HSLF: HTTP Header Sequence based LSH fingerprints for Application Traffic Classification

Zixian Tang[1,2], Qiang Wang[1,2], Wenhao Li[1,2], Huaifeng Bao[1,2], Feng Liu[1,2], and Wen Wang[1,2]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, CAS,
Beijing, China

[2] School of Cyber Security, University of Chinese Academy of Sciences,
Beijing, China

{tangzixian,wangqiang3113,liwenhao,baohuaifeng,liufeng,wangwen}@iie.ac.cn

Abstract. Distinguishing the prosperous network application is a challenging task in network management that has been extensively studied for many years. Unfortunately, previous work on HTTP traffic classification rely heavily on prior knowledge with coarse grained thus are limited in detecting the evolution of new emerging application and network behaviors. In this paper, we propose HSLF, a hierarchical system that employs application fingerprint to classify HTTP traffic. Specifically, we employ local-sensitive hashing algorithm to obtain the importance of each field in HTTP header, from which a rational weight allocation scheme and fingerprint of each HTTP session are generated. Then, similarities of fingerprints among each application are calculated to classify the unknown HTTP traffic. Performance on a real-world dataset of HSLF achieves an accuracy of 96.6%, which outperforms classic machine learning methods and state-of-the-art models.

Keywords: HTTP header fields · Application traffic classification · Local-sensitive hashing · Traffic fingerprinting

1 Introduction

Network traffic classification is an important task in network management. With the vigorous development of information technology, numerous applications flush into terminal devices, which results in the increasingly complicated network traffic. In order to manage network more effectively, a powerful network traffic classification solution needs to be implemented. Although HTTPS has been popularized in recent years, we found that HTTP is still one of the most mainstream network protocols. Many applications implement client-server communications based on HTTP, so that the importance of the HTTP protocol has not diminished.

HTTP headers play a very important role in application traffic. The headers are flexible to design according to developers' needs. For example, in a POST request, Content-Type is often designed to indicate the form of payload submitted

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

by the client. The server will perform different processing strategies for various types of data according to the value of Content-Type. The design of header fields have their specific meanings which directly reflects the developer's thinking in communication module development. Many researchers have conducted in-depth research on HTTP header fields. However, their research on header fields are one-sided that only focuses on limited fields rather than delving into all of them.

In this paper, we propose HSLF, a system that transforms HTTP traffic into fingerprint and classifies application traffic. HSLF generates an HTTP Header Sequence (HHS) based on HTTP headers. The sequence contains all header fields. For each HHS, we perform a Local-sensitive hashing algorithm to convert each sequence into a simple fingerprint. In this way, the sessions sent by various applications are saved as fingerprints with ground-truth labels. Through a large collection of traffic fingerprints, we can gradually describe the features of an application's HTTP traffic, and store fingerprints in a massive database that is continuously updated. We implement application identification by measuring the similarities between labeled fingerprints and fingerprints to be identified. The experimental results indicate that, HSLF outperforms classic machine learning models and two state-of-the-arts in four metrics.

In general, our main contributions can be concluded as follows:

- We comprehensively analyze most of HTTP headers and rationally assessed their importance based on machine learning algorithm.
- We build a dynamically updated fingerprinting database based on LSH algorithm which can be applied for measuring the similarities between HTTP sessions.
- To the best of our knowledge, we are among the first to propose a method that includes all HTTP headers. Evaluated on real-world datasets, HSLF achieves eye-catching performance and is superior to the state-of-the-arts.

2 Background and Related Work

2.1 Traffic Classification Solutions

Many previous traffic classification methods have some limitations that prevent them from being useful in real-time environment. These methods can be divided into rule-based matching and machine learning-based methods. Rule matching methods are mainly based on TCP quintuples (Source IP, Destination IP, Source port, Destination port, protocol) [5] or payload. TCP quintuples based methods may fail for some applications that use ephemeral port allocation or CDN policies. Crotti et al. [3] proposed a flow classification mechanism based on statistical properties of the captured IP packets. However, statistical features based methods are difficult to classify applications accurately. Mainstream payload based methods like Deep Packet Inspection (DPI) [10] and signature seeking are not only infringe on the privacy of end users, but also unsuitable for real-time traffic analysis due to the high resource overhead. Another popular methods

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

are machine learning based models, they train their classifier based on some statistical features in application-generated traffic [15] [19] [12]. However, due to the limitations of machine learning, it cannot adapt to the growing traffic scale. These methods are far behind network management requirements. In addition to the limitations of the method itself, many traditional classification methods are coarse-grained that include limited categories (e.g. email, news, and games) [8] [11]. There are also many methods using fingerprinting algorithm [22] [18] [4] for traffic classification, and they are proved successful in real environments.

2.2 Research on HTTP headers

Many researchers have conducted in-depth research on HTTP header fields. Kien Pham et al. [16] studied the $UserAgent$ and identified web crawler traffic based on this field. Fei Xu et al. [20] discovered the *ContentType* inconsistency phenomenon and used the inconsistency to detect malwares. Arturs et al. [9] and William J. Buchanan1 et al. [1] investigated the use of security headers on the Alexa top 1 million websites to conduct security assessments.

Xu et al. [21] proposed a method that was using the identifier contained in the $UserAgent$ of the HTTP header to distinguish mobile applications. Yao et al. [22] proposed SAMPLES, which automatically collect conjunctive rule corresponds to the lexical context, associated with an application identifier found in a snippet of the HTTP header. However, previous researches pay attention to limited header fields, and few works can adapt to the real environment which contains complex combinations of HTTP header fields.

3 Preprocessing

In this section, we introduce some preprocessing steps. We investigate HTTP headers to determine the weight of each header field. Then we introduce HTTP Header Sequence (HHS), which is used to create a standardized template for an HTTP session.

3.1 Weight Assignment

Different headers contribute differently in application identification. For example, the field $Content-Type$ reflects the MIME type of the subsequent payload, that is, the data processing format, which directly reflects the data type passed in the application; while the field *Date* represents the time at which the message was originated. It indicates the creation time of the message. Obviously, the contribution of $Content-Type$ to traffic classification is much greater than that of *Date*.

In order to maximize the value of each field, we design the weighting algorithm of the HTTP headers. We perform machine learning methods to train fields

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

to study the influence of headers. Some machine learning models have mechanisms for scoring features which makes them easier to be applied to feature selection tasks. We select 10 typical application including browsing, chat, video etc. By one-hot encoding and standardizing all the header fields, the headers can be learned by the machine learning model.

The machine learning model we choose is Random Forest (RF). RF consists of multiple decision trees. Each node in the tree is a condition about a certain feature, in order to divide the data into two according to different response variables. The node can be determined by the impurity (optimal condition). For classification purposes, Gini Impurity or information divergence is usually used. When training a decision tree, we can calculate how much the tree's impurity is reduced by each feature. For a decision tree forest, it is possible to figure out the average reduction of each feature which can be regarded as the value of feature importance.

The feature score we choose is Gini Impurity (Gini Index). Select a header field F and count the sum of the Gini index (GI) decline degree (or impurity decline degree) of the branch nodes formed by t in each tree of RF. It is defined as:

$$GI(t) = 1 - \sum_{k=1}^{K} p_{k|t}^2 \tag{1}$$

k indicates that there are k categories, and $p_{k|t}$ indicates the proportion of the category k in the node t.

The importance of field F_j at node t ($I_{j|t}$), that is, the amount of Gini index change before and after the node T branch is computed as:

$$I_{j|t} = GI(t) - GI(l) - GI(r) \tag{2}$$

where $GI(l)$ and $GI(r)$ respectively represent the index of F_j on the left and right node. Then, add up the importance of F_i in the whole forest (m trees in) as:

$$I_j = \sum_{i=1}^{m} \sum_{t \in T} I_{j|t} \tag{3}$$

Finally, we normalize all I_j to get each field's final importance (i_j) score. The normalization process is computed as:

$$i_j = \frac{I_j}{\sum_{i=1}^{n} I_i} \tag{4}$$

Importance i indicates the field F's contribution to application classification. We calculate i of all the fields for weight assignment. The results are shown in Table 1. We treat the importance of headers as weights.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

Table 1. Importance Ranking list of HTTP headers (Top 20)

Rank	Header	Importance	Rank	Header	Importance
1	User-Agent	0.10407	11	Accept-Language	0.04164
2	Server	0.09036	12	Accept-Encoding	0.0337
3	Path	0.0864	13	Upgrade	0.02585
4	Connection	0.08349	14	Pragma	0.02216
5	Cache-Control	0.06304	15	Date	0.02192
6	Content-Type	0.06136	16	Expires	0.02158
7	Host	0.05998	17	Method	0.01954
8	Accept	0.05816	18	Referer	0.01689
9	Content-Length	0.05671	19	Status-Line	0.01645
10	Cookie	0.04616	20	Last-Modified	0.01463

3.2 HTTP Header Sequence

We define two types of headers: character headers and numeric headers. The character headers are mainly composed of character strings, while a numeric header contains a definite value. We propose different approaches to deal with these two type of headers. For numeric headers, we keep the value directly. For a character header, if the form of characters in this field is relatively simple, such as a Server header "nginx", we retain the original character strings. If the character field is more complicated, such as the *Host* field which will appear with multiple directory separators, we further split the string according to the delimiter. Taking "pan.baidu.com" for example, we split it into "pan", "baidu", "com". Such segmentation is conducive to extracting as many features of long characters as possible.

Meanwhile, there may be multiple request and response exchanges in a session, each header may appear more than once. In most cases, values of a header are constant. However, there are also cases where values are different and cannot be ignored. So we choose to generate a first draft of this session $S = \{F_1 : k_1; F_2 : k_2; ...F_n : k_n\}$ based on the first request and response pair. If the k'_i in the subsequent data is not the same as the k_i in S, then we append k'_i to F_i to update S as $S = \{f_1 : k_1; f_2 : k_2; ...f_i : k_i, k'_i; ...f_n : k_n\}$. Then we allocate the weights achieved in Section 3 to each F to form a complete sequence (HHS) for the HTTP session $S = \{F_1(w_1) : k_1; F_2(w_2) : k_2; ...F_i(w_i) : k_i, k'_i; ...F_n(w_n) : k_n\}$.

4 HTTP Header Sequence based LSH Fingerprints

HSLF is a hierarchical system shown in Fig. 1. In this section, we will present HSLF in detail. We first introduce the proposed fingerprinting algorithm developed from SimHash [2] to generate the fingerprint of HHS. Based on it, we establish a fingerprint database which enable the system to identify applications in HTTP traffic.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

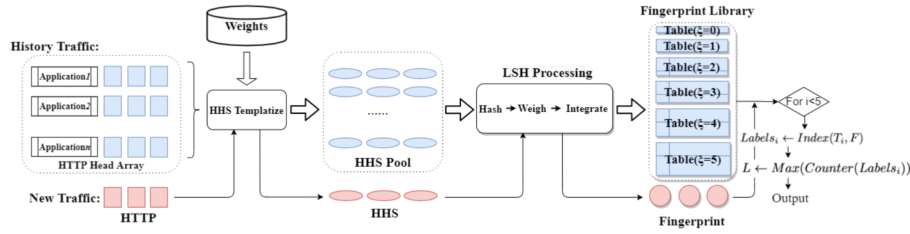


Fig. 1. The overview of HSLF framework.

4.1 LSH Fingerprint

Although we have made predictions through the RF model in Section 3 and obtained some successful predictions, general machine learning algorithms cannot adapt to large-scale data classification. In real environments, there are far more than 10 categories. If we use machine learning algorithms for modeling and prediction, we need to learn the characteristics of each application, in which case the consumption of resources will multiply and the increased categories will also decrease the accuracy of model predictions. When we increased the number of applications from 10 to 50, the classification accuracy dropped from 95.8% to 87.2%. Obviously, such accuracy is far from satisfactory for application identification. In addition, it is also difficult to standardize the headers of the string type when modeling machine learning. The efficiency of standardization also directly affects the prediction performance. Therefore, we propose to employ fingerprinting algorithm to describe application traffic.

Local-sensitive hashing (LSH) [6] is an important method with solid theoretical basis for measuring text similarity. It is also widely used in the nearest neighbor search algorithm. The traditional Hash algorithm is responsible for mapping the original content as uniformly and randomly into a signature, which is equivalent to a pseudo-random number generation algorithm in principle. The traditional hash algorithm no longer provides any information except that the original content is not same. The two signatures generated by it may be very different, even if the original content differs by only one byte. Therefore, the traditional hash cannot measure the similarity of the original content in the dimension of the signature. However, the hash signature generated by LSH can represent the similarity of the original content to a certain extent. Its main application is to mine similar data from massive data, which can be specifically applied to text similarity detection, web search and other fields. SimHash is one of LSH algorithms proposed by Charikar [2], and it has been widely used in data retrieval and near-duplicates detection. It maps high-dimensional feature vectors to fixed-length binary bit strings.

The HHS is essentially a high-dimensional vector composed of natural language. Each sequence represents a network communication of an application. In order to identify applications on HTTP traffic, we propose the HHS based

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

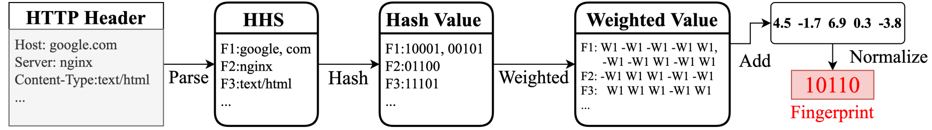


Fig. 2. Example of fingerprint generating.

SimHash algorithm to generate a 64-bit fingerprint for each HTTP session. The details of fingerprinting are as follows.

Templatizing Templatizing module is to transform an HTTP session into HHS like $\{F_1(w_1) : k_1; F_2(w_2) : k_2; ...; F_i(w_i) : k_i, k_i'; ...; F_n(w_n) : k_n\}$, and saves them in the pool.

Hash In order to get a fingerprint that can be used to summarize the session and save the characteristics of the session, we utilize the hash algorithm independently on each k_i. Then we add weights and merge these irrelevant hash values to process the dimensionality reduction into the fingerprint format for normal storage. The hash algorithm we choose is MD5 algorithm. The first 64 bits of MD5 are preserved as the result of a hash operation. Since the MD5 algorithm is a classic algorithm, we will not describe it in detail here. k_i's hash result is $h_i = Hash_{MD5}(k_i)$. As we have highly divided the string in the foregoing, the local hash algorithm will not have an avalanche effect on the entire sequence.

Integrate After calculating the hash for each k_i, we perform a bit-wise accumulation on all h_i: if a bit of $h_{i,j}$ is 0, add the weight value $-w_i$, if it is 1, add the weight value w_i, and finally get the value on each bit weighted value v_i. v_i is computed by:

$$v_i = \sum_{i=1}^{n} (-1)^{1-h_{i,j}} w_i \tag{5}$$

The s_i of each digit on the final fingerprint is accumulated by all v_i bits and judged according to positive and negative, it is defined as follows:

$$s_i = \begin{cases} 0, & v_i < 0 \\ 1, & v_i \geq 0 \end{cases} \tag{6}$$

After completing the above steps, we get a fingerprint of 64 bits. The Fig. 2 shows an example flow chart of fingerprinting. For one HTTP session, the first step is to extract headers and generate an HHS. Then, hash each fields' strings and assign weights. Finally, merge all bits with addition to get the fingerprint represents the session (we use a 5-bit value like 10110 for easy explanation).

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

4.2 Fingerprint Database

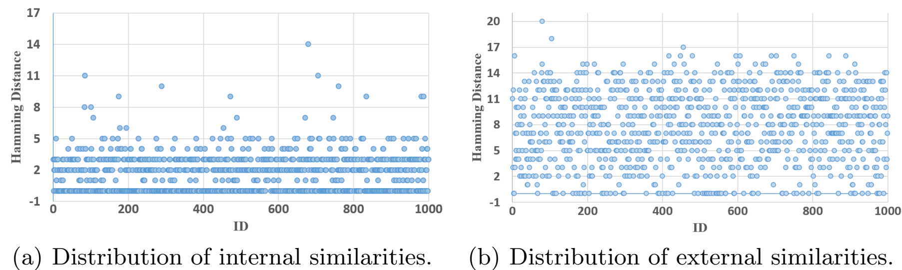
We process each HTTP session into a 64-bit fingerprint and measure the similarity between fingerprints by the Hamming distance. Hamming distance represents the number of different bits in two (same length) words. Charikar [2] proposed that when the Hamming distance is less than 3, the text can be considered similar. Our purpose here is not to select all similar texts, but to select the closest one.

First, we need to determine the threshold of the Hamming distance. HSLF and SimHash are substantively different that HHS is not a text composed of natural language. Therefore, we cannot directly use the threshold 3 of natural language text as our threshold for judging similarity. To determine the threshold, we randomly selected 1000 fingerprints and calculated the minimum value of similarities between one of 1000 and all other fingerprints under the same application, which we call internal similarities; and the minimum value of similarities between the one and other fingerprints of different applications, which we call external similarities. The distribution of internal similarities and external similarities is shown in the Fig. 3(a) and Fig. 3(b).

It can be found that most internal similarities are between 0-3, and the internal similarities between 0-5 account for 98.2%; the distribution of external similarities is more scattered that similarities greater than 5 account for 68.7%. So in order to locate a new session to its original application, we consider both the distributions of internal similarities and external similarities and set the threshold to 5. Fingerprints with a similarity of 5 and below are listed as optional.

Gurmeet et al. [14] realized near-duplicate detection at the scale of 8 billion web pages, they found that the problem was how to quickly find fingerprints with a Hamming distance of less than 3, so they proposed an algorithm to speed up the index and successfully used in Google's detection module. Although the amount of fingerprints we achieve is far less large, but if we compare fingerprints one by one to find the most similar fingerprint, the resource overhead is also huge. So we build our method of fingerprint storage and indexing on the basis of Gurmeet's method, which win the time at the cost of space.

In order to facilitate indexing, we divide all fingerprint libraries into multiple levels and store them into 6 tables $\{T_0, T_1, ..., T_5\}$ according to Hamming distance



(a) Distribution of internal similarities. (b) Distribution of external similarities.

Fig. 3. The Similarity distribution of LSH fingerprints

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

from 0 to 5. The table with Hamming distance 0 is the usual Hash table, a theoretical index structure with a time complexity of $O(1)$. Tables with Hamming distance greater than or equal to 1 are stored as follows:

Copy the entire fingerprint database with the order of 2^d into t sub-tables, each sub-table has an integer p_i and a permutation π_i, the sub-tables $Z_1, Z_2 ... Z_t$ are constructed by π_i that permutes the p_i bits of all fingerprints in the sub-table to top bits. Then, the fingerprint F to be identified is changed by $\pi_i(F)$. $\pi_i(F)$ will be matched with top p_i bits in each Z_i in parallel. If the fingerprints are all random sequences, there are probably 2^{d-p_i} fingerprints left. Finally, find out fingerprints whose Hamming distance with F are ξ_i. p_i and t are determined according to the actual situation. If p_i is too small, the number of matching fingerprints 2^{d-p_i} is too large, and the search volume is still very high. If p_i is too large, the quantity of sub-tables t will grow a lot.

4.3 Application Identification

The application identification method consists of application fingerprint database pre-work and real-time traffic stream identification. This section describes the algorithm. For a session to be identified, HHS function is used to extract HHS with different fields. $LSHFingerprint$ function calculate session's fingerprint.

The Algorithm 1 describes how an unknown session is identified from a set of requests. The input is fingerprint tables (work as indexable memory data structure) and a new session s. Fingerprint tables $\{T_0, T_1, ..., T_5\}$ are labeled with known application names. Each capture session s will be transformed into an HHS and fingerprinted to get F. Then algorithm queries F from the fingerprint tables T_{ξ_i} in the order of ξ_i from 0 to 5: if a similar fingerprint of F is indexed when $\xi_i = 0$, HSLF will return the application label of the matched fingerprint and end the loop immediately, otherwise continue to find fingerprint with ξ_i self-increasing until ξ_i equal to 5 (the max of pre-work). If multiple fingerprints with different application labels are hit, the return value will be the max number of accumulate labels. If the fingerprint to be predicted does not match any application when $x_i \in [0, 5]$, we will treat such a fingerprint as a new fingerprint and store it in the unknown set for future re-predict. As the size of the fingerprint database continues to increase, fingerprints in the unknown set will be gradually identified.

5 Experiments

In this section, we evaluate our proposed system on real-world dataset to verify the rationality and effectiveness of the system. First, we introduce the public dataset used in the experiments. Then, we detail specific aspects of our approach such as the performance of multi-classification and the superiority compared to machine learning algorithms. Finally, we compare HSLF against some state-of-the-art methods.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

Algorithm 1: Application identification

Input: Fingerprint tables $T_0, T_1, ..., T_5$ where $\xi_i \in [0, 5]$,
A new session s to be identified.
Output: Application identification result L.

```
1  S ← HHS(s) F ← LSHFingerprint(S) i ← 0
2  while i ≤ 5 do
3  |  Labels_i ← Index(T_i, F)
4  |  num ← len(Counter(Labels_i))
5  |  if num == 0 then
6  |  |  i ← i + 1
7  |  else if num == 1 then
8  |  |  return L ← Counter(Labels_i)
9  |  else if num > 1 then
10 |  |  return L ← Max(Counter(Labels_i))
11 end
```

5.1 Dataset

We collect our original traffic on the lab gateway by using the process log tool PPfilter developed ourselves to mark the application traffic. PPfilter collects network and process log information at the terminal to form a log with a 5-tuple including: IP (source IP and destination IP), port (source port and destination port), port open time, process number (Corresponds to this port), and application-related process name. We use PPfilter to collect their daily traffic for 15 volunteers on campus for one month, and filter out the traffic of each application in the total traffic. During the filtering process, we perform anonymization and privacy protection by replacing or deleting the IP address and some payloads. All collected traffic is marked with application labels. In total, ProcFlow contains 72GB PCAP files from 35 applications.

5.2 Multi-class Classification Performance

We conduct 10-fold cross validation on ProcFlow and separately verify the performance of HSLF on 10 typical applications. The classification result is shown in Fig. 4. It can be found that each application's three evaluation metrics Precision, Recall, F1-score are satisfactory, and the averages (AVE) are all higher than 0.9. Some applications such as Thunder and Xunfeng can be classified absolutely right.

5.3 Compared with classic Machine Learning Models

Many previous studies employed machine learning (ML) models for application recognition and proved effective. To prove the superiority of HSLF, we compare it with 4 classic machine learning models: C4.5 (C4.5 decision tree) [7],

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

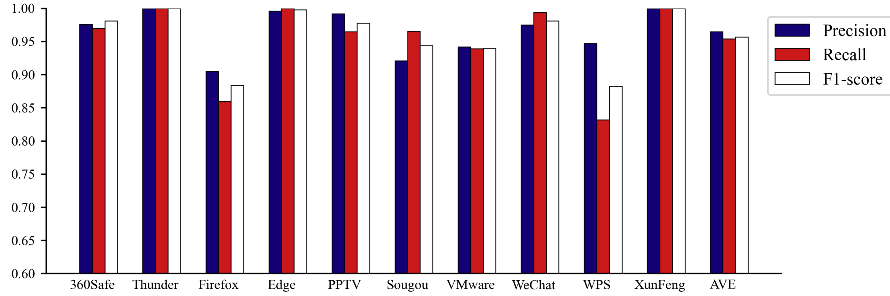


Fig. 4. Precision, recall, F1-score of HSLF on 10 popular Apps.

RF (Random Forests) [17], SVM (Support Vector Machines) [13], kNN (K-NearestNeighbor) [23]. The input of ML models are the one-hot encodings of the headers. To achieve a fair result, all the parameters of machine learning models have been adjusted to the best. We also conduct a 10-fold cross validation on ProcFlow2020 dataset for each model.

The classification results are shown in Table 2. It can be found that the four evaluation metrics of HSLF are: 0.966, 0.926, 0.901, 0.916. Although the accuracy of the four machine learning models are all higher than 85%, the precision, recall, and F1-score are much lower compared to HSLF. It indicates that traditional machine learning classification is unstable, and may have slip in some categories. In contrast, HSLF stay strong performance (higher than 0.9) on precision, recall and F1-score. The comparision result shows that HSLF outperforms machine learning models.

Table 2. Comparison Results of HSLF and classic ML methods

Methods	ProcFlow2020			
	ACC	Precision	Recall	F1-Score
C4.5	0.8934	0.7075	0.6417	0.6527
Random Forest	0.8946	0.7486	0.6555	0.6806
kNN	0.8777	0.5936	0.5116	0.5310
SVM	0.8699	0.5658	0.4568	0.4848
HSLF	**0.9655**	**0.9257**	**0.9055**	**0.9155**

5.4 Comparison with other Approaches

In this section, we compare HSLF with two state-of-the-art application traffic classification methods to describe the superiority of HSLF.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

Table 3. Comparison Results of Miner-Killer and Other Methods

Methods	ProcFlow2020			
	ACC	Precision	Recall	F1-Score
TrafficAV	0.9020	0.9006	0.9020	0.8995
FlowPrint	0.5007	0.5232	0.5007	0.5063
HSLF	**0.9655**	**0.9257**	**0.9055**	**0.9155**

TrafficAV [18] is an effective malware identification and classification method. It extracts traffic features, and then uses detection models based on machine learning to judge whether the app is malicious or not. TrafficAV contains two detection models namely HTTP detection model and TCP flow detection model. We choose the HTTP detection model for comparison with HSLF. The authors of TrafficAV describe the depiction of the analysis steps and detection solution without open-source implement. Therefore, we faithfully reimplemented the HTTP detection model of TrafficAV feature extraction strategy, and build a classifier based on C4.5 decision tree algorithm the same as TrafficAV.

FlowPrint [4] is a semi-supervised approach for fingerprinting mobile apps from network traffic. It can find temporal correlations among destination-related features of network traffic and use these correlations to generate app fingerprints. FlowPrint calculates the Jaccard distance between two fingerprints and compares with threshold to measure the similarity. The application label of each traffic flow receives the same label as the fingerprint that is most similar to it.

Table 3 shows the identification performance of TrafficAV, FlowPrint and HSLF. It can be found that HSLF outperforms both of them on four metrics. Although the traffic classification purposes of TrafficAV and FlowPrint are different from ours, there are similarities in our solutions. TrafficAV focus on some key information in the HTTP headers, however, TrafficAV only selects 4 headers which ignores the valid information that may be contained in the other headers. In an environment with numerous applications, it is likely to cause collision of feature sequences. Our method HSLF learns from natural language processing to treat the entire headers of the HTTP session as a sequence, and generates a fingerprint of the session to mark the application according to a reasonable weight assignment method. It does not miss any header features and does not rely on specific identifiers. Compared with TrafficAV and FlowPrint, HSLF has better robustness.

6 Conclusion

In this paper, we propose HSLF, a system transforms HTTP sessions into fingerprints for applications traffic classification. HSLF retains almost all the features of HTTP headers and templatizes the headers into HTTP Header Sequence with rational weights. With the help of LSH algorithm, HSLF processes high-dimensional vectors into fingerprints and stores them as tables of different levels.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

HTTP based applications traffic can be classified by fast index. Experimental results show that HSLF have a satisfactory classification performance and outperforms some classic machine learning methods. Compared with some state-of-the-art approaches, HSLF also has a big improvement. In the future, we will optimize HSLF to fit more network protocols, such as encrypted protocols SSL / TLS.

7 Acknowledgment

This work was supported by the National Key R&D Program of China with No. 2018YFC0806900 and No. 2018YFB0805004, Beijing Municipal Science & Technology Commission with Project No. Z191100007119009, NSFC No.61902397, NSFC No. U2003111 and NSFC No. 61871378.

References

1. Buchanan, W.J., Helme, S., Woodward, A.: Analysis of the adoption of security headers in http. IET Information Security **12**(2), 118–126 (2017)
2. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. pp. 380–388 (2002)
3. Crotti, M., Dusi, M., Gringoli, F., Salgarelli, L.: Traffic classification through simple statistical fingerprinting. ACM SIGCOMM Computer Communication Review **37**(1), 5–16 (2007)
4. van Ede, T., Bortolameotti, R., Continella, A., Ren, J., Dubois, D.J., Lindorfer, M., Choffnes, D., van Steen, M., Peter, A.: Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In: Network and Distributed System Security Symposium, NDSS 2020. Internet Society (2020)
5. Fraleigh, C., Moon, S., Lyles, B., Cotton, C., Khan, M., Moll, D., Rockell, R., Seely, T., Diot, S.C.: Packet-level traffic measurements from the sprint ip backbone. IEEE network **17**(6), 6–16 (2003)
6. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613 (1998)
7. Jie, Y., Lun, Y., Yang, H., Chen, L.y.: Timely traffic identification on p2p streaming media. The Journal of China Universities of Posts and Telecommunications **19**(2), 67–73 (2012)
8. Kaoprakhon, S., Visoottiviseth, V.: Classification of audio and video traffic over http protocol. In: 2009 9th International Symposium on Communications and Information Technology. pp. 1534–1539. IEEE (2009)
9. Lavrenovs, A., Melón, F.J.R.: Http security headers analysis of top one million websites. In: 2018 10th International Conference on Cyber Conflict (CyCon). pp. 345–370. IEEE (2018)
10. Li, Y., Li, J.: Multiclassifier: A combination of dpi and ml for application-layer classification in sdn. In: The 2014 2nd International Conference on Systems and Informatics (ICSAI 2014). pp. 682–686. IEEE (2014)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5

11. Li, Z., Yuan, R., Guan, X.: Accurate classification of the internet traffic based on the svm method. In: 2007 IEEE International Conference on Communications. pp. 1373–1378. IEEE (2007)
12. Liu, C., He, L., Xiong, G., Cao, Z., Li, Z.: Fs-net: A flow sequence network for encrypted traffic classification. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications. pp. 1171–1179. IEEE (2019)
13. Liu, C.C., Chang, Y., Tseng, C.W., Yang, Y.T., Lai, M.S., Chou, L.D.: Svm-based classification mechanism and its application in sdn networks. In: 2018 10th International Conference on Communication Software and Networks (ICCSN). pp. 45–49. IEEE (2018)
14. Manku, G.S., Jain, A., Das Sarma, A.: Detecting near-duplicates for web crawling. In: Proceedings of the 16th international conference on World Wide Web. pp. 141–150 (2007)
15. Moore, A., Zuev, D., Crogan, M.: Discriminators for use in flow-based classification. Tech. rep. (2013)
16. Pham, K., Santos, A., Freire, J.: Understanding website behavior based on user agent. In: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. pp. 1053–1056 (2016)
17. Raghuramu, A., Pathak, P.H., Zang, H., Han, J., Liu, C., Chuah, C.N.: Uncovering the footprints of malicious traffic in wireless/mobile networks. Computer Communications **95**, 95–107 (2016)
18. Wang, S., Chen, Z., Zhang, L., Yan, Q., Yang, B., Peng, L., Jia, Z.: Trafficav: An effective and explainable detection of mobile malware behavior using network traffic. In: 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS). pp. 1–6. IEEE (2016)
19. Williams, N., Zander, S.: Evaluating machine learning algorithms for automated network application identification (2006)
20. Xu, F., Pan, H., Cao, Z., Li, Z., Xiong, G., Guan, Y., Yiu, S.M.: Identifying malware with http content type inconsistency via header-payload comparison. In: 2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC). pp. 1–7. IEEE (2017)
21. Xu, Q., Erman, J., Gerber, A., Mao, Z., Pang, J., Venkataraman, S.: Identifying diverse usage behaviors of smartphone apps. In: Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference. pp. 329–344 (2011)
22. Yao, H., Ranjan, G., Tongaonkar, A., Liao, Y., Mao, Z.M.: Samples: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic. In: Proceedings of the 21st Annual International Conference on Mobile Computing and Networking. pp. 439–451 (2015)
23. Zhang, J., Xiang, Y., Zhou, W., Wang, Y.: Unsupervised traffic classification using flow statistical properties and ip packet payload. Journal of Computer and System Sciences **79**(5), 573–585 (2013)

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77961-0_5