Classifying HTTP/2 Traffic Using Machine Learning

Ze Ran Lu
School of Computer Science
University of Waterloo
zrlu@uwaterloo.ca

September 23, 2020

1 Motivation

In the last decades, many protocols have been developed to overcome the weaknesses of HTTP/1.1. These include, for instance, SPDY, QUIC and especially HTTP/2 which becomes more popular since its standardization in 2015. However, HTTP/1.1 is still widely used because of its long existence. Hence, we are interested in classifying HTTP traffics.

It has been found that most HTTP/2 traffic are transmitted over a secure channel such as TLS and it is extremely rare that HTTP/2 is non-encrypted [1]. Due to encryption, it is not possible to look into the payload of the packets. While the TLS handshake fields provide hints about the application protocol (which will be used for build the ground truth later), they are not available in the network flow statistics such as NetFlow, which is commonly used in real deployments[2]. Therefore, we need to extract features of network flows and learn about these features using machine learning algorithms and the machine learning method must not be computationally expensive.

2 Related works

2.1 Building the ground truth

Manzoor *et al.* [1] used the Application-Layer Protocol Negotiation (ALPN) field in TLS handshake as a hint of the HTTP version, which can be used to build the ground truth for the training set. ALPN is a two-step protocol.

The following experiment on UNB's VPN 2016 dataset [3] using Wireshark shows the two-step protocol. First, the client sends a list of supported protocol in the TLS `ClientHello` message to the server. We observe that the message contains the following fields:

```
Extension: application_layer_protocol_negotiation (len=35)
```

```
Type: application_layer_protocol_negotiation (16)
Length: 35
ALPN Extension Length: 33
ALPN Protocol
    ALPN string length: 5
    ALPN Next Protocol: h2-15
...
    ALPN string length: 2
    ALPN Next Protocol: h2
    ALPN string length: 8
    ALPN Next Protocol: spdy/3.1
    ALPN string length: 8
    ALPN Next Protocol: http/1.1
```

Note that the client also accepts protocols with pattern `h2-*`. This means the client accept draft versions of HTTP/2 for compatibility reasons. Then, the server responds with a *single* protocol name in the `ServerHello` message:

```
Extension: application_layer_protocol_negotiation (len=8)
    Type: application_layer_protocol_negotiation (16)
    Length: 8
    ALPN Extension Length: 6
    ALPN Protocol
        ALPN string length: 5
        ALPN Next Protocol: h2-15
```

It suffices to use the following Wireshark query to list all the `ServerHello`-type messages which select the HTTP/2 protocol:

```
tls.handshake.type == 2 &&
tls.handshake.extensions_alpn_str contains "h2"
```

Therefore, if a flow contains a `ServerHello` listed above, we can immediately label the flow as HTTP/2. This method also works on HTTP/1.1, SPDY, etc.

2.2 Machine learning models

Lightweight machine learning algorithms like random forest and Bayesian networks are used in [1] because of their fast classification speed and uses only features in NetFlow data. These features include: number of client bytes, number of server bytes, number of client packets, number of server packets, average client bytes per packet, average server bytes per packet and flow duration [1].

3 Data preparation

3.1 Packet capturing

It is known that most of the HTTP/2 traffic uses HTTPS [1]. Therefore, we focus on the traffic through the port 443. In order to generate sufficient HTTPS traffic, we

use Selenium WebDriver to simulate Web browsing by human. Selenium WebDriver allows scripted control of browsers such as Chrome and Firefox and can be executed in parallel and headlessly. We choose Selenium WebDriver because it allows files such as images and CSS stylesheets to be loaded, whereas Web crawlers only request HTML files. Hence, the simulated traffic is more "natural".

The data collection system is designed as follows: We initialize a queue of 500 URLs of popular websites. We have 10 instances of browsers that consumes the URLs in the queue. Once the page is fully loaded, for each hyperlink with HTTPS scheme, we add the URL to queue, then visits the next URL. A browser is given a maximum timeout of 30 seconds to wait for the page to be fully loaded. To simulate heavy traffic and obtain as much traffic as possible, we allow 10 instances of Selenium WebDrivers to run concurrently. To avoid redundant traffic in the same website, we maintain the number of occurrences of each network location. For example, if the number of "www.youtube.com" is seen more than 1000 times, we no longer add the URL to the queue.

We use Wireshark to dump packets through port 443. After 24 hours of data collection, we obtain 11.6 GB of .pcapng files.

3.2 Flow extraction and labeling

We created a Python script based on *dpkt* to extract flow information from .pcapng files. *dpkt* is a Python library which can parse .pcapng file into structured Ethernet frames, TCP segment and TLS records.

For simplicity, we ignore packets such as SYN, SYNACK and ACK which occurs before TLS handshakes. We create a `Flow` object whenever a `ClientHello` is received. The flow object is identified by a 4-tuple: the source IP address, the destination IP address, the source port number and the destination port number. We don't use the 5-tuple because we are only interested in TLS. The 4-tuple allows us to reconstruct flows from packets. Whenever a `ServerHello` message is received, we use the method described in Section 2.1 to label the `Flow` by the selected application layer protocol. Unlike in the VPN 2016 dataset [3], we did not find any protocol other than HTTP/1.1 and HTTP/2 (non draft version) in the `ServerHello` messages.

There exists some unlabelled flows, which suggest that a `ServerHello` message has never been received or the `ServerHello` message does not contain a valid ALPN information. These flows are simply excluded from our dataset.

We successfully extracted 7965 flows from the dumped traffic. Each flow contains the following field: the source IP address, the destination IP address, the source port number, the destination port number, the number of client packets, the total client bytes, the average client packet size in bytes, the number of server packets, the total server bytes, the average server packet size, the flow duration and the application layer protocol label. From the 7965 flows, 4869 are HTTP/1.1 and 3076 are HTTP/2.

	HTTP/1.1		HTTP/2	
	Mean	SD	Mean	SD
Client packet count	7.37	67.21	7.94	16.37
Total client bytes	2394.49	9062.83	1952.57	4936.97
Avg. client pkt. size	457.87	156.12	275.60	90.30
Server packet count	6.86	63.54	12.18	46.13
Total server bytes	1945.72	9315.55	8483.50	51744.40
Avg. server pkt. size	325.41	224.50	289.97	276.63
Flow duration	115.02	1135.30	71.37	175.30

Table 1: Mean and standard derivation of each feature per class in the dataset

3.3 Data analysis

We performed a simple analysis of the dataset. Table 1 shows the mean and standard derivation for each chosen feature in the dataset. Note that the mean average client packet size differs significantly between two classes and both have relatively small standard derivation. As per Figure 1 shows, that the average client packet size could be an important hint for flow classification.

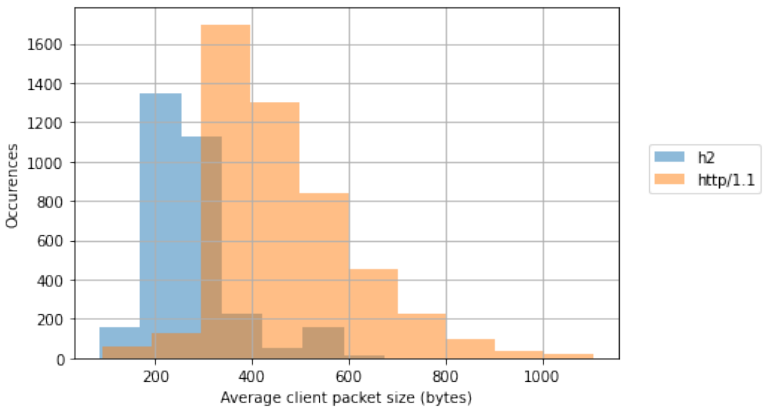
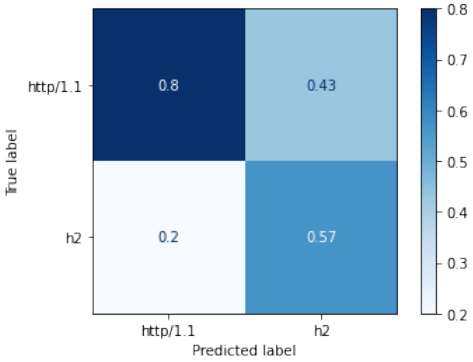


Figure 1: Average client packet size distribution per class

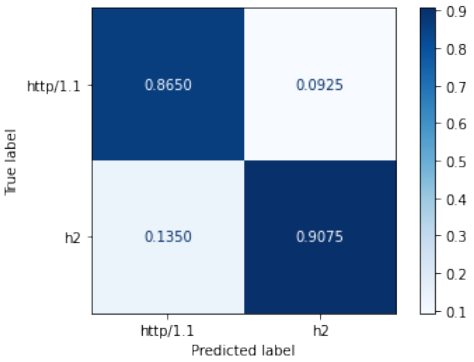
3.4 Train and test sets preparation

Because the number of HTTP/2 instances is smaller than the number of HTTP/1.1 instances, we augmented the dataset by sampling from the HTTP/2 instances with replacement so that both classes have the same number of instances. To prepare our training and test sets, we shuffled the augmented dataset and made 33 percent of the instances the test set.

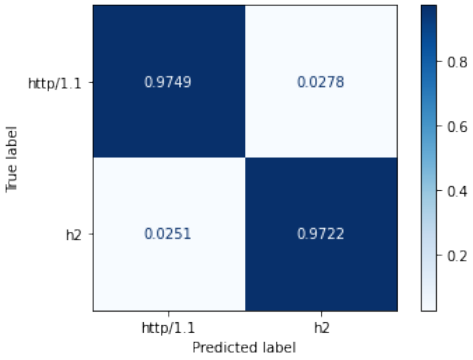
4 Results and discussions



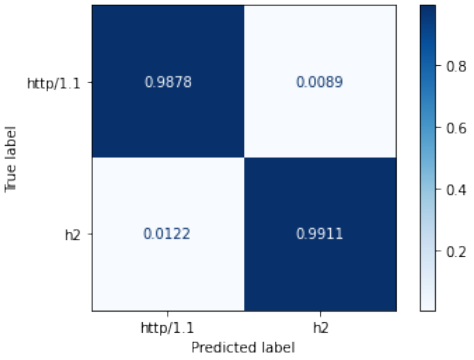
(a) baseline: Naive Bayes



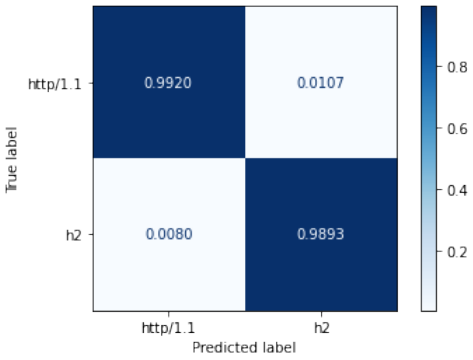
(b) Decision Tree (CART), leaves=4



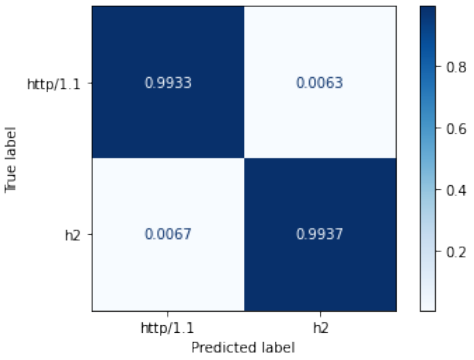
(c) Decision Tree (CART), leaves=16



(d) Decision Tree (CART), leaves=101



(e) Random Forest, classifiers=15



(f) AdaBoost, classifier=15

Figure 2: Normalized confusion matrices of our classifiers

We focus on decision tree and random forest, which is an ensemble of decision trees. In Manzoor *et al.* [1], the random forest model achieved the best result. We did not implement the Bayesian network because Manzoor *et al.* [1] already showed that its accuracy is lower than decision tree. In addition, we implemented naive Bayes classifier as the baseline and AdaBoost classifier with decision trees as improved version of the ensemble classifier to see if the accuracy can be increased.

Figure 2 shows the result of the classifiers in the form of normalized confusion matrices. Note that the naive Bayes classifier tends to give a lot of false positives (43% of the predicted HTTP/2 are wrong).

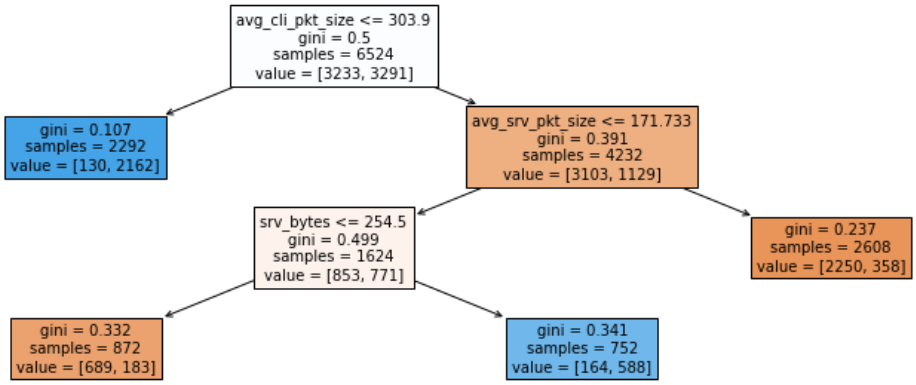


Figure 3: Our 4-leaf decision tree

4.1 Decision tree classifier

For decision tree, instead of the C4.5 algorithm used in Manzoor *et al.* [1], we use CART to generate the decision tree as it is *sklearn*'s default splitting algorithm. Figure 2 (b), 2(c) shows the performance of decision tree of different number of leaves (4 and 16). We intentionally limit the number of leaves so that the tree splitting algorithm ends early and we want to see their performance. Without the limit, the algorithm will generate an optimal number of leaves, which is 101, and its performance is shown in Figure 2(d). Interestingly, we found that even a simple decision tree of 4 leaves can achieve high accuracy. The 4-leaf tree can correctly classify 86.5% of HTTP/1.1 flows and 90.75% of HTTP/2 flows.

Figure 3 visualizes this 4-leaf tree. Due to readability, the 16-leaf tree and 101-leaf tree are not presented. In fact, the 4-leaf tree is simply a trimmed tree of decision trees with higher number of leaves. This helps us understand how which feature is used to perform the initial best split. Each tree node is filled with color. White color means both classes have the same number. Blue color means the majority of flows are HTTP/2 and orange color means the majority of flows are HTTP/1.1.

Note that the first feature selected for splitting is the average client packet size. This is expected because in Section 3.3 we showed that the mean of both classes differs

significantly (HTTP/1.1 has a mean 457 bytes and HTTP/2 has mean 275.6 bytes) and the variance is relatively small. Possible explanation is the header compression in HTTP/2 [1]. The second feature selected is the average server packet size and the third is the total server bytes.

4.1.1 Random Forest

A random forest is an ensemble of decision trees trained from sub-samples with replacement. This process is called bootstrapping. Random forest can be trained in parallel. For a classification problem, each decision trees outputs its prediction and the majority wins. Figure 2(e) shows the accuracy of random forest. Here, we use 15 classifiers.

4.1.2 AdaBoost

AdaBoost assigns weights to each its component classifiers. "Good" component classifier is weighted more. Like the random forest model we trained, we also use 15 decision tree as base classifiers in our experiment. AdaBoost generally gives better accuracy. Compared to random forest, the training time is longer and cannot be parallelized. Figure 2(f) shows the accuracy of AdaBoost, which has the highest accuracy among our classifiers.

4.2 Classification speed

Since the programming language used for implementation is different, we do not compare the classification speed with [1]. Table 2 shows the time needed to classify 100K flows for each trained classifier on Google Colab. Note that the three decision trees roughly require the same amount of time and faster than the naive Bayes classifier. The largest tree with 101 leaves requires slightly more time due to the height of the tree. However, the random forest and AdaBoost are significantly slower. This is because 15 classifiers need to be evaluated sequentially. However, for the ensemble classifiers, it is possible to achieve better classification speed using parallelization.

Trained classifier	Time (s)
Naive Bayes	10.87
Decision tree (4 leaves)	5.84
Decision tree (16 leaves)	5.79
Decision tree (101 leaves)	6.27
Random forest (15 classifiers)	143.44
AdaBoost (15 classifiers)	173.03

Table 2: Time needed to classify 100K flows for each trained classifier

Classifier	Precision	Recall	Accuracy
Ours			
Naive Bayes	0.568	0.915	0.616
Decision tree (4 leaves)	0.907	0.851	0.885
Decision tree (16 leaves)	0.974	0.972	0.974
Decision tree (101 leaves)	0.991	0.987	0.989
Random forest (15 classifiers)	0.989	0.991	0.991
AdaBoost (15 classifiers)	0.994	0.993	0.993
Manzoor *et al.* [1]			
Random forest	0.962 [1]	- [2]	0.975 [3]
C4.5	0.932	-	0.95
NB tree	0.886	-	0.93-0.95
Bayes net	0.857	-	0.86-0.88

Table 3: Comparing precision, recall and accuracy with [1]

4.3 Performance comparison

Finally, we compare the performance of our trained models with [1]. Note that our classification is binary (HTTP/1.1 and HTTP/2), whereas in [1] there are 3 classes (HTTP/1, HTTP/2 and others). We define HTTP/2 to be positive in both cases. In Table 3, we compare our models with [1] using metrics such as precision, recall and accuracy. Note that some metrics are missing in [1]. Our random forest and AdaBoost classifiers slightly outperform their best model, which is random forest classifier. In general, our decision tree with more than 16 leaves performs as good as their C4.5 decision tree. Also, our training set is quite different from theirs because their dataset contains 500K instances.

5 Conclusions

In conclusion, we showed how to extract annotated flow information from packet dump files and implemented decision tree and ensemble classifiers and compared their performance. We also analyzed our dataset and discovered that the best feature to separate our classes is the average client packet size. The decision tree model can accurately predict the application protocol in a flow. Although our ensemble classifiers can achieve higher accuracy, they are slower in classifying flows sequentially, but the speed can be improved using parallel computation.

[1] The precision measures are extracted from the confusion matrices for the Campus dataset only.
[2] The recall measures are missing because they only show normalized confusion matrices.
[3] The accuracy measures are obtained from cross-validation using Residential and Campus datasets.

References

[1] J. Manzoor, I. Drago, and R. Sadre. How http/2 is changing web traffic and how to detect it. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–9, June 2017. doi: 10.23919/TMA.2017.8002899.

[2] 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. *IEEE Communications Surveys Tutorials*, 16(4):2037–2064, 2014.

[3] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. Characterization of encrypted and vpn traffic using time-related features. In *ICISSP*, 2016.