Mobile Application Traffic Identification Method Based on Visual Perception Features (Postprint)

Authors: Li Ding, Zhu Yuefei, Lin Wei

Date: 2018-05-02T00:00:00+00:00

Abstract

Since most mobile applications communicate via the HTTP protocol, traditional port-based identification methods have essentially become ineffective. Furthermore, both deep packet inspection and machine learning approaches based on flow statistical features face challenges in manual feature engineering and labeled sample acquisition. Leveraging the strengths of the computer vision domain, we propose a mobile application traffic identification method based on visual perception features. First, application-layer payload data is transformed into visually meaningful images, and real-world data is collected from network gateways to construct the sample dataset IMTD17. Next, we design a convolutional perception network model 2D-CPN with visual feature extraction capabilities, which employs convolutional autoencoders to learn from large quantities of unlabeled samples and establishes a mapping from hidden-layer features to application types through multi-class regression. Experimental results indicate that the traffic identification accuracy of this method satisfies the requirements for practical deployment.

Full Text

Preamble

Mobile App Traffic Identification Based on Visual Perception Features

Li Ding, Zhu Yuefei, Lin Wei
(State Key Laboratory of Mathematical Engineering & Advanced Computing, Zhengzhou 450001, China)

Abstract: Since most mobile applications communicate via HTTP protocol, traditional port-based identification methods have become largely ineffective. Furthermore, both deep packet inspection and flow-statistics-based machine

learning approaches face difficulties in manual feature engineering and sample labeling. Drawing inspiration from the successes of computer vision, this paper proposes a mobile app traffic identification method based on visual perception features. First, we convert application-layer payload data into visually meaningful images and collect real data from network gateways to establish the sample dataset IMTD17. Then, we design a convolutional perception network model 2D-CPN with visual feature extraction capabilities, which leverages convolutional autoencoders to learn from large amounts of unlabeled samples and establishes a mapping from latent features to application types through multi-class regression. Experimental results demonstrate that the traffic identification accuracy of our method meets practical requirements.

Keywords: mobile app; traffic identification; convolutional autoencoder; latent feature

0 Introduction

Traffic identification—the process of mapping network traffic to network protocols or generating applications—plays a crucial role in network supervision and malicious app detection. Traditional traffic identification primarily targeted desktop applications such as commercial software and malware. Since these typically used specific ports or proprietary protocols with characteristic fields in their headers, matching protocol header fields like application-layer port numbers could achieve satisfactory identification results.

Unlike desktop applications, most mobile apps communicate with servers through the unified HTTP application-layer protocol [1,2], rendering traditional port-based identification methods largely ineffective. To identify mobile app traffic, some researchers have attempted to find fingerprint information that can uniquely identify applications. However, mobile traffic may not contain such special characteristic fields, making it difficult to find effective fingerprints and keep pace with mobile app updates—an arduous and time-consuming task. In addition to feature extraction difficulties, another challenge facing traffic identification is the lack of effective datasets [3]. Due to the massive scale of mobile app traffic in network data and the continuous updates and emergence of mobile apps, it is difficult to ensure the timeliness and effectiveness of labeled traffic datasets. Therefore, how to leverage the large amount of unlabeled data in networks is key to the development of traffic identification technology.

To address these two challenges, this paper proposes a deep learning method for traffic identification based on visual perception features. First, borrowing ideas from image recognition in deep learning, we convert mobile app traffic into visually meaningful images, enabling machines to analyze traffic similarly to how network analysts use Wireshark. Then, through the two-dimensional visual perception network 2D-CPN, we achieve feature extraction from large amounts of unlabeled network traffic data. Finally, through small-scale supervised learning with multi-class regression, we achieve accurate identification of application traf-

fic. Additionally, to test model performance and address the dataset scarcity problem, we have established the mobile app traffic sample dataset IMTD17.

1 Related Work

Numerous studies have addressed mobile app traffic identification. The most direct approach matches special fields in HTTP protocols. Xu et al. [4] first used the User-Agent field to identify Android apps, but this field is not mandatory for app identification, and mobile app developers do not always follow the convention of adding User-Agent. Dai et al. [5] also considered the request-line URL and host fields, building corresponding fingerprints for different applications. Miskovic et al. [6] proposed an automated feature extraction system called AppPrint that comprehensively extracts URLs and various header field features from HTTP protocols, such as User-Agent and Cookies, but this method' s effectiveness depends on manually specified identifiers. References [7,8] attempted to aggregate HTTP traffic from mobile apps, arguing that a single user operation on a mobile phone generates multiple HTTP flows (e.g., browsing web pages, opening images or videos), and that the correlation among these data flows is mainly reflected in packet lengths and intervals between adjacent packets. Although many mature methods exist for identifying mobile app traffic, they all rely on manually designed features or feature discovery methods. In such cases, machines cannot surpass the identification boundaries defined by humans to automatically discover hidden unknown features in traffic data.

In recent years, with the rapid development of computer vision and other fields, some traffic identification techniques based on deep learning have emerged. Wang [9] first used artificial neural networks (ANN) to learn and classify TCP payload data. They converted the first 1,024 bytes of labeled TCP sessions into one-dimensional vectors as ANN model inputs, and after training found that the most important bytes were concentrated at the beginning. This method' s limitation is its reliance on differences in application-layer protocol headers, making it difficult to distinguish application traffic based on the same application-layer protocol, such as HTTP used by mobile apps. Reference [10] borrowed computer vision methods, using convolutional neural networks (CNN) to classify malware traffic. They first converted the first 784 bytes of data into 28×28 two-dimensional images, then used CNN for supervised classification learning. When selecting input data, this method considered the difference between TCP flows and sessions, and through experiments found that including transport layer, network layer, and data link layer header data achieved the best classification results. However, since the formats of these header data are defined by RFC documents with fixed and explicit field meanings, precise traffic identification can be achieved by matching specific protocol header fields (e.g., TCP protocol ports, flags, etc.). Therefore, this data extraction method actually leads to information loss. Moreover, convolutional neural networks belong to supervised learning methods and thus face the same challenge of difficulty in manually labeling large amounts of network data samples.

2 Methods and Models

This chapter proposes a network traffic data image conversion method based on visual perception features, establishes the mobile app traffic sample image dataset IMTD17 based on this method, and then presents the two-dimensional convolutional perception network model 2D-CPN for traffic identification, detailing the model derivation process.

2.1 Image Conversion Method

A key reason for the excellent results in computer vision is that images are first visually recognizable and meaningful, which then guides machines to learn more images following human thinking patterns. However, network data itself has structured characteristics, with relatively fixed data attributes and logical relationships between fields. If network data is directly filled into images, these structural characteristics are broken, making them difficult to visually recognize and contradicting the essence of computer vision. Based on this motivation, we attempt to format network traffic data (but not parse it) and then convert it into visual images.

According to RFC 2616 specifications, the HTTP protocol consists of a start line, several header fields, and a body. Let R represent the request line, S the status line, H the header fields, B the body, and C the carriage return line feed (CRLF). Then an HTTP session composed of a request (Req) and response (Res) can be described as:

$$\begin{aligned}\text{Req} &= R \parallel C \parallel H_1 \parallel C \parallel H_2 \parallel C \parallel \cdots \parallel H_a \parallel C \parallel C \parallel B_{\text{Req}} \\ \text{Res} &= S \parallel C \parallel H_1 \parallel C \parallel H_2 \parallel C \parallel \cdots \parallel H_b \parallel C \parallel C \parallel B_{\text{Res}}\end{aligned}$$

where $\parallel$ is the byte concatenation operator; a and b represent the number of header fields in the corresponding messages; and $C \parallel C$ represents the separation between header and body. To extract effective HTTP protocol fields, we analyze the effectiveness of each component:

a) **Request line R**: The first line of the request message, consisting of method, URL, and version, where the URL is related to specific application requests.

b) **Status line S**: The first line of the response message, containing only HTTP status information and thus not application-related.

c) **Header fields H**: Starting from the second line of HTTP request and response messages, their types, quantities, and order are uncertain. Through extensive analysis, we found that in HTTP session data generated by the same mobile app communication, the number and order of H are basically fixed. The intuitive explanation is: for the mobile app side, as long as it is a publicly released platform version, its program attributes are

determined by developers, making H features fixed; for the server side, although H features are somewhat correlated with the server platform, they are mainly determined by the server configuration of application developers, and services generally need to maintain backward compatibility, so configurations are not frequently updated nor new features added.

d) **Body B**: The last part of request and response messages, with diverse data types and optional. When its content is control information, it has some connection with application behavior; when its content is resource data, it can be considered random and thus not closely related to the application.

Based on the above analysis, we select R and H related to the application as effective data. Additionally, to preserve visually distinguishable contour information, we retain the line feed character C. Therefore, the final extracted effective data is:

$$\text{Image} = Q(P(R \parallel C \parallel H_1 \parallel C \parallel H_2 \parallel C \parallel \cdots \parallel H_a \parallel C), P(R \parallel C \parallel H_1 \parallel C \parallel H_2 \parallel C \parallel \cdots \parallel H_b \parallel C))$$

In the extracted data, the number of lines and byte length per line are uncertain. Since computer vision accepts images of fixed size, we need to normalize the extracted data by padding or discarding some data to convert it into sample images suitable as model input. Assuming the target image size to be converted is $M \times N$, to obtain a row of image data pixels, we define the row operation P as:

$$P(d_{1:n}) = \begin{cases} O_{1:n} = d_{1:n}, & n < N \\ O_{1:n} = d_{1:N}, & n \geq N \end{cases}$$

where $d_{1:n}$ represents a data fragment of length n bytes; $O_{1:n}$ represents k NUL character connections. To obtain a complete image, we define the column operation Q as:

$$Q(e_{1:m}) = \begin{cases} O_{1:m} = e_{1:m}, & m < M \\ O_{1:m} = e_{1:M}, & m \geq M \end{cases}$$

where $e_{1:m}$ represents two-dimensional image data with m rows. Using the above equations, the extracted effective data can be converted into model input images. $\emptyset$ is data of length 0. To preserve visual features as much as possible, we experimented with various image sizes and determined $M = 28$, $N = 36$. [Figure 1: see original paper] shows sample images of mobile app traffic, with four samples randomly selected from communications of each app. We can see that different traffic sample images from the same app have high consistency,

while traffic sample images from different apps have high differentiation. Therefore, using images obtained through this conversion method as input for traffic identification models is reasonable.

Based on the image conversion method, we established the IMTD17 dataset. First, we obtained traffic data from campus network gateways, where mobile app traffic was generated by smartphones connected via Wi-Fi. Then, we extracted TCP payloads based on quadruples (source IP, source port, destination IP, destination port) to obtain raw HTTP sessions. IMTD17 consists of three parts, as shown in . The first part is used for the first-stage (S1) unsupervised training, consisting of a large number of unlabeled samples including a small number of mobile app traffic sample images (APP) and a large number of unrelated background traffic sample images (BG). The second part is used for the second-stage (S2) supervised classification training, comprising a small number of manually labeled mobile app traffic samples extracted from the first part, including 12 types of Android app traffic samples labeled 0~11, as shown in . The third part is used for model testing (T), consisting of labeled samples with composition similar to the first part, where only the 12 categories corresponding to the second part have true labels, and the remaining background traffic samples have label value 12.

2.2 Convolutional Perception Network Model

For the mobile app traffic images converted in Section 2.1, we designed a two-dimensional convolutional perception network model (2D-CPN) adapted to the image size, as shown in [Figure 2: see original paper]. This model is based on the convolutional autoencoder algorithm [11,12]. Compared with other autoencoder algorithms, convolutional autoencoders can extract local two-dimensional features from raw inputs and achieve global weight sharing, rather than treating all features as global features. Convolution operations preserve the local correlation of input image data, converting spatial location information into high-order feature representations. Then, using the nonlinear fitting capability of multi-layer perceptrons (MLP), these high-order features are converted into low-dimensional latent features. Since these latent features have distinguishable distance relationships in low-dimensional space, they can be mapped to corresponding types through Softmax multi-class regression, thereby achieving sample type identification.

In the 2D-CPN model, the recognizer first converts the original 28×36 image sample x into high-order features $\hat{x}$. Through nonlinear fitting of the MLP-encoder network, high-order features $\hat{x}$ are converted into latent features z. During the sample reconstruction phase, latent features z are converted to corresponding high-order features $\hat{x}$ through the MLP-decoder network, and then the generator converts high-order features $\hat{x}$ into the reconstructed image sample $\tilde{x}$. In this process, the convolution group operation (conv) in the recognizer is responsible for decomposing feature maps, while the deconvolution group operation (deconv) in the generator is responsible for recomposing feature maps.

The computational functions and specific parameters involved in these operations are shown in [Figure 3: see original paper]. To reconstruct image samples, convolution operations should not attenuate original image information. However, basic convolution operations neglect edge features of images, causing the output feature map size to be reduced compared to the input image according to the convolution kernel size. Therefore, appropriate padding operations need to be added before basic convolution operations.

For given horizontal and vertical parameters (s_x, s_y), the subsampling function reduces data in both directions simultaneously, resulting in an output feature map size of $(H/s_y, W/s_x)$. Through the above derivation, the feature map size produced by one convolution group operation in the recognizer of [Figure 2: see original paper] is $(H - s_y(k_y - 1), W - s_x(k_x - 1))$.

The generator network in [Figure 2: see original paper] converts the original image x into high-order features $\hat{x}$ with good representation. To achieve classification and identification, MLP is used for nonlinear fitting of high-order features to obtain a mapping from high-order features to latent features z. For the MLP encoder and decoder networks in [Figure 2: see original paper], the l-th layer can be represented as:

$$h_l = \phi(W_l h_{l-1} + b_l)$$

$$\hat{h}_l = \phi(\hat{W}_l \hat{h}_{l+1} + \hat{b}_l)$$

where W and b are perceptron node weights and biases; ϕ is the activation function for each layer, where we use softplus to better map high-order features uniformly into the latent space [16]. Assuming the number of MLP layers is p (typically $p = 2$ to avoid overfitting [17]), the latent feature z is obtained through the MLP encoder network:

$$z = h_p = W_p h_{p-1} + b_p$$

When performing classification and identification, the model maps latent features z to certain output categories through multi-class regression softmax function. The output values of softmax layer nodes represent the confidence that the original input sample belongs to the corresponding type:

$$\hat{y}_j = \frac{e^{z_j}}{\sum_i e^{z_i}}$$

where $\hat{y}$ is the output vector matrix of softmax layer nodes. Notably, most traffic data in real networks consists of unknown-category background traffic. However, the type results output by the softmax layer have clear boundary limitations,

i.e., $\dim(y)$. If classification is performed based solely on output confidence values, many incorrect classification results will be produced. Therefore, we add threshold conditions to the softmax layer output results, where a sample is considered to belong to a corresponding type only when its confidence value exceeds the corresponding threshold.

3 Experiments and Analysis

This chapter conducts experiments and analysis on the training and testing processes of the 2D-CPN model. First, we verify the model' s ability to extract sample latent features, then use test datasets to evaluate the model' s traffic identification performance.

3.1 Latent Feature Visualization

To demonstrate that classification and identification can be achieved using latent features, thereby identifying the applications generating the traffic, we visualize latent features in two-dimensional space. Specifically, we first reduce the latent space dimension to $\dim(z) = 2$, mapping latent features to points on a plane. Then we retrain the reconstruction model in [FIGURE:4(c)] to establish a mapping from input image samples x to two-dimensional latent features z. After training, we use test dataset T containing both app traffic and large amounts of background traffic as input to the reconstruction model to obtain latent features z. Finally, we correlate the label values y of samples in dataset T (from) with latent features z, using the two-dimensional values of latent features z as coordinates on the plane to plot the positions of app traffic and background traffic samples, as shown in [Figure 5: see original paper]. For clarity, different app traffic sample points are represented by different shapes, while background traffic sample points are represented by small circles.

The model' s training and testing processes are shown in [Figure 4: see original paper]. Before formal training, the two pre-training processes in [FIGURE:4(a)] and (b) can help extract latent features z more quickly and accurately. The formal training process consists of two stages: unsupervised reconstruction model (reconstructor) training in [FIGURE:4(c)] and supervised classification model (categorizer) training in [FIGURE:4(d)]. The former stage is similar to infants observing the objective world—they can distinguish different categories of things through observation but don' t know what specific things are. The latter stage is equivalent to teaching infants what the things they distinguish actually are.

In the unsupervised training stage, the reconstruction model extracts latent features z from the unlabeled dataset S1. Before training begins, convolution kernels and MLP weights are initialized using the Xavier method to make the variance of outputs from each layer as equal as possible [18]. To obtain good parameters through backpropagation, the reconstruction model uses cross-entropy as the loss function during training:

$$C = -\sum_i [x_i \log \tilde{x}_i + (1 - x_i) \log(1 - \tilde{x}_i)]$$

Meanwhile, convolution kernels, MLP weights, and bias values are corrected through backpropagation. In the supervised training stage, the classification model establishes a mapping from latent features z to specific categories through the labeled dataset S2. During training, the classification model uses multi-class cross-entropy loss function:

$$C = -\sum_i [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

where y is the one-hot encoded type vector matrix and $\hat{y}$ is the output vector matrix of the softmax layer.

From the figure, we can see that latent features of the same class of samples are very concentrated, indicating high uniformity of latent features extracted by the model. Latent features of different classes have sufficient distinguishing distances, indicating high discriminability of latent features extracted by the model. Additionally, background traffic samples are uniformly distributed near the coordinate origin, with some concentrated sample points that likely belong to the same traffic category. Therefore, this method can help people discover unknown categories of application traffic. Through two-dimensional visualization analysis, we can reasonably infer the representation capability of latent features: as the dimension of latent space increases, the spatial distance between latent features of different class samples further expands, thereby enhancing the sample representation capability of latent features.

The traffic identification process in [FIGURE:4(e)] tests the model' s identification performance using test dataset T based on the trained categorizer. The classification model' s identification results are evaluated using three metrics: accuracy (Acc), precision (Pre), and recall (Rec):

$$\mathrm{Acc} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \times 100\%$$

$$\mathrm{Pre} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \times 100\%$$

$$\mathrm{Rec} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \times 100\%$$

where TP and FP represent correct and incorrect classification numbers; P and N represent positive and negative instance numbers, corresponding to mobile app traffic sample numbers and unknown background traffic sample numbers

in this paper. Among these metrics, accuracy reflects the overall classification performance of the model; precision concerns the prediction results, indicating how many of the predicted positive samples are truly positive; recall concerns the original samples, indicating how many positive samples in the sample set are correctly predicted.

When thresholds for each category reach optimal values, traffic identification results achieve optimal performance, as shown in . Although the recognition accuracy and precision are high, recall is relatively low due to the presence of some confusing background traffic.

TABLE:2 Model Identification Results on Dataset T

Metric	Acc	Pre	Rec
Value	99.5	100	95.4

To identify which specific background traffic affects identification results for further research, we list the identification results for each category of mobile app (in this case, negative sample number N represents background traffic sample numbers plus unrelated mobile app traffic sample numbers), as shown in . Only individual apps have relatively low recall, such as QQ Mail and WeChat, indicating that some background traffic images are very similar to these app traffic images and have entered the identification boundaries of these app traffic sample latent features. However, valuable guidance can be obtained from these erroneous identification results: on one hand, they can help people check whether manual labeling is correct; on the other hand, this method can automatically discover unknown traffic types, helping people achieve more intelligent traffic identification.

TABLE:3 Mobile App Type Labels and Identification Results

Label	App	Pre	Rec
0	Alipay	100	100
1	Baidu	100	100
2	QQ Mail	100	81.9
3	Bilibili	100	91.2
4	CNTV	100	95.7
5	Kugou	100	100
6	Taobao	100	100
7	Weibo	100	100
8	QQ Music	100	100
9	WeChat	100	89.3
10	QQ	100	100
11	Youku	100	100

4 Conclusion

This paper proposes a mobile app traffic identification method based on visual perception features. Drawing on the advantages of computer vision, we convert raw application-layer payload data into visually meaningful images. Simultaneously, using the visual feature extraction capability of convolutional autoencoder algorithms, we preserve the local correlation of input images and achieve global sharing of feature weights. Finally, we utilize the convolutional perception network to achieve automatic learning from large amounts of unlabeled samples and traffic identification under supervision. Results show that the model has good adaptability to traffic samples, with identification precision meeting practical usage requirements.

References

[1] Falaki H, Lymberopoulos D, Mahajan R, et al. A first look at traffic on smartphones [C]// Proc of ACM SIGCOMM Conference on Internet Measurement. 2010: 281-287.

[2] Lee S W, Park J S, Lee H S, et al. A study on smart-phone traffic analysis [C]// Proc of IEEE Network Operations and Management Symposium. 2011: 1-5.

[3] Dainotti A, Pescape A, Claffy K C. Issues and future directions in traffic classification [J]. Network IEEE, 2012, 26 (1): 35-40.

[4] Xu Q, Erman J, Gerber A, et al. Identifying diverse usage behaviors of smartphone apps [C]// Proc of ACM SIGCOMM Conference on Internet Measurement Conference. 2011: 329-344.

[5] Dai Shuaifu, Tongaonkar A, Wang Xiaoyin, et al. NetworkProfiler: towards automatic fingerprinting of Android apps [C]//Proc of the 32nd IEEE Conference on Compute Communications. 2005: 809-817.

[6] Miskovic S, Lee G M, Liao Y, et al. AppPrint: automatic fingerprinting of mobile applications in network traffic [M]// Passive and Active Measurement. Springer International Publishing. 2015: 57-69.

[7] Mongkolluksamee S, Visoottiviseth V, Fukuda K. Enhancing performance of mobile traffic identification with communication patterns [C]// Proc of IEEE Computer Software and Applications Conference. 2015: 163-168.

[8] Su X, Zhang D, Dai S, et al. Mobile traffic identification based on application' s network signature [J]. International Journal of Embedded Systems, 2016, 8 (2//3): 217.

[9] Wang Z. The applications of deep learning on traffic identification [EB/OL]. (2015) [2017-10-19]. http://www.blackhat.com/docs/us-15/materials/us-15-Wang-The-Applications-Of-Deep-Learning-On-Traffic-Identification-wp.pdf.

[10] Wang W, Zhu M, Zeng X, et al. Malware traffic classification using convolutional neural network for representation learning [C]// Proc of IEEE International Conference on Information Networking. 2017: 712-717.

[11] Masci J, Meier U. Stacked convolutional auto-encoders for hierarchical feature extraction [C]// Proc of International Conference on Artificial Neural Networks. [S. l.]: Springer-Verlag, 2011: 52-59.

[12] Zeiler M D, Taylor G W, Fergus R. Adaptive deconvolutional networks for mid and high level feature learning [C]// Proc of International Conference on Computer Vision. [S. l.]: IEEE Computer Society, 2011: 2018-2025.

[13] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks [C]// Proc of International Conference on Neural Information Processing Systems. [S. l.]: Curran Associates Inc. 2012: 1097-1105.

[14] Zhao J, Mathieu M, Goroshin R, et al. Stacked what-where auto-encoders [J]. Computer Science, 2015, 15 (1): 3563-3593.

[15] Dosovitskiy A, Springenberg J, Tatarchenko M, et al. Learning to generate chairs, tables and cars with convolutional networks [J]. IEEE Trans on Pattern Analysis & Machine Intelligence, 2017, 39 (4): 692.

[16] Glorot X, Bordes A, Bengio Y. Deep sparse rectifier neural networks [C]// Proc of International Conference on Artificial Intelligence and Statistics. 2012: 315-323.

[17] Lawrence S, Giles C L. Overfitting and neural networks: conjugate gradient and backpropagation [C]// Proc of Ieee-Inns-Enns International Joint Conference on Neural Networks. 2002: 114-119 vol. 1.

[18] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks [J]. Journal of Machine Learning Research, 2010, 9: 249-256.

Note: Figure translations are in progress. See original paper for figures.