller than in HTTP/1 for the same service, respectively.

Second, these differences in basic traffic features motivate the search for methods to identify HTTP versions when DPI is not possible. We propose the use of machine learning with basic features available in NetFlow measurements to classify HTTP traffic – thus, introducing a classifier that is able to operate with existing traces and monitoring equipment, and easily scalable to high-speed networks. We validate the classifier using the real traces where ground truth is known by means of DPI on TLS handshakes. We test various learning algorithms and conclude that decision trees are quite suitable for this problem. After training the system with a relatively small set of flows, the classifier can achieve very good accuracy at high speed with only basic NetFlow features. Finally, we show that the system accurately classifies traffic from different locations and for several months without the need for continuous retraining.

Next, Section II discusses the evolution of HTTP and related work. We characterize HTTP traffic based on passive traces

in Section III. Our HTTP classifier is presented in Section IV and validated in Section V, whereas Section VI concludes the paper.

II. Background and Related work

A. Evolution of HTTP

HTTP was initially designed for the delivery of simple web pages. In HTTP/1.0, a separate network connection is required to request each resource. HTTP/1.1 introduced the concept of *persistent connections*, so that the same connection is reused to request additional resources from the server. To further improve speed, *pipelining* allows the client to asynchronously send multiple requests to the server without waiting for the response. However, the specification requires that the server sends the responses in the same order in which it received the requests, which can result in so-called *head-of-line* (HOL) blocking. A widely used optimization technique in HTTP/1.1 is to open multiple connections to the server.

After standardization of HTTP/1.1 there was no major update of the protocol for nearly 15 years. Instead, several tweaks like *spriting*, *inlining* and *domain sharding* were introduced to improve performance. In 2015, HTTP/2 was standardized with the objective of improving web experience by reducing latency. Among others, HTTP/2 has the following major features: (i) multiplexing, i.e., the use of multiple streams inside a single TCP connection for concurrent requests; (ii) reducing bandwidth by using header compression; (iii) allowing servers to push data to clients; and (iv) content prioritization, to allow clients to specify the order in which resources should be sent by servers, so to render pages more efficiently.

B. Related work

Given that HTTP has become the *de facto* protocol for deploying new applications and services, there are many studies on HTTP/1 web traffic, e.g., [2], [5], [6]. With the advent of protocols like QUIC, SPDY and HTTP/2 the characteristics of web traffic are bound to change. Performance improvements promised by these new protocols are studied in [7], [8], [9], [10]. In [3], the authors build a measurement platform that actively monitors HTTP/2 adoption. They also provide content analysis and page-level characterization of HTTP/1 and HTTP/2 traffic using active measurements. We use passive measurements to provide a flow-level characterization of HTTP/1 and HTTP/2 traffic.

Machine learning has been used for traffic classification for more than a decade. Various machine learning methods have been employed to identify the applications and protocols behind Internet traffic (see [11] for a survey on early work in this field). We follow a similar idea and explore whether machine learning algorithms are effective in identifying the exact HTTP version used in encrypted web traffic.

Most work on traffic classification relies on DPI or on the analysis of packet-level features, such as packet sizes. Since our objective is to create a lightweight method, we only use aggregated flow-level features which are computationally less expensive. This advantage of flow data has been also recognized by others: For example, the authors of [12] analyze NetFlow records to detect HTTPS webmail traffic. As we are dealing with web traffic transported over TLS, we use information from TLS handshake messages to identify the HTTP version to construct the ground truth. This information is however only exploited for evaluating the performance of the algorithms, which are trained with basic fields present in ordinary NetFlow records.

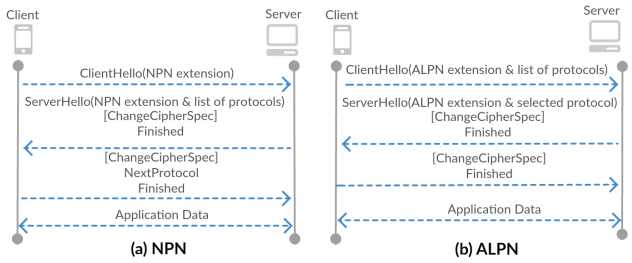


Fig. 1: TLS handshake with application protocol negotiation.

III. Characterization of HTTP/2 traffic

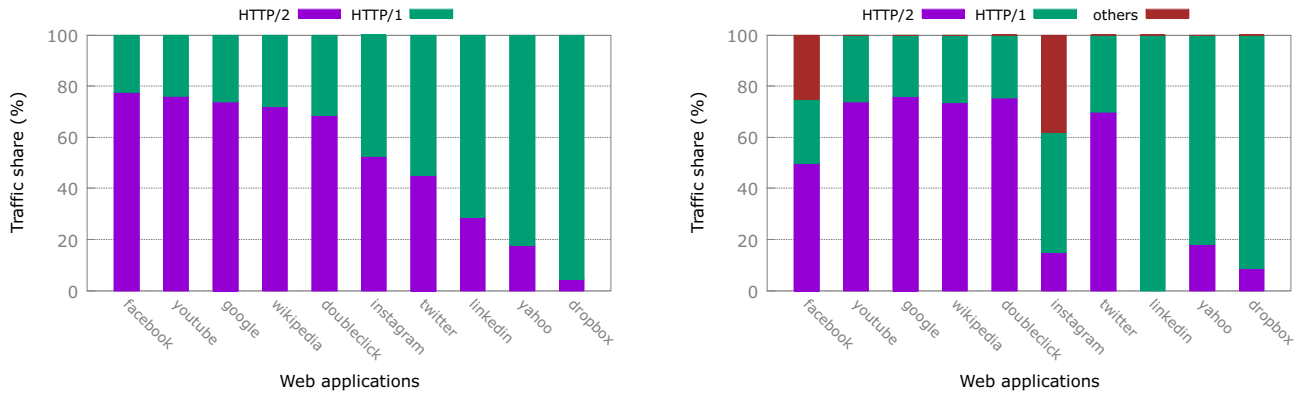
The new features of HTTP/2 are expected to result in a different network fingerprint when compared to HTTP/1. In this section, we compare flow-level characteristics of HTTP/1 and HTTP/2 traffic. We first describe our datasets and how we have built the ground truth. Then we discuss the adoption of HTTP/2 by popular web services and give a flow-level characterization of the observed traffic.

A. Datasets

We rely on data exported by Tstat [13] in our evaluation. Tstat acts as an advanced flow exporter, monitoring all TCP connections in the network and exporting more than 100 metrics for each flow. These metrics include all basic features present in popular versions of NetFlow — e.g., IP addresses, bytes and packet counters and flow start/end timestamps. Tstat adopts the classic five-tuple definition for a flow: client and server IP addresses, client and server port numbers and the transport layer protocol define which packets belong to a flow.

We collected traffic traces with Tstat in two different networks from June to December 2016. The *Campus* dataset was collected at border routers of a European university campus network. It includes traffic of around 15 000 users connected through wired network in research and administrative offices as well as WiFi access points. The *Residential* dataset was collected at a Point of Presence (PoP) of a European ISP. This network includes traffic of around 25 000 households that are provided access via ADSL and Fiber To The Home (FTTH).

Although the HTTP/2 specification does not require the use of TLS, currently all major browsers only support encrypted HTTP/2 traffic. This means that negligible amount of plaintext HTTP/2 traffic is expected on TCP port 80. Since we are interested in classifying HTTP flows transported over TLS, we isolate the flows with server port equal to 443, discarding all the remaining flows.



(a) June 2016

(b) December 2016

Fig. 2: HTTP/1 and HTTP/2 share of ten popular web applications.

B. Ground truth

To build the ground truth, we analyze the NPN (Next Protocol Negotiation) and APLN (Application-Layer Protocol Negotiation) information exported by Tstat. NPN and ALPN are TLS extensions that allow clients and servers to negotiate the application protocol to be used in a connection. Since 2016, NPN is deprecated and has been replaced by ALPN, which is an IETF standard. Clients may use both NPN and ALPN extensions, while servers will always use only one of them. The sequence of steps taken in NPN and ALPN negotiations is shown in Figure 1.

The negotiation of the application layer protocol using NPN involves the following steps [14]: (i) the client sends the `next_protocol_negotiation` extension in the TLS `ClientHello` message with empty `extension_data`; (ii) the server sends a list of supported application protocols in the `ServerHello` message; (iii) the client selects *one* suitable protocol and sends it back in the `next_protocol` message under encryption before finishing the TLS handshake.

ALPN is instead negotiated in two steps [15]: (i) the client sends a list of supported application protocols using the `application_layer_protocol_negotiation` extension in the TLS `ClientHello` message; (ii) the server selects *one* suitable protocol and sends it back in the `ServerHello` message. In this way, the protocol negotiation is done in the clear in a single round trip within the TLS handshake.

Tstat has methods to extract NPN and ALPN information from the TLS handshake. It observes non-encrypted TLS messages and exports the list of protocols offered by clients and servers. We annotate all HTTPS flows with the selected application layer protocol using the NPN and ALPN flags exported by Tstat. Those labels serve as the ground truth to train the machine learning models (see Section IV), as well as to validate the classification performance. Remaining features exported by Tstat are discarded from now on. In our datasets the HTTPS flows comprise roughly 68% HTTP/1, 27% HTTP/2 and 5% other non-HTTP traffic on port 443.

C. HTTP/2 usage evolution in popular web applications

According to [16], the majority ($\approx$ 77%) of the globally used web browsers support HTTP/2, therefore the adoption of HTTP/2 mainly depends on the support provided by the servers. To find out which HTTP versions are in use by some of the most popular web applications we use information from the TLS Server Name Indication (SNI) extension. This extension allows a client to indicate to the server which hostname it attempts to connect to. Tstat extracts the SNI hostname for each flow and we includes this feature in our ground truth. We aggregate all the flows that belong to the same second-level domain and also assign flows from known content delivery networks (CDN) to the respective web application, e.g., *fbcdn* is used by Facebook Inc. Figure 2 shows ten popular web applications and their usage of HTTP/1 and HTTP/2 in June and December 2016. Google, YouTube, DoubleClick ad service and Wikipedia heavily use HTTP/2 and it has remained constant in the last six months. Twitter increased the usage of HTTP/2 for its services from 45% to 70%. Interestingly Linkedin had around 30% share of HTTP/2 traffic in June, but now it has completely switched back to HTTP/1. Dropbox still mostly uses HTTP/1 and its share of HTTP/2 has only slightly increased.

The most interesting change is in Facebook and Instagram traffic. In the past a majority of this traffic was HTTP/2, but now only 50% of Facebook and 15% of Instagram traffic constitutes HTTP/2. In Figure 2b we can see that the 26% of Facebook and 38% of Instagram traffic on port 443 belong to some protocol other than TLS. This is caused by a recent migration of Facebook to a proprietary protocol in mobile devices [17]. The Facebook protocol, called *Zero*, is a zero round-trip (0-RTT) security protocol implemented over TCP and based on QUIC's crypto protocol. At the application level, Facebook still relies on HTTP/2 in connections negotiated using the Zero protocol. Facebook claims to have significantly reduced the connection establishment time (and therefore the mobile app's cold start time) using this protocol. Since this

traffic does not use standard TLS protocol, it was labeled as other non-HTTP traffic by Tstat during our captures.

D. Flow-level characterization of HTTP/1 and HTTP/2

It is expected that HTTP/2 will have a different network fingerprint when compared to HTTP/1 due to the newly introduced features. For instance, the use of plain-text headers in HTTP/1 causes the protocol to (usually) take many round trips to send headers from clients to servers for a single HTTP request. HTTP/2 can perform similar operations in a single packet, thanks to header compression, which can reduce the header size by roughly 30% to 80%. Similarly, due to multiplexing in HTTP/2, fewer connections can be used to transfer various objects of a web page as compared to HTTP/1 [18]. Moreover, it is expected that both browsers and servers have been adapted to exploit new features of HTTP/2.

We quantify how all such differences impact HTTP traffic in Figure 3 by utilizing our ground-truth traces. We select a subset containing 1 hour of web traffic from June 2016 belonging to the five largest web applications (in terms of number of flows), namely Google, Facebook, Doubleclick, Youtube and Twitter. The subset contains around 700,000 HTTP/2 and 300,000 HTTP/1 flows.

Figure 3a shows the empirical CDF of the flow duration for the two protocol versions. We can see that the majority of HTTP/2 flows have longer duration. Around 25% of HTTP/1 flows are of very small duration, less than 1s, and another 13% last between 1s and 10s. In HTTP/2 only 5% flows are smaller than 1s and another 6% between 1s and 10s. Around 33% HTTP/1 and 53% HTTP/2 flows have a duration longer than 100s. On average, HTTP/2 flows are 36% longer than HTTP/1 flows. Furthermore, the duration distribution of HTTP/2 flows depicts two pronounced peaks at around 65s and 240s. We investigated it further and found that the flows in the first peak belong to Facebook and those in the second peak belong to Google domains. They are pronounced only in the HTTP/2 curve because majority of the flows of Facebook and Google belong to HTTP/2 which is evident from Figure 2. The reason for the two peaks is probably the different timeout values used at the server side.

We also observe that HTTP/2 flows usually have many more packets per flow, likely because they are longer in duration than the HTTP/1 ones. This effect is clearly visible in Figure 3b. For this figure, the CDF has been calculated separately for the data sent to the server (request) and received from the server (response). The conclusion is however similar for both traffic directions: 50% of the HTTP/2 flows have 20 or more packets in the traffic direction, whereas only 22% of HTTP/1 flows carry 20 or more packets.

Interesting, while flows are longer and carry more packets, packet sizes of HTTP/2 flows are reduced substantially. Figure 3c gives the empirical CDF of the *average packet size* of HTTP/1 and HTTP/2 flows. The average packet size is calculated by dividing the total amount of bytes by the number of packets in a flow. Again, independent lines are plotted for request and response packets. HTTP/2 request and response packets are 40% and 29% smaller in size than in HTTP/1, respectively. While our data does not prove causality, we conjecture that most of this effect is caused by header compression in HTTP/2. Around 80% of HTTP/2 and 55% of HTTP/1 request packets are smaller than 100 bytes. The advantage of header compression is clearly more prominent in smaller response packets (<600 bytes).

Figure 3d quantifies overall effects in flow sizes, by showing the empirical CDF of bytes per flow for the two protocol versions. Here a much more complicated figure emerges, in which many different effects are likely to interplay. Focusing on lines depicting response flow sizes, notice in the tails of the distributions that large flows (e.g., larger than 10000 bytes) are more common in HTTP/2 than in HTTP/1. The longer duration of HTTP/2 flows (e.g., due to client and server implementations) are again a reasonable explanation for that effect. On the other hand, observe that flows with very limited number of bytes (e.g., less than around 4000 bytes) are slightly more frequent in HTTP/2 than in HTTP/1 as well, probably thanks to the header compression feature of the former.

Finally, we take a random sample of 1500 flows from each HTTP version and depict their properties in scatter plots. Figure 3e shows the request and response packet size. In case of HTTP/2 most flows are clustered between 400 bytes request size and 800 bytes response size. For HTTP/1 the request size is evenly spread throughout the range while the response size is mostly above 600 bytes. HTTP/1 and HTTP/2 flows look cleanly separated into different clusters. Similarly, in Figure 3f HTTP/2 flows are grouped together with higher duration and smaller request size while the majority of HTTP/1 flows have smaller duration and larger request size. We next exploit these characteristics to derive a method to identify traffic of each protocol without inspecting payloads.

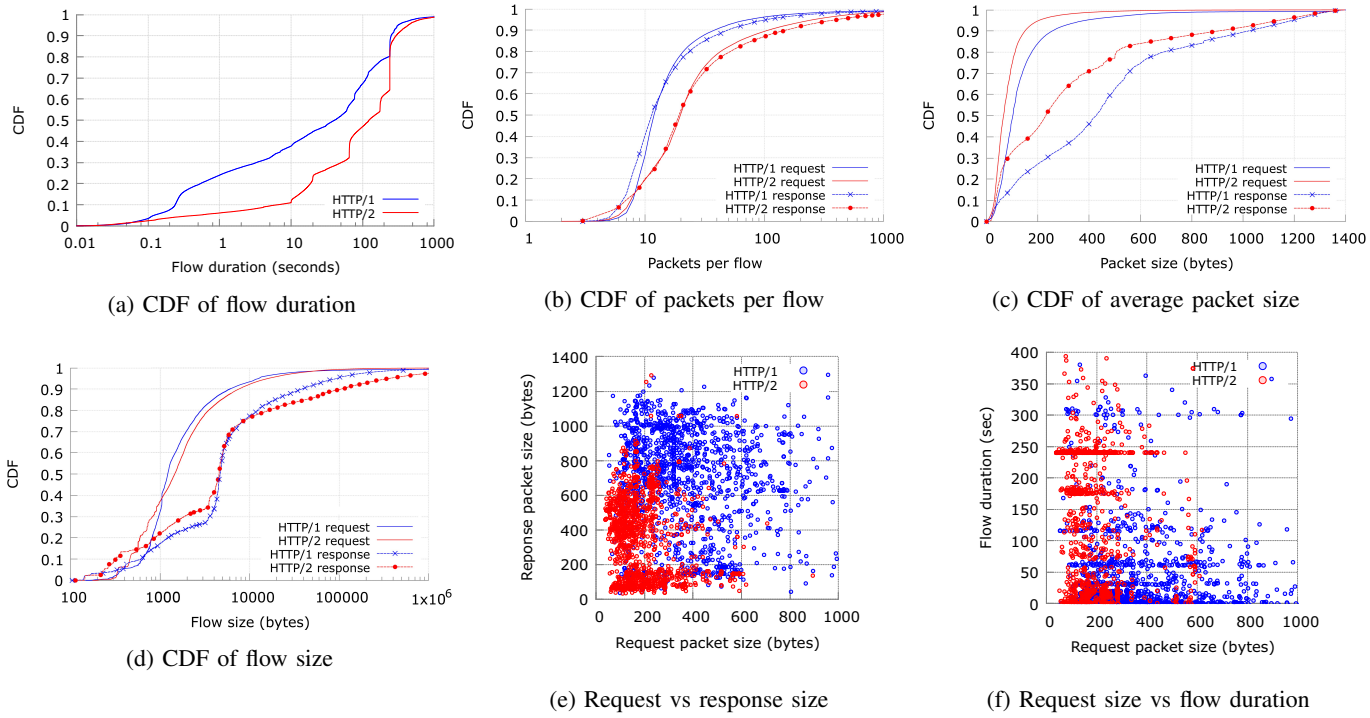
IV. Flow-based identification of HTTP/2

A. Outline

It is clear from Figure 3 that HTTP/1 and HTTP/2 traffic are different. This allows us to develop models that are able to classify traffic according to the HTTP versions without payload inspection. Since classification rules are hard to be manually derived from traffic, we opt to follow a machine learning approach.

In machine learning, classification is the task of assigning a *class* to unknown instances, based on features that describe the instances and a sample of instances with known classes. In our case, the instances to be classified are flow records exported by measurement devices, such as NetFlow exporters. The features that characterize the instances are the metrics exported by NetFlow (e.g., client and server IP addresses, bytes and packet counters etc.), as well as features derived from these data (e.g, average flow throughput). The classes to be assigned to flows are the possible HTTP versions (i.e., *HTTP/1* or *HTTP/2*) or *Others* for all remaining traffic.

Generally, the classification follows two steps in what is called *supervised learning*. Firstly, a dataset of instances with known classes is presented to the algorithm for *training*,



(a) CDF of flow duration

(b) CDF of packets per flow

(c) CDF of average packet size

(d) CDF of flow size

(e) Request vs response size

(f) Request size vs flow duration

Fig. 3: Characterization of HTTP/1 and HTTP/2 traffic.

i.e., the algorithm learns relations between features and the classes of the problem. Secondly, using the model built during the training phase, the algorithm assigns classes to unknown instances during the *classification* phase.

B. Classification algorithms

Our goal is to assess the machine learning approach for the identification of HTTP versions behind network flows. Among the many classification algorithms found in the literature, we take four different options and check their performance with our data. We are not interested in performing an exhaustive assessment of which algorithm is the best one, but instead we want to coarsely select an algorithm that delivers good performance for the particular problem we are solving.

We consider four algorithms: Bayesian Networks (Bayes Net), Naive Bayes Tree (NB Tree), C4.5 Decision Tree and Random Forest. These algorithms are appealing for being fast and for requiring little parameter tuning. Methods like Support Vector Machines or k-Nearest Neighbors are not considered due to their complexity or slower speed. For the sake of brevity, we refrain from providing details of the classification algorithms. We use the implementations offered by Weka, and readers can refer to [19] for further information about the algorithms and the toolset implementing them.

C. Evaluation methodology

We use the datasets described in Section III-A for training and testing the machine learning algorithms. Flows are labeled as HTTP/1, HTTP/2 or Others based on TLS handshakes (see Section III-B). This enables us to compare actual and predicted classes and calculate performance metrics.

We evaluate five classic machine learning performance metrics [20]: (i) *Accuracy* – the overall number of instances that are correctly classified; (ii) *Precision* – the number of instances correctly classified as belonging to a particular class; (iii) *Recall* – the number of instances belonging to a particular class that are correctly classified; (iv) *F-measure*:

$$F\text{-}measure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

(v) *Kappa statistic*:

$$Kappa = \frac{P_o - P_e}{1 - P_e}$$

where

- P_o = Observed Accuracy computed in the experiment
- P_e = Expected Accuracy, i.e. probability of accuracy by random chance.

Accuracy is the most popular metric, but it can be misleading particularly when there is large class imbalance. Precision and Recall are measures of exactness and completeness in respect to a particular class, whereas F-measure is a balance between the two. The Kappa statistic is considered a more robust metric because it takes into account the correct classifications that occur by chance.

The training set contains flows that are used to build classification models while an independent testing set contains flows representing the unknown traffic that we want to classify. To create training and testing sets, we use stratified 10-fold

cross-validation, in which the original dataset is divided into 10 equal subsets. One subset is used for testing while the remaining are used for training. This process is repeated 10 times and each time a different subset is used for testing. The results are averaged over all repetitions.

D. Feature selection

The features used for training a classifier are usually key for the classification performance. It is however hard to know upfront which features are informative for classification. Therefore, the training phase in supervised learning is usually accompanied by a *feature selection phase*, in which a large set of features is analyzed for shortlisting those with high discriminating power. This is achieved, firstly, by defining and extracting the largest possible number of features from the input data (e.g., using domain knowledge). Then, feature selection algorithms determine the most informative ones.

From the salient characteristics of the HTTP versions and results presented in previous sections, we create a set with 17 candidate features, including: the overall flow duration and, for both client to server communication and vice-versa, the number of packets, number of bytes, average number of bytes per packet, average packet throughput, average byte throughput, and whether the flows have SYN, ACK or FIN flag set.

Our goal of building a classifier that can operate with NetFlow data constraints the set of features we can extract. In particular, none of the above features requires access to packet payload, nor they require information beyond the transport layer (e.g., from TLS handshakes). Moreover, to keep the classifier lightweight, we do not use features that require correlation of multiple flows (e.g., the number of connections between client and server to load a web page), as it would introduce additional processing steps on raw NetFlow data.

We then apply correlation-based feature subset selection and best-first search algorithms to rank the features. Again, we use the implementation available in Weka [19]. We select the best subset of seven features with highest predictive power for training our classifiers: (i) Number of client bytes; (ii) number of server bytes; (iii) number of client packets; (iv) number of server packets; (v) average client bytes per packet; (vi) average server bytes per packet; and (vii) flow duration.

V. Classifier Validation

A. Classification performance

In order to find the optimal size for the training set, we first test the accuracy of learning algorithms using training sets of increasing sizes – ranging from 1 K to 1 M flows. We use the Campus dataset in this experiment. We notice that accuracy improves for all algorithms, provided that at least 100 K flows are in the training set, and it practically stalls for training sets containing more than 500 K samples. Therefore we conclude that a training set of limited size (e.g., 500 K instances) is sufficient for training the classifier.

We then perform 10-fold stratified cross-validation using both Campus and Residential datasets. The results are in Figure 4. Random Forest has the highest accuracy – i.e., 97.5% – followed by C4.5 and NBTree having roughly 95% and 93% accuracy. Overall, numbers are very similar in both datasets for these classification models, reinforcing conclusions. BayesNet has the lowest accuracy of 88% in Campus and 86% in Residential dataset. Table I shows that F-measure (average for the three classes) and Kappa coefficient values are quite high for all algorithms, except BayesNet. This shows the good classification power of the flow-based approach and confirms that the high accuracy is not just by chance.

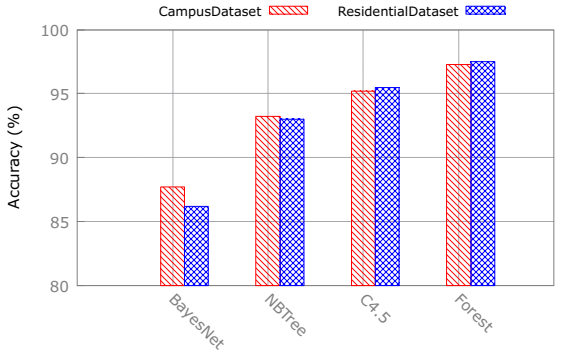


Fig. 4: Accuracy of the algorithms for different datasets.

TABLE I: Evaluation of the algorithms.

	Dataset	Campus	Residential
F-Measure	Random Forest	97.3	97.5
	C4.5	95.2	95.5
	NB Tree	93.2	93
	BayesNet	87.7	86.2
Kappa	Random Forest	94.8	94.7
	C4.5	90.8	90.4
	NB Tree	86.4	87
	BayesNet	76.5	70

To further explain results, confusion matrices for the campus dataset are shown in Figure 5. Rows mark the actual classes of instances, columns mark the predicted classes, and cells are normalized by number of instances of each actual class – i.e., each row sums up to 100%. Cells in the diagonals thus report the recall of the class, and other cells show how instances are misclassified. Focusing on Random Forest at Figure 5a, which has the best results, we notice how it consistently classifies instances of all classes with recall greater than 96%. Few errors are observed in particular for HTTP/2 flows that are mistakenly marked as HTTP/1.

B. Per service performance

There is usually a handful of web applications, such as video streaming and social networking, that generates the majority of traffic in any kind of network. It is crucial to verify that the classification accuracy of the proposed method is due to the characteristics of the protocol versions and not the applications.

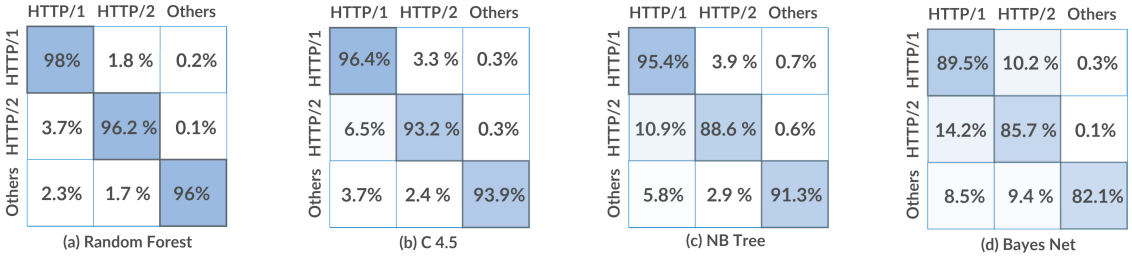


Fig. 5: Confusion matrices of the algorithms.

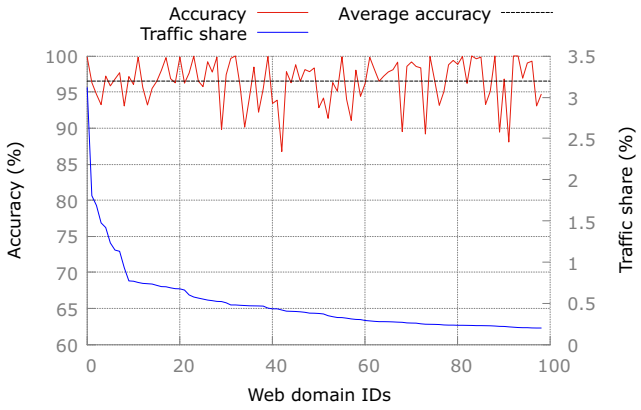


Fig. 6: Classification accuracy vs traffic share of top-100 domain names.

We use the SNI field available in the ground-truth to select the top-100 domain names according to the number of flows. We then compute the percentage of flows that are correctly classified for each domain name. Fig. 6 presents the accuracy per domain name (left y-axis) of top-100 domain names (x-axis) and their share of flows (right y-axis). The red curve represents the accuracy per domain name, while the blue curve represents the traffic share for each domain. The black horizontal line marks the overall average accuracy. As expected, we see that the top-5 domain names account for the biggest traffic share, with a long tail of less popular domain names. More important, it is evident from this graph that the traffic of most domain names has been classified accurately(¿90% accuracy). It verifies that the learning ability of the algorithms is not impacted by specific characteristics of popular applications and the traffic of the applications with very small share is also classified with high accuracy.

C. Temporal stability

Ground truth collection for training machine learning algorithms is not an easy process but frequent retraining is essential to maintain a certain level of accuracy in most cases because the old model becomes outdated after a certain amount of time. The retraining process can be performed manually under human supervision or automatically [21], but in either case it is not convenient for system administrators to frequently use

TABLE II: Long-term classification accuracy.

Date	Test Instances	Accuracy (%)
Jun	800K	95.8
Jul	700K	94.8
Aug	400K	95.9
Sep	1.5M	95.5
Oct	2.1M	91
Nov	2.2M	90.4
Dec	1.4M	90.8

DPI based methods on network traffic due to legal and privacy reasons. Therefore, the longer the model stays stable after initial training, the better it is. The temporal stability measures how accurate a classification method will remain over time. To test the temporal stability of our system, we used 1 hour of traffic trace from start of June 2016 as training set containing roughly 650K flows to build a classification model with the Random Forest algorithm.

We used seven test sets, each selected from a random day and time from each month from June to December 2016. The details of the test sets and the resulting classification accuracy are shown in Table II. From October onwards we see a big drop of 4-5% in classification accuracy. Upon investigation we found that this is caused by the use of the *Zero* protocol by Facebook and Instagram mobile apps which we have explained in Section III-C. Since this traffic does not use standard TLS protocol, it is labeled as other non-HTTP traffic by Tstat. It constitutes a big portion of around 28% of *other* traffic and therefore results in the increase of misclassification. In the future, Facebook is going to subsume this protocol into their implementation of TLS 1.3. To fix this issue for the time being, we labeled this traffic of the *Zero* protocol as HTTP/2 instead of *Other* for the December test set. We repeated the classification using the same model that was built from the training set from June and the accuracy increased to 92.3%.

We can see that the classification model built in June maintains a minimum accuracy level of at least 92% for six months, which shows that the system has high temporal stability and can continue to provide accurate classification for several months without the need for retraining.

D. Spatial stability

The spatial stability measures how classification models perform across different networks. We tested the spatial stability

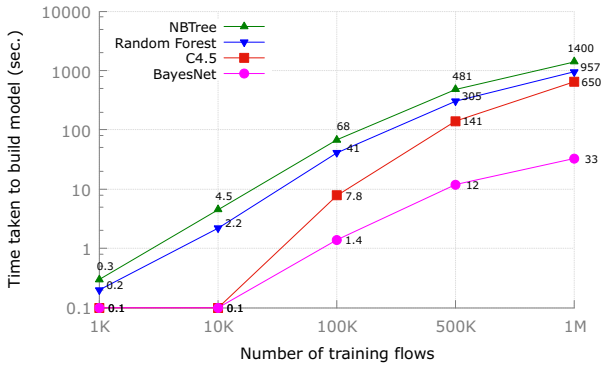


Fig. 7: Comparison of training speed of the algorithms.

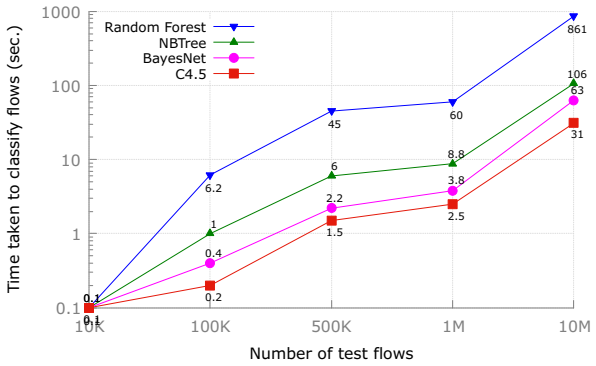


Fig. 8: Comparison of classification speed of the algorithms.

of our system by using datasets from two totally different networks. We trained the Random Forest algorithm using traffic flows from the Campus dataset and used this model to classify flows from the Residential dataset and vice versa. 500K instances each were used for both training and testing set. The accuracy in both cases was around 92%, which is 4% lower than in the scenarios where traffic from the same network dataset is used for both training and testing. This is probably due to usage of different services in these networks. The traffic in the campus network will naturally have more educational and scientific content which might not be there in a residential network. On the other hand, the residential network traffic usually has more entertainment-related content. As the different usage profile is not captured by the model, it may be the reason for the increase in misclassifications. However, the accuracy is still good enough for most scenarios and shows that this method can be extremely useful in cases where labeled datasets are not available for a particular network. In such cases a small labeled training set from another network can be conveniently used.

E. Computational performance

We calculated the time required for training and classification by varying the size of the training set from 1K to 1M entries and the size of the testing set from 10K to 10M. The experiments were performed on a PC with 2.5GHz Intel Core i7 processor and 8GB RAM. Figure 7 and Figure 8 show the training time and classification time. BayesNet is the fastest algorithm in training and takes only 12s for 500K instances. It is followed by C4.5, Random Forest and NB Tree that take 141s, 305s and 481s respectively. C4.5 is the fastest classification algorithm and takes only 31s to classify 10 million flows ($\approx$ 323,000 classifications/s). It is followed by BayesNet and NB Tree that take 63s and 106s respectively. Random Forest is the slowest algorithm with 861s.

These results show that Random Forest and C4.5 algorithms are best suited for this particular classification problem. C4.5 can be used in online classification systems where high speed is required while Random Forest is a better choice for offline systems as it provides slightly higher accuracy.

VI. Conclusion

We presented a comparison of HTTP/1 and HTTP/2 traffic. We studied basic statistics usually exported by NetFlow-like measurement devices and found that differences in the protocols, likely coupled with client and server adaptations to exploit new HTTP/2 features, result in very distinct network fingerprints. This calls for a reappraisal of traffic models, as well as HTTP traffic simulation and benchmarking approaches, which we will pursue in future work.

Differences in network fingerprints motivated us to develop a classifier able to identify the HTTP version based on basic traffic features. The proposed method is lightweight and uses only features found in NetFlow data. We showed that decision trees are suitable for this problem, and once the model is built on a relatively small training dataset, it stays valid for several months, thus eliminating the need for frequent ground truth collection and retraining.

Our work is a step forward in the direction of understanding modern web traffic. It provides a methodology to identify HTTP/2 traffic in flow traces where no information about application layer protocols is available. We expect that our work will improve visibility into network traffic and help in analyzing the adoption of HTTP/2 from passive traces.

References

[1] B. Krishnamurthy and C. E. Wills, "Analyzing factors that influence end-to-end web performance," *Computer Networks*, vol. 33, no. 1, pp. 17–32, 2000.

[2] B. A. Mah, "An empirical model of http network traffic," in *INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., Proceedings IEEE*, vol. 2. IEEE, 1997, pp. 592–600.

[3] M. Varvello, K. Schomp, D. Naylor, J. Blackburn, A. Finamore, and K. Papagiannaki, "Is the web http/2 yet?" in *Proceedings of the 17th International Conference on Passive and Active Measurements (PAM 2016)*, 2016, pp. 218–232.

[4] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX," *Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 2037–2064, 2014.

[5] R. Sadre and B. R. Haverkort, "Changes in the web from 2000 to 2007," in *International Workshop on Distributed Systems: Operations and Management*. Springer, 2008, pp. 136–148.

[6] S. Ihm and V. S. Pai, "Towards understanding modern web traffic," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 295–312.

[7] H. de Saxcé, I. Oprescu, and Y. Chen, "Is http/2 really faster than http/1.1?" in *2015 IEEE Conference on Computer Communications Workshops*, April 2015, pp. 293–299.

[8] J. Erman, V. Gopalakrishnan, R. Jana, and K. K. Ramakrishnan, "Towards a spdy'ier mobile web?" *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 2010–2023, 2015.

[9] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is spdy?" in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, 2014, pp. 387–399.

[10] Y. Elkhatib, G. Tyson, and M. Welzl, "Can spdy really make the web faster?" in *Networking Conference, 2014 IFIP*, June 2014, pp. 1–9.

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

[12] D. Schatzmann, W. Mühlbauer, T. Spyropoulos, and X. Dimitropoulos, "Digging into https: Flow-based classification of webmail traffic," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. ACM, 2010, pp. 322–327.

[13] A. Finamore, M. Mellia, M. Meo, M. M. Munafò, and D. Rossi, "Experiences of Internet Traffic Monitoring with Tstat," *IEEE Network*, vol. 25, no. 3, pp. 8–14, 2011.

[14] A. Langley, "Transport layer security (tls) next protocol negotiation extension," Working Draft, IETF Secretariat, Internet-Draft draft-agl-tls-nextprotoneg-03, April 2012. [Online]. Available: http://www.ietf.org/internet-drafts/draft-agl-tls-nextprotoneg-03.txt

[15] S. Friedl, A. Popov, A. Langley, and E. Stephan, "Transport layer security (tls) application-layer protocol negotiation extension," Internet Requests for Comments, RFC Editor, RFC 7301, July 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7301.txt

[16] Browser support tables. [Online]. Available: http://caniuse.com/

[17] Zero protocol by facebook. [Online]. Available: https://code.facebook.com/posts/608854979307125

[18] J. Manzoor, I. Drago, and R. Sadre, "The curious case of parallel connections in http/2," in *12th International Conference on Network and Service Management (CNSM)*, Oct 2016, pp. 174–180.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656278

[20] T. Mitchell, *Machine Learning*, 1st ed. New York: McGraw-Hill, 1997.

[21] V. Carela-Español, P. Barlet-Ros, O. Mula-Valls, and J. Solé-Pareta, "An autonomic traffic classification system for network operation and management," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 401–419, 2015.