Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 4707909, 14 pages
https://doi.org/10.1155/2020/4707909

Research Article

Real Network Traffic Collection and Deep Learning for Mobile App Identification

Xin Wang[1], **Shuhui Chen**[1] **and Jinshu Su**[1,2]

[1]*School of Computer, National University of Defense Technology, Changsha, Hunan 410073, China*
[2]*National Key Laboratory of Parallel and Distributed Processing (PDL), National University of Defense Technology, Changsha, Hunan 410073, China*

Correspondence should be addressed to Xin Wang; wangxin09@nudt.edu.cn

Received 15 June 2019; Revised 2 January 2020; Accepted 23 January 2020; Published 19 February 2020

Academic Editor: Marco Picone

The proliferation of mobile devices over recent years has led to a dramatic increase in mobile traffic. Demand for enabling accurate mobile app identification is coming as it is an essential step to improve a multitude of network services: accounting, security monitoring, traffic forecasting, and quality-of-service. However, traditional traffic classification techniques do not work well for mobile traffic. Besides, multiple machine learning solutions developed in this field are severely restricted by their handcrafted features as well as unreliable datasets. In this paper, we propose a framework for real network traffic collection and labeling in a scalable way. A dedicated Android traffic capture tool is developed to build datasets with perfect ground truth. Using our established dataset, we make an empirical exploration on deep learning methods for the task of mobile app identification, which can automate the feature engineering process in an end-to-end fashion. We introduce three of the most representative deep learning models and design and evaluate our dedicated classifiers, namely, a SDAE, a 1D CNN, and a bidirectional LSTM network, respectively. In comparison with two other baseline solutions, our CNN and RNN models with raw traffic inputs are capable of achieving state-of-the-art results regardless of TLS encryption. Specifically, the 1D CNN classifier obtains the best performance with an accuracy of 91.8% and macroaverage F-measure of 90.1%. To further understand the trained model, sample-specific interpretations are performed, showing how it can automatically learn important and advanced features from the uppermost bytes of an app's raw flows.

1. Introduction

Traffic through a typical network is heterogeneous and consists of flows from multiple applications and utilities. Associating traffic flows with the applications that generate them is known as traffic classification (or traffic identification), which is an essential step to prioritize, protect, or prevent certain traffic [1]. With accurate and complete traffic classification, various network actions or services, such as accounting, monitoring, control, and optimization, can be performed with the end goal of improving network performance or security. In recent years, the increasingly growing mobile traffic due to the proliferation of mobile devices (mainly smartphones) has changed the characteristics of network traffic greatly. This trend is expected to create a nearly 4-fold increase in global mobile data over the next 4 years [2]. Thereupon, mobile traffic analysis is coming into focus along with the growing demand and difficulty to implement mobile app identification (APP-ID). In addition to the benefits for mobile operators, mobile app identification is also important for companies when bring-your-own-device (BYOD) policies are enabled to manage access to corporate resources [3]. Despite the fact that it may raise privacy risks, some groups like advertisers and security agencies are also interested in its potential for valuable profiling information.

The state of the art in traffic classification has experienced a major boost in the past decade. From port-based, deep packet inspection (DPI) to machine learning (ML) approaches, the technology is in continuous development to

keep up with the ever-evolving Internet. The requirements and challenges of APP-ID in mobile networks are even more formidable. Different from traditional desktop applications whose communication patterns are usually simple, mobile apps are hard to be identified by their protocols and port numbers. Typically, they offer multiple services with different protocols (e.g., HTTP/HTTPS) using common or arbitrary port numbers and rarely include unique signatures within the packet as recommended for identification. Additionally, many mobile apps use content delivery networks (CDNs) and third-party services (e.g., advertisement, analytics), making strategies like domain name resolution and IP address lookup unreliable [4].

As encryption technologies are gaining traction every day, more and more traffic is being transmitted over encrypted protocols (e.g., TLS) to avoid interception at the network level. This trend, however, makes DPI based on signatures no longer viable and ML techniques gain more and more attention from scientific research to commercial applications. Multiple conventional ML classifiers have proven successful in mobile as well as traditional traffic classification [5–8]. Nevertheless, they are subject to the manual process of feature engineering, which can be time-consuming and prone to obsolescence. Based on intuition and expert knowledge, feature engineering aims to find a representation of raw data which conveys characteristics that are most relevant to the learning problem. In fact, in many applications including APP-ID, it reveals even greater importance than the choice of the specific machine learning algorithm [5, 9]. In this work, we thus explore whether we can leverage deep learning to improve identification accuracy of mobile apps. Benefiting from the ability to process natural data in their raw form, deep learning (DL) can discover good features in an automatic way without human engineering ahead of time [10].

Last but not least, we are confronted by the usual obstacle to progress on APP-ID: lack of a variety of real-world mobile traffic to serve as train-test data as well as ground truth (i.e., annotated flow objects used as reference) for validation. Most of the previous works seem to neglect this quandary, which is quite important for training and testing ML or DL models [4–6]. Usually, they base their results on ground truth built from private datasets and labeled by means of unknown reliability. Therefore, a methodology that can efficiently construct a trusted real network dataset is in high demand.

This paper proposes a solution to the problem of constructing real-world mobile datasets and performs a complete evaluation of applying DL techniques to APP-ID. In the experiment results, we demonstrate that our DL-based APP-ID techniques can achieve state-of-the-art results in spite of TLS encryption. For a better understanding of our results, we also exploit state-of-the-art interpretation techniques to interpret the predictions of our best-performing DL model. The contributions of this paper are fourfold:

(i) We develop NetLog—a novel Android app tool that can capture all traffic from the smartphone and accurately label flows by app names.

(ii) We construct a large mobile traffic dataset that consists of 142 apps, including both considerable unencrypted and encrypted traffic flows, for classification evaluation. The dataset is of high richness and is easy to scale with accurate ground truth obtained using our dedicated tool.

(iii) We apply modern DL techniques to the task of APP-ID and evaluate three different DL classifiers on our dataset: Stacked Denoising Autoencoder (SDAE), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM) network. All three models are specially designed and optimized for our task. To the best of our knowledge, it is the first comprehensive DL evaluation practice in this field.

(iv) We use one of the state-of-the-art DL interpretation techniques to explain our model. The interpretations give insights into how a prediction can be achieved through deep learning from raw traffic, which can in turn guide the task of APP-ID.

The rest of this paper is structured as follows. Section 2 gives the related work on APP-ID as well as the use of DL models for traffic classification. In Section 3, we describe the methodology of our work including dataset collection approach and applied DL models. We give the evaluation results and discussion in Section 4 and model interpretation in Section 5. A conclusion to our work is presented at last in Section 6.

2. Related Work

There have been a plethora of works in the field of traffic classification research including papers representing attempts to classify various traffic samples with various techniques. Here, we give a brief overview of most recent achievements focused on APP-ID. Several efforts introducing DL to Internet traffic classification are also discussed.

2.1. Traffic Classification for APP-ID. The research on mobile app identification mainly falls into two categories under different assumptions: (1) unencrypted traffic and (2) encrypted traffic. In the first scenario, researchers focus on how to improve app coverage or flow coverage (i.e., identification covers as many apps or flows as possible) in addition to accuracy. Typically, they use DPI and analyze HTTP headers to extract app fingerprints, assuming that the app network traces are already given or generated from automatic UI exploration techniques [11–14]. Some other works use static analysis of apps to guide the signature discovery, enabling APP-ID in a massive scale [15, 16]. However, all previous cases do not deal with encrypted traffic about which the second category of research cares most. Our paper also takes it into consideration.

Previous work dealing with encrypted traffic mostly applies ML techniques. Wang et al. [17] propose a system for identifying smartphone apps from encrypted wireless traffic. They collect data from 13 arbitrarily chosen apps by running them dynamically and training a Random Forest (RF)

classifier with features from Layer 2 frames. To overcome their weaknesses, Taylor et al. propose AppScanner [5], a framework to classify apps with encrypted traffic by leveraging only side-channel information. It is trained and tested on 110 apps with traffic collected using a demultiplexing technique to obtain ground truth. The multiclass classification accuracy for the dataset is up to 73.1% using RF. The authors further attempt to improve results by introducing an approach to separate ambiguous traffic at the cost of reducing classified flows. Without this practice, their model's robustness is frustrated in terms of devices and app versions [4]. Recently, Aceto et al. [6] proposed a multiclassification approach of intelligently combining outputs from state-of-the-art classifiers to improve the performance of APP-ID. The performance can be improved according to all considered metrics up to +9.5% recall score with respect to the best base classifier. On the whole, the state-of-the-art solutions on APP-ID considering encrypted traffic rely on traditional ML techniques with handcrafted features. Besides, the datasets they use give little attention to either accurate ground truth labeling or richness of real-world data.

2.2. DL Practices in Traffic Classification. Recently, as much larger and much deeper neural networks have shown impressive capability across a range of difficult problem domains, researchers have begun to apply deep learning on Internet traffic classification and reveal promise for that. The first successful attempt is introduced in [18] by a security engineer. It focuses on Multilayer Perceptron (MLP) and Stacked Autoencoder (SAE) with raw traffic as input. It uses the real data collected from enterprise network and shows that the deep learning approach works well on the applications of feature learning, protocol classification, and anomalous protocol detection. Wang et al. [19] propose an end-to-end method of encrypted traffic classification with one-dimensional convolution neural networks (1D CNN). They evaluate the model on the public ISCX VPN-non-VPN traffic dataset and show that it achieves outstanding performance on both non-VPN and VPN traffic, about 10% higher than the state-of-the-art method in precision and recall. The same dataset is also used in [20], which proposes a framework embedding SAE and CNN to classify network traffic. Differently, it keeps the IP header and the first 1480 bytes of each IP packet as input and performs classification at packet level. The framework achieves F1 score of 0.95 in application identification task and 0.97 in traffic characterization task. In addition to CNN, Recurrent Neural Network (RNN) is also introduced for traffic classification in [21]. It presents a technique based on combination of CNN and RNN, which can be used for Internet of Things (IoT) traffic. In order to get better results, the authors try different set of features that include header and statistic information other than payloads. It is shown that port numbers play an important role in IoT traffic classification, which is intuitive, as many services keep their assigned ports. The DL practices on traffic classification mainly focus on public datasets that rarely include mobile traffic. In this paper, we are dedicated to constructing a reliable mobile dataset and evaluating a variety of deep learning models for APP-ID comprehensively.

3. Methodology

In this section, we first give the problem definition of APP-ID, declaring some key concepts and principles. Then we give a description of our dataset methodology, specifically on how to achieve the goal of collecting traffic with trusted ground truth as well as scalability using NetLog. A brief introduction of our DL models for APP-ID is provided at last. Their architectures are specially designed and optimized for the data and task.

3.1. Preliminaries. In this work, we follow the majority of prior works and formulate APP-ID as a multiclass classification problem. Namely, we perform a supervised multiclass classification to find a function that maps a traffic bidirectional flow (biflow) to a mobile app that generated it. More formally, suppose that a biflow f that was created by an app a is an instance of the form (f, a). Given a set of observed biflows from the network, $S = \{(f_i, a_j): f_i \in F, a_j \in A\}$, where F is the set of unlabeled biflows and A is the set of all possible apps that generate them, the problem is to find a function $g: F \longrightarrow A$, such that each unlabeled biflow $f_s \in F$ can be assigned to an app $a_t \in A$, satisfying as much as possible the fact that $(f_s, a_t) \in S$.

In the context of APP-ID, a traffic flow is a sequence of packets defined by the typical 5-tuple data structure $(\mathrm{IP}_{\mathrm{src}}, \mathrm{Port}_{\mathrm{src}}, \mathrm{IP}_{\mathrm{dest}}, \mathrm{Port}_{\mathrm{dest}}, \mathrm{Protocol}_{\mathrm{L4}})$, namely, the source and destination IP addresses as well as port numbers plus the Layer 4 protocol. On the basis of that, a biflow is the combination of two flows right in the opposite direction, which can be treated as packets in a session between a mobile device and a remote host. We believe that a biflow is the largest granularity we can use for APP-ID, which requires the least effort to classify all the traffic. That is why we consider it as our classification object.

In this work, we apply various DL methods to APP-ID, which train deep neural network classifiers that can be fed with raw data and automatically discover the features needed for identification. In practice, the raw data for the DL models to consume would be a fix-length sequence of bytes from the payloads in a biflow. We will remove all the packet headers in a datagram (i.e., the Ethernet header, the IP header, and the TCP/UDP header) and only use as input the top hundreds of bytes in a biflow's payloads. Our choice of this input is supported by the knowledge that the first few packets within a biflow normally capture the application negotiation phase in their payloads, which is distinct among applications. Moreover, byte sequences can be easily normalized or encoded to feed the networks without losing much information. Although some packet header fields such as IP and ports may also contribute to the identification, they are not so trustworthy due to reasons we mentioned; thus we want to totally avoid them.

3.2. Mobile Traffic Collection and Dataset Construction. Most previously proposed APP-ID solutions report decent performance with results based on private datasets, usually labeled by approaches of unknown reliability. In order to obtain the ground truth, the most common way is running apps one by one separately and manually label the trace [6]. It is, however, not viable due to the background traffic generated by system or sleeping apps. Though some public tools may aid the process of labeling [4], they are basically obsolete with the need for ROOT access. Automatic ground truth labeling is by no means a trivial work. We thus develop NetLog—an Android monitoring tool for accurate mobile trace labeling. With the help of it, the construction of real network mobile traffic datasets for APP-ID is quite convenient.

3.2.1. NetLog. NetLog is capable of monitoring all network access attempts and communications originating from both user and system apps. It can generate logs for each app's TCP/UDP communications and export all raw traffic as pcap files. The exported pcap traffic files and the corresponding logs will be uploaded to a server through HTTP, where further analysis can be performed, including automatically associating each flow with its generating app.

The core functions of NetLog are built upon the *VpnService* class—a standard API provided by the Android SDK. It creates a virtual network interface (i.e., TUN interface), configures addresses and routing rules, and returns a file descriptor to NetLog. Then NetLog will be able to operate raw network traffic at user space by reading or writing the special file descriptor, which is why root permissions are no longer needed. The networking procedure of NetLog is illustrated in Figure 1. All the smartphone's generated traffic that passed through the network stack is routed to the *tunX* interface first. NetLog then has access to all the outgoing IP packets by reading the/*dev/tunX* file. It deals with the raw packets like a Layer 3/4 network stack, ensuring that all packets are sent to their intended destination through the socket. Meanwhile, a duplicate copy of the packets is created and exported for further uploading. The process is similar for incoming packets. In order to accurately distinguish traffic from different apps, NetLog obtains UIDs to each flow by reading the/*proc/net/tcp* and/*proc/net/udp* files. An UID can be mapped to a specific app by Android's *PackageManager* API. Finally, a log for all observed app-flow mappings is created and waits to be uploaded to a cloud server along with the traffic files.

3.2.2. Dataset. NetLog is designed to automatically upload raw packets and app-flow mapping files to a shared cloud server when connected to Wi-Fi. It is efficient to scale up, since the tool is completely transparent to users. For a wide collection campaign, we distributed NetLog to different volunteered Android users both on and off campus. With no guidelines or interferences, all the volunteers needed to do is using their smartphones normally as usual. The data collection process is divided into two periods. The first lasted 4 weeks (August 2018–September 2018) with 12 volunteers on campus involved and the second lasted 10 weeks (May 2019–July 2019) with 16 off-campus volunteers involved. Eventually, we are able to achieve our goal of building a large-scale dataset for APP-ID which is (1) accurately labeled and (2) scalable and (3) produces high richness in real network.

We collected traffic from 142 mobile apps in total, which add up to a volume of 70.4 Gigabytes, covering a variety of categories like messaging, social, shopping, videos, and so forth. All of them are among the most popular apps in China. Some of the most prevalent ones like WeChat, QQ, and Taobao contributed the largest part. After collection, the traffic was first divided to be labeled by different app name based on the app-flow mapping list. Then the traffic was filtered to include only TCP and UDP packets that were error-free. Using a modified open source package, Joy [22], we further analyzed the cleaned traffic and resolved it into file of "biflows." The "biflow" here is a data structure containing a big amount of statistical information, including 5-tuples, packets length/interval sequences, and payload bytes of fixed length. The packet sequences are used for extracting statistical features to feed the traditional ML classifier. The payload data, segmented from the chronologically concatenated packet payloads in a biflow, is used as inputs of the DL models.

In association with the app name labels, our dataset was finally constructed, consisting of 850 K + biflows (300 K + TLS biflows). As the traffic was collected through a process of freely using the smartphones in daily life, we can see from Figure 2 that the quantity of samples for each app is highly unbalanced. The longest bar on the graph stands for WeChat, the predominant messaging and social app in China. The encrypted traffic biflows represented by TLS are shown in red.

3.3. DL Models for APP-ID. Deep learning methods can train supervised neural network classifiers with raw data. During the training, DL models learn multiple levels of features in which higher layers amplify aspects of the input which are important for discrimination and suppress irrelevant variations [10]. To demonstrate DL's superior ability of feature learning for APP-ID, we apply three major types of DL models here: Autoencoders, Convolutional Neural Networks, and Recurrent Neural Networks. They are among the most popular and well-studied deep learning technologies and will probably provide empirical insights into solutions to APP-ID.

3.3.1. Autoencoders. An Autoencoder (AE) is a shallow feedforward neural network designed for learning efficient data representations by simply learning to copy its input to its output [23]. The network is always composed of two parts: an encoder (or recognition network) that converts the inputs to an internal representation, followed by a decoder (or generative network) that converts the internal representation to the outputs. Usually AEs are designed to be restricted in ways that allow them to copy the inputs only approximately and unable to learn to copy perfectly,

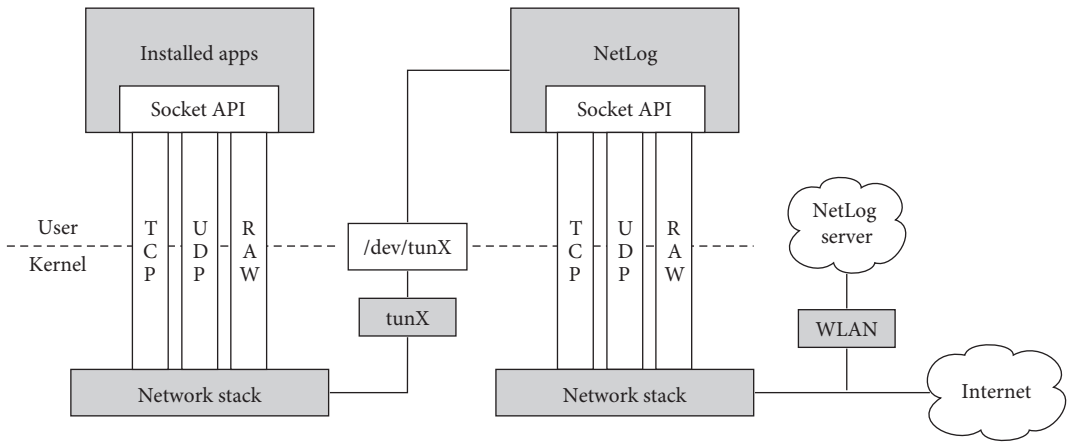

Figure 1: The workflow of NetLog.

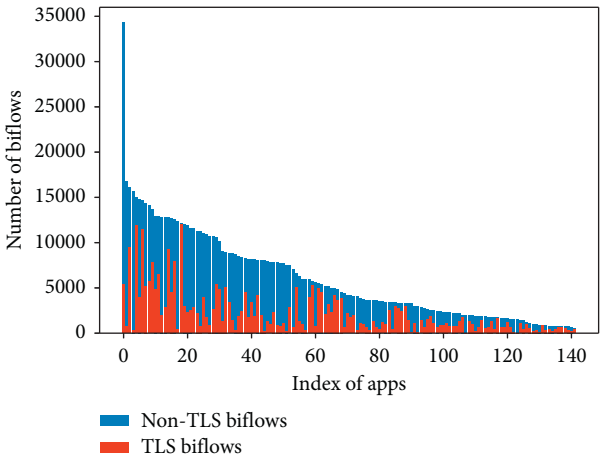

Figure 2: Description of the dataset.

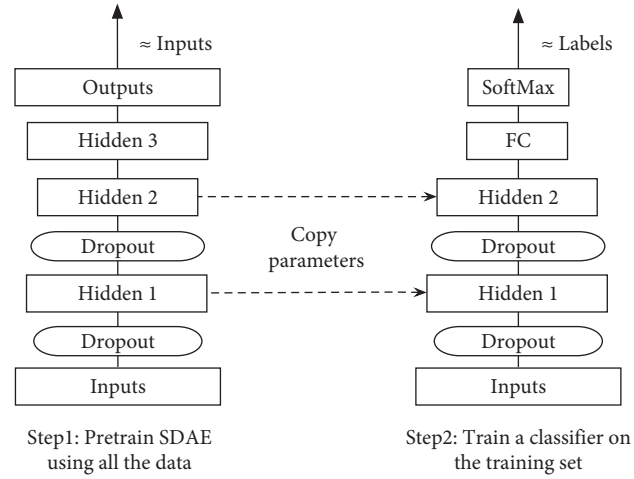

Figure 3: SDAE for APP-ID.

yet they must find a way to output a copy of the inputs. They are forced to prioritize which aspects of the inputs should be copied and learn the most important features in the data. One way to force the autoencoder to learn useful features is to add noise to its inputs, training it to recover the original, noise-free inputs. This is called Denoising Autoencoder (DAE). Chaining multiple DAEs together, we will have a Stacked Denoising Autoencoder (SDAE). It can extract data from the input hierarchically to learn features of different levels of abstraction. Geoffrey Hinton et al. [24] discovered that deep neural networks can be pretrained for better performance in an unsupervised fashion, which also works for autoencoders. Thus, we can pretrain a SDAE using all data and reuse its encoder layers to create a new neural network for classification. The new model will learn to classify the input by back-propagating the classification errors in a supervised way.

To build our SDAE for APP-ID, we stack multiple hidden layers and add noise by randomly switching off inputs using the dropout layer. We first train the SDAE using all the data and then reuse its encoder layers to create a new neural network for classification, as depicted in Figure 3.

3.3.2. Convolutional Neural Networks. A Convolutional Neural Network (CNN) is quite similar to an ordinary neural network. It is specifically designed to process data that come in the form of multiple arrays. There are four key ideas behind CNN, which take advantage of the properties of natural signals: local connections, shared weights, pooling, and the use of many layers [10]. The most important building block of a CNN is the convolutional layer. Each neuron in the convolutional layer is connected only to neurons located within a small rectangle in the preceding layer. This allows the network to concentrate on low-level features and assemble them into higher-level features in the next layer. The pooling layer is used by a CNN to subsample the input in order to reduce the computational load and the number of parameters, thereby limiting the risk of overfitting. Typical CNN architectures stack a whole series of convolution and pooling layers, and the resulting outputs need to be flattened and concluded by at least one regular fully connected layer prior to classification. Since many data modalities are in the form of multiple arrays, 1D for signals and sequences including language, 2D for images or audio spectrograms, and 3D for video or volumetric images, CNNs have been applied in a variety of tasks and achieved

superhuman performance. Taking network traffic as byte sequences, it is intuitive to fit a 1D CNN model to it for classification.

Our raw data inputs are fixed-length byte segments, where the location of the feature within the segment is not of high relevance, thus making a 1D CNN, which is effective to derive features from fixed-length segments of the overall dataset, an ideal model. Unlike the densely connected neural network layer that needs 1D data as input, a 1D CNN accepts 2D matrices. Therefore, each byte in our segments can be encoded into vectors of fixed size to feed the model. The typical 1D CNN architecture for APP-ID is depicted in Figure 4.

3.3.3. Recurrent Neural Networks. A Recurrent Neural Network (RNN) is a type of neural network architecture particularly suited for modeling sequential phenomena. Given a standard feedforward MLP network, an RNN can be thought of as the addition of loops to the architecture. For example, in a given layer, each neuron may pass its signal laterally (sideways) in addition to forwarding to the next layer. The output of the network may feedback as an input to the network with the next input vector, and so on. The recurrent connections add state or memory to the network and allow it to learn broader abstractions from the input sequences theoretically. In practice, however, learning long-range dependencies with a vanilla RNN is difficult due to vanishing/exploding gradients [25]. Long Short-Term Memory (LSTM) addresses the problem by augmenting the RNN with a memory cell vector at each time step [26]. This allows deep networks with multiple layers to be created, which is often crucial for obtaining competitive performance on various tasks.

As RNNs typically deal with sequential data to capture the contextual information, we have to divide each sample in our dataset into multiple small segments to feed the network one by one. Considering that the RNN is a biased model, where later inputs are more dominant than earlier inputs, we use bidirectional RNNs for better learning. A bidirectional model simply put two independent RNNs together, while the input sequence is fed in normal time order for one network and in reverse time order for another. The outputs of the two networks are usually concatenated, allowing the networks to have both backward and forward information about the sequence at every time step, thus mitigating the bias. Combining bidirectional RNNs with LSTM gives bidirectional LSTM, which can access long-range context in both input directions. The final bidirectional LSTM network for APP-ID is depicted in Figure 5.

4. Evaluation

In this section, we provide a thorough experimental evaluation for the three different DL models based on our dataset. The performance measures and metrics are given first. Then we investigate the selection of some important parameters. After this, the performance evaluation of our DL models including comparisons with the baseline methods is presented.

4.1. Performance Measures and Metrics. There are several metrics for evaluating our methods. Before the definitions, we first introduce four related terms. For each app in the test set, True Positives (TP) indicates the number of biflows identified as this app which indeed belong to it. True Negatives (TN) is the number of biflows that are correctly judged as not belonging to this app. False Positives (FP) refers to the number of biflows that are wrongly identified as this app. False Negatives (FN) is the number of biflows belonging to this app while incorrectly classified as belonging to other apps.

For evaluation of the test set, we have Precision, Recall, and F-measure (also $F1$ score or F-score) for every target app:

$$\begin{aligned} \text{Precision} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\ \text{Recall} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ F - \text{measure} &= \frac{2\,(\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}. \end{aligned} \tag{1}$$

The Precision can be seen as a measure of exactness or fidelity, whereas Recall is a measure of completeness. F-measure, the harmonic mean of Precision and Recall, gives equal importance to them. We also have the global accuracy for all the target apps:

$$\text{accuracy} = \frac{\sum_{i=1}^{N} TP_i}{\text{Total}}. \tag{2}$$

where N is the number of apps, Total is the total number of biflows in the training set, and TP_i means the number of app i's TPs. Besides, considering the presence of class imbalance in our multiclass settings, we use the macroaveraged F-measure as the main performance measure due to its equal emphasis on rare classes.

To make our DL models suitable for classification tasks, we use the SoftMax function in the output layer to produce a probability distribution. The training loss that expresses the errors is a categorical entropy:

$$E = -\frac{1}{N} \sum_{i=1}^{N} (p_i \log\, p_i), \tag{3}$$

where p_i is a returned probability for the predicted class with N categories in total. A classifier confident of its decisions gives a high probability for each predicted class, which results in a minimized entropy.

4.2. Parameter Selection. There are several important parameters for training our DL models including the number of payload bytes as input and the hyperparameters for the neural networks. Here we give the selection and setting of those parameters.

4.2.1. Number of Input Payload Bytes. For a couple of reasons, we have chosen a fixed-length payload segment as

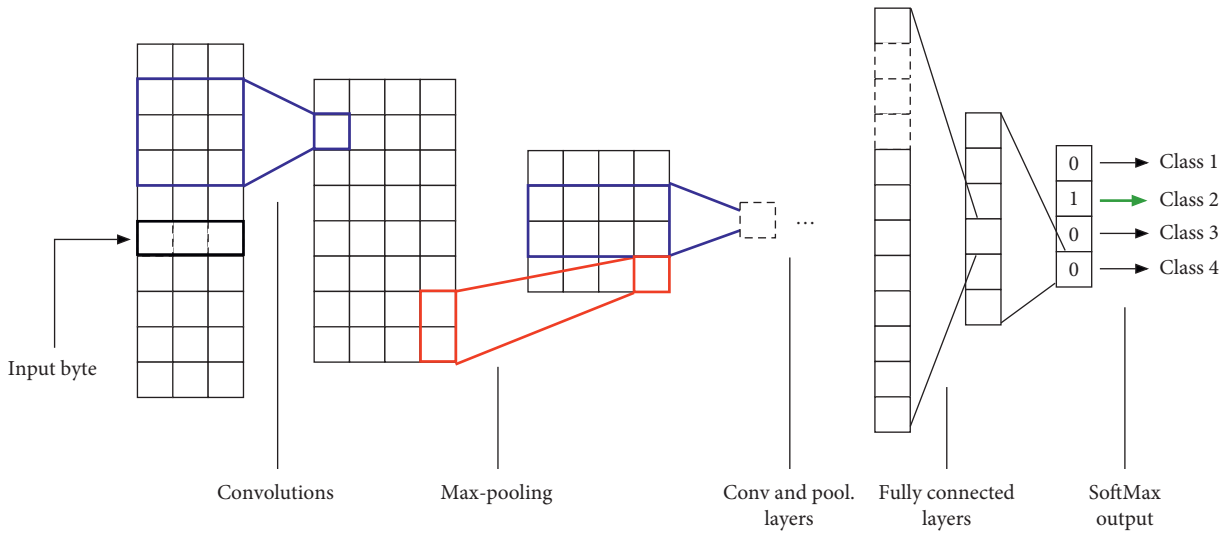

Figure 4: 1D CNN for APP-ID.

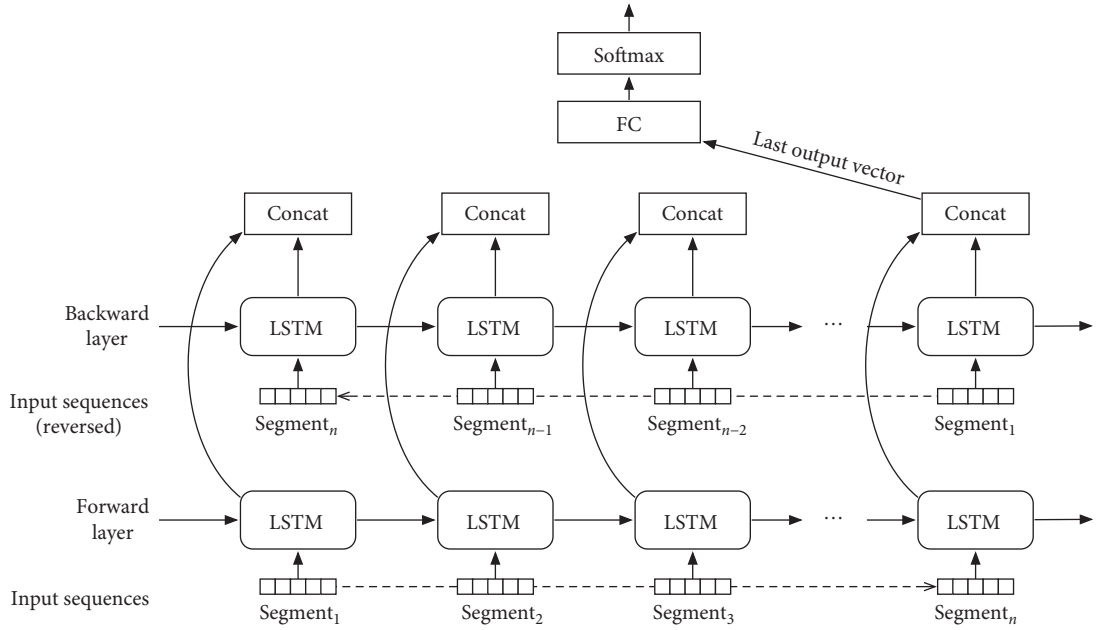

Figure 5: Bidirectional LSTM network for APP-ID.

raw input data for our DL models. Considering that too short a payload segment may miss some key features while too long an input suggests redundant data and unnecessary training cost, it is very useful to set an appropriate length for it in practice.

To find the optimal input payload length for APP-ID, we evaluated the skill of 1D CNN model on various numbers of valid input bytes. The dedicated 1D CNN model for APP-ID has a fixed input length of 1014, within which we believe adequate features for classification can be derived. We formatted each sample in the dataset into a shorter payload segment and padded it to the length of 1014 with zeros at the bottom in order to feed the model for training. Through a series of tests, we can find out the optimal length of payload segments.

The evaluation results are shown in Figure 6, presented with the model's best validation accuracy after training for 20 epochs. We can see that the model's performance rises dramatically when the number of payload bytes for input increases from 0 to 100 and steadily grows until the number reaches around 300. With input payload bytes more than 300, the accuracy shows no evident signs of improvement any more. The results indicate that 300 bytes are generally sufficient for DL models to learn features for APP-ID (other models perform not as good as 1D CNN). It is probably the optimum number of input payload bytes we should apply.

4.2.2. Hyperparameters of the DL Models. The performance of many contemporary DL algorithms depends crucially on

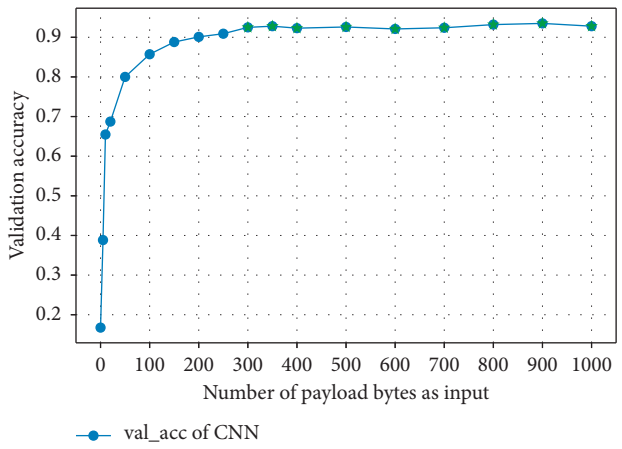

Figure 6: Impact on the CNN's performance with a growing number of input payload bytes.

the specific initialization of hyperparameters such as the general architecture, the optimizers, regularization parameters, and many others. We set hyperparameters of our DL models with best effort to achieve decent classification performance and to enhance their capabilities to generalize well to unlabeled traffic. Table 1 presents the details.

The proposed SDAE architecture has 3 hidden encoding layers stacked on top of each other with respective dropout layers ahead of them. We limit the size of the encoding layers to 200 and 100, making it undercomplete. For the APP-ID task, a SoftMax classifier is added to the encoder part, which is pretrained over the whole dataset. Besides, we use 300-byte payload segments as input, which has been discussed before.

The proposed CNN is designed to accept fixed-length inputs of 1014 bytes, which is tested to be sufficient for our purpose. We use two identical 1D convolutional layers and 1D max-pooling layers in pairs to learn traffic representations. The convolutional layers have stride of 1 and pooling layers are all nonoverlapping ones. They are followed by flatten and fully connected layers in preparation for the final SoftMax classifier. We also insert a dropout layer with dropout rate of 0.5 after the fully connected layer for regularization.

Different from SDAE or CNN whose input data is the whole payload segment, the input for our LSTM network is a sequence of small segments equally divided from the original payload bytes. The length of the small segment consumed by a LSTM cell is set to 20, which is long enough to capture the critical information. The dimension of LSTM is set to 100. We use two connected bidirectional LSTM layers, with the first generating a sequence of concatenated vectors and the second outputting the last one vector. A dropout layer is also added for regularization.

In the training step, we choose Adam—an adaptive learning rate optimization algorithm, as our optimizer in terms of speed of training. For SDAE and CNN, training with 20 epochs will achieve global optimal accuracy. The process is longer for LSTM network, which needs 100 epochs.

4.3. Experimental Results. Here we provide the performance evaluation for our proposed DL models. A comparison with two baseline approaches is also included. The first baseline approach is a multiclass Random Forest Classifier based on statistical features extracted from traffic flows. This method is proposed in [4, 5], which represents the state-of-the-art mobile traffic classifier using traditional machine learning techniques. To reflect the benefits of "deep" learning models, we use a Multilayer Perceptron (MLP) classifier with two hidden layers of 100 nodes as our second baseline approach. This has been introduced to the task of traffic protocol identification in [18], which also takes raw payload data as input and shows impressive capability of feature extraction.

Table 1: Hyperparameters of the proposed DL models.

Hyperparameters	SDAE	1D CNN	LSTM
Input units	300	1014	20
Hidden layer units	300,200,100	—	100
Number of layers	7	8	5
Dropout rate	0.05	0.5	0.5
Activation	Tanh	ReLU	ReLU
Optimizer	Adam	Adam	Adam
Batch size	30	30	30
Training epochs	20	20	100
Number of filters	—	256	—
Filter size	—	7	—
Pooling size	—	3	—

4.3.1. Experiment Setup. The evaluation for our proposed DL models and the baseline approaches are based on the same dataset that is split into train/test set in proportion of 75%/25% with stratifying and shuffling. The dataset is constructed from preprocessed traffic data to include both flow statistical features and payload segments required by the tests. According to [5], the statistical features are derived from three packet length sequences (i.e., incoming, outgoing, and bidirectional) within a biflow. The three packet sequences actually represent a burst part rather than the whole biflow with the burst threshold set as 1 second. For each of the sequences, 18 statistical values are computed. Then a feature selection is performed to keep the top 40 of the total 54 values as final features for the baseline RF classifier. On the other hand, the payload segments of biflows can be accessed much more easily. They can be conveniently used as inputs for our end-to-end DL models as well as the baseline MLP directly after a simple normalization. To normalize a byte sequence for the neural network that accepts vector of values in the range [−1, 1], we only require each byte to be divided by 255, namely, the maximum value of a byte. For the 1D CNN that accepts inputs with an additional feature number dimension, we replace every byte by a vector of size 256 using the one-hot encoding method.

In addition to the evaluation with the whole dataset, we also run experiments on the encrypted part of the dataset to estimate our models' skill to classify encrypted traffic. It is hard to recognize every single biflow that has applied encryption. Considering that HTTPS/TLS is widely used nowadays and the TLS biflows account for a considerable proportion of over 30% in our dataset, we separate them

from the original dataset to create a new dataset representing the encrypted traffic. Evaluations with the new dataset are performed in the same way.

Training a network to find the global minimum loss generally requires iterating many times through the whole training set, which is reflected in the hyperparameter setting of epochs. During epochs of training, we checkpoint all observed improvements and save the model with the best accuracy on the validation set for analysis. In order to control the model variance, we apply a tenfold cross validation for each considered test. The average of each performance measure will be reported as the final evaluation result.

Our DL experiments are all carried out on a workstation with a Nvidia GeForce GTX 1080 GPU (2560 cores and 8 GB memory). The DL models are built by Keras API with the backend of TensorFlow.

4.3.2. Evaluation Results. In the experiment settings, we perform evaluation tests on the DL and baseline models with two datasets, respectively. To present the final results concisely, we use accuracy and macro-average *F*-measure as the performance metrics. All results are reported in Table 2 with the best ones highlighted.

From the results, we can see that the DL models of CNN and LSTM significantly outperform the baseline models of RF and MLP with accuracy and macro-*F*1 all higher than 80%. It is worth noting that those two DL models plus MLP all have shown far better performance than the RF with improvements over 20 percent, which implies that the features they learn from raw traffic data are far superior to the statistical features for the task of APP-ID. However, despite the outstanding results, the DL model of SDAE has presented poor performance, which is even worse than the RF classifier. We may find some clues by drawing a comparison between SDAE and MLP, since they share a similar structure of fully connected layers as well as the inputs. The SDAE classifier has pretrained layers with frozen weights for the purpose of preserving the ability of learning good representations of data. Nevertheless, the representations learned by the intermediate layers of SDAE in an unsupervised fashion do not need to have general "goodness" for other types of tasks like classification. Thus, a supervised trained MLP classifier would likely be a better choice for our task.

For the evaluations based on the TLS dataset, the results are close to those on the whole dataset with even small improvements of accuracy. This reveals the fact that encrypted TLS traffic can be classified by its raw data as well. When looking into the input payload information of TLS biflows, the *Client Hello* and *Server Hello* messages that are transmitted in cleartext are typically found within. Moreover, these messages generally contain distinguished fields like *server_name* that can be learned by DL models for the task of classification. It is probably the reason why TLS biflows are showing no difficulties to be classified just as the ordinary biflows do. We will substantiate this with interpretation results later.

Throughout the experiments, the LSTM network model has exhibited impressive potential for APP-ID, while the best model for the task is undoubtedly the 1D CNN, which presents 91.80% accuracy and 90.10% macroaverage *F*-measure that overshadow all others. Technically, the 1D CNN is an unbiased model that can fairly determine discriminative patterns in a sequence with a max-pooling layer. Thus, it may better capture the local features compared to the RNNs. To further demonstrate the APP-ID skill of these two DL models and discover their error patterns, we create their confusion matrices, respectively. Due to the limitation of space, we rank all the apps according to their *F*-measure values and only select the worst 50 to display. From the heatmaps in Figure 7, we can naturally see that these two DL models' confusion matrices are quite alike. For example, they both have the same two dark nodes at the location of (6, 5) and (22, 21), right adjacent to the diagonal line. It indicates that both models have a poor classification recall on app 5 (Jingdong) and app 21 (Tmall). The strong resemblance actually implies that the two models with different architectures are trained to learn a set of similar features for APP-ID. Besides, we can also infer from the matrices that app 22 (Taobao), which has a large number of FPs and FNs, is probably one of the most complicated apps. In general, there are only few apps in the app set which have very poor classification results. They are mostly related to each other and offer the same or similar services, which makes them lack unique signatures.

4.3.3. Effect of Training App Set Size. To evaluate the robustness of our DL models when training with a larger app set, we run experiments with different sizes of training app set to test the performance of the models. For each different size, a certain number of apps are randomly selected and a subdataset including only these apps is created to train the models. The random app selection and training for each app set size is repeated 10 times with the average as the final result.

Figure 8 shows the impact of the training app set size on the performance of the DL models. On the whole, the performances of two DL models show similar trends. Consistent with the previous results, the 1D CNN outperforms the LSTM network. When the app set size is limited to a minimum, that is, 2, the *F*-measures are both higher than 0.96, since the majority of the apps are easy to distinguish from each other as mentioned before. As the training app set size increases, the performance slowly declines, since those apps that are difficult to classify are more likely to be included. Although the performance of the DL models has been degraded in terms of the results, it does not mean that the models will not be as effective with a larger app set. When the app set is close to complete, the performance remains stable around the previous evaluation results. This indicates that further increasing the number of apps would probably not degrade the performance, since the added apps are very likely to be easy to identify and the models are able to learn effective features to classify as many of them as possible.

Table 2: Accuracy and macroaverage *F*-measure for the baseline and DL models on two datasets, respectively.

	Model	Dataset-all		Dataset-TLS	
		Accuracy (%)	Macro-F1 (%)	Accuracy (%)	Macro-F1 (%)
Baseline	RF	61.32	50.30	62.91	50.07
	MLP	81.24	70.64	83.57	65.79
DL	SDAE	58.73	39.86	60.37	41.45
	1D CNN	**91.80**	**90.10**	**93.14**	**84.25**
	LSTM	85.09	80.77	89.36	80.51

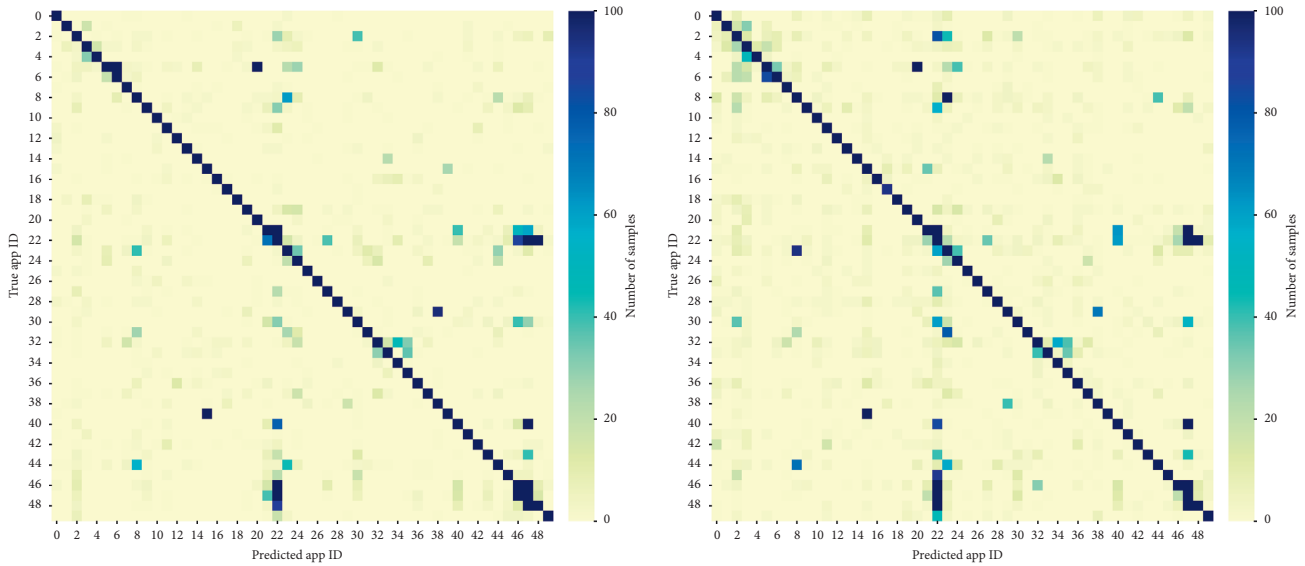

Figure 7: Confusion matrices for two DL models. (a) 1D CNN. (b) LSTM network.

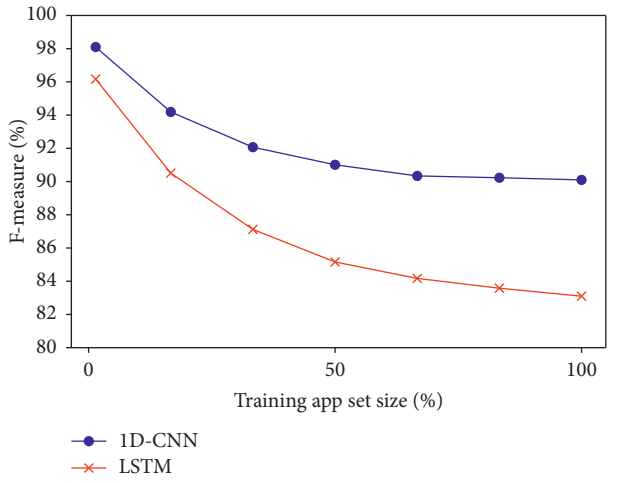

Figure 8: Impact of the training app set size on the *F*-measures of two DL models. 100% training app set size refers to using the training set with all the apps (142).

4.3.4. Limitations and Open Issues. Deep learning architectures with a large number of parameters typically need a proportional number of samples to train to achieve good performance. Our dataset, though delicately established, may not be sufficient to train a very deep state-of-the-art neural network model. It is also unbalanced with few apps only having hundreds of biflows, which is prone to leading to biased results. With our proposed NetLog and the scalable approach, however, the dataset can be enlarged regularly in a convenient way.

It seems that the DL models trained for APP-ID mainly learn useful features from the top hundreds of payload bytes in a similar way to performing text classification. We will give detailed information through model interpretation in the following section. The ability to deal with the actual encrypted traffic is yet to be explored.

In previous research, the SAE has been proposed to be effective for the task of traditional traffic classification [20]. However, in our systematic experiments, the classifier using autoencoders did not show any potential on the task of APP-ID, while training an MLP with the same layers would do much better. It seems that a loss layer of our interested task rather than a loss layer of reconstruction is preferable. On the other hand, the success of DL models like CNN and RNN has encouraged hybrid and more advanced network architectures, like ResNet or Inception layers, to be applied for APP-ID.

5. Model Interpretation

Understanding why a model makes a certain prediction can be as crucial as the prediction's accuracy in many

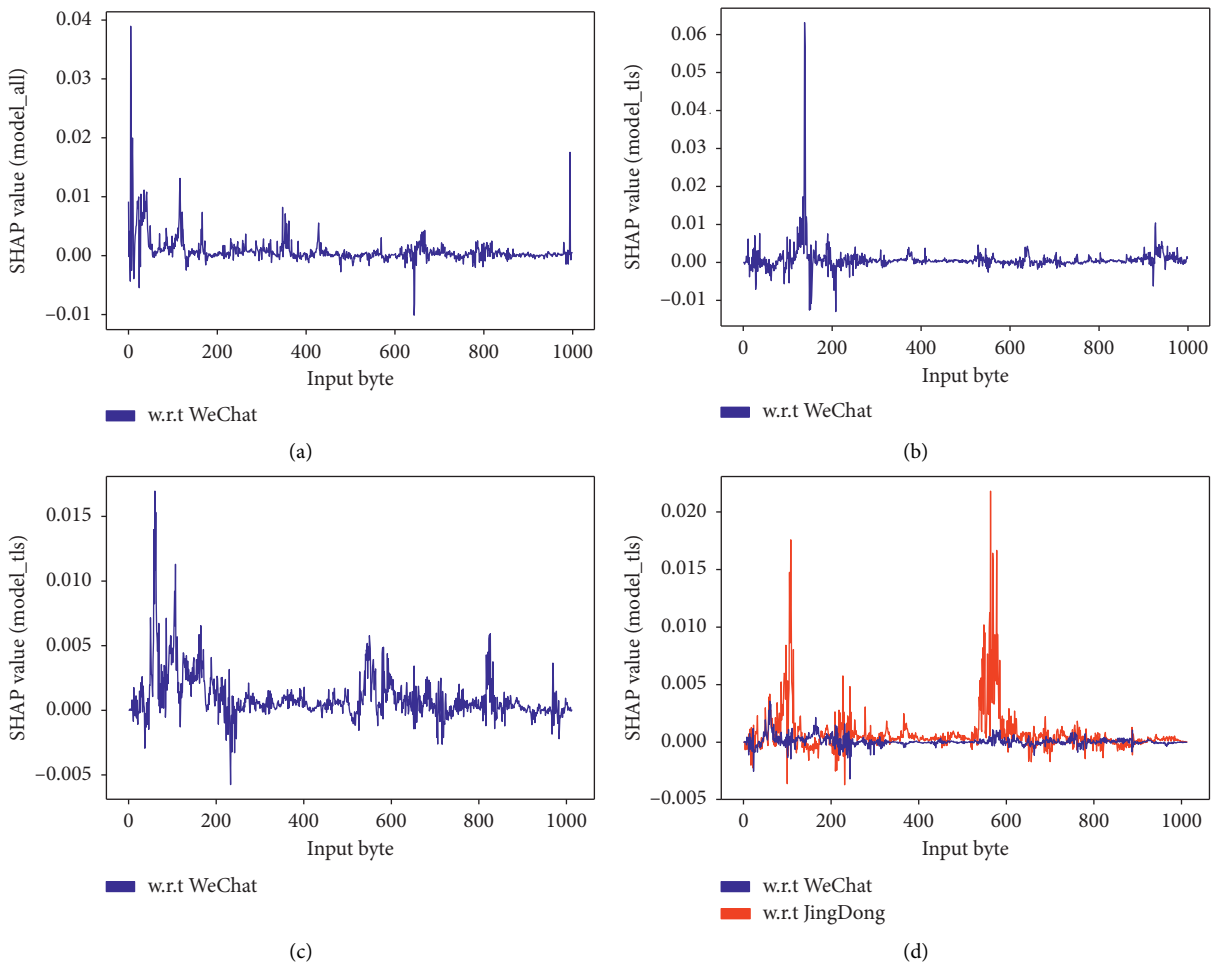

Figure 9: Distribution of SHAP values on input bytes for 4 different test samples of WeChat. (a) Test sample 1. (b) Test sample 2. (c) Test sample 3. (d) Test sample 4.

applications. A well-performing prediction model probably discovers patterns that users would very much like to understand. In some cases, interpretation of outputs can also help to learn the reason why a model might fail and provides insight into how it may be improved. Here, after training and evaluation, we will give our sample-specific explanations on the best-performing DL model of 1D CNN by using Deep SHAP [27]. It is a high-speed approximation algorithm for SHAP values in deep learning models, which builds on a connection with DeepLIFT [28], a recursive prediction explanation method for deep learning, which can be thought of as a fast approximation of the SHAP values.

The SHAP value is an alternative formulation of the Shapley values [29], which can be used to calculate the importance of an input by comparing what a model predicts with and without the input. It measures the average marginal effect of including an input over all possible orderings in which inputs can be included. Here we use the Python package *shap*, which implements several explanation methods including Deep SHAP in a unified approach, to explain our model predictions. For simplicity, we choose one representative app for explanation and calculate the SHAP values for each sample of it. WeChat, the most popular messaging and social app in China, has revealed the largest complexity in the model prediction, thus becoming our target.

The interpretation of our 1D CNN model based on Deep SHAP is sample-specific and here we give four representative outcomes in detail, as shown in Figure 9. Different from Figure 8(a) whose outcome is derived from the model trained on *Dataset-All* (*model_all*), the other three plots are all based on the *Dataset-TLS* trained model (*model_tls*). In the plots, a higher SHAP value means a greater contribution of the input to the target output, which gives different importance to the byte segments across the whole input with respect to the predicted app class. Generally, the SHAP values larger than 0.01 turn out to be the most prominent in the distribution and correspond to the most distinguished patterns found in the input bytes. Sample 1 represents a non-TLS traffic flow of WeChat typically with the top few bytes highly scored. However, the recognized patterns are not always the common HTTP header fields like GET, POST, and so forth. Here, in Figure 9(a), for example, the pattern stands for the byte sequence "032714eab5c49c" which is unknown. Sample 2 is a typical TLS flow with the greatest values located in the Extension: *server_name* field. Server Name Indication (SNI)

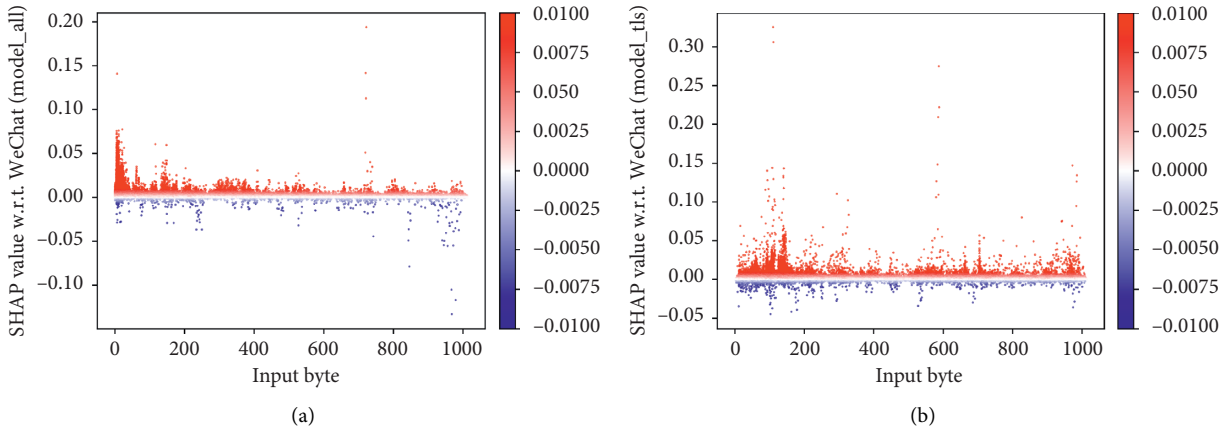

Figure 10: Stacked SHAP values over WeChat test samples for two different trained models. (a) Calculated based on *model_all*. (b) Calculated based on *model_tls*.

indicates which hostname the client is attempting to connect, and it is probably the most well-known service-related pattern at the start of the TLS handshake process. Specifically, the pattern revealed in Figure 9(b), "mp.weixin.qq.com," is contained in 134 TLS flows among the training set, with 117 of which belonging to WeChat. Sample 3 is a special TLS flow that has a deceptive SNI but is properly predicted. The SNI "http://www.zhihu.com" would have attributed the request to the Zhihu app instead of WeChat as it appears 46 times in the Zhihu training set compared to only 4 times in the WeChat training set. Despite that, the model has made correct prediction by learning more complex features. It is shown in Figure 9(c) that a part of the Cipher Suites, "cca8cc14cc13," receives the highest scores rather than the SNI. In fact, these three cipher suites exist only in WeChat and other 14 apps, not including Zhihu, which explains why they are so important for the model to make the decision. No model is perfect. Sample 4 gives a false negative example showing how our model may fail for the task. The red plot in Figure 9(d) represents the calculated values with respect to the false app. They are much more positive compared to the blue ones calculated with respect to the true label, making the model confident enough to give the wrong prediction. There are two peaks in the red plot, which stand for the SNI "item.m.jd.com" and a Certificate subject "Beijing Jingdong Shangke Information Technology Co., Ltd.," respectively. Unfortunately, both of them frequently appear in the Jingdong training set and show little connection with WeChat. The model cannot learn good features from the limited samples of information to divide them.

To better illustrate the overall SHAP value distributions on the WeChat samples, we present two stacked scatter plots in Figure 10. The red dots represent positive SHAP values that increase the probability of the class, while the blue dots represent negative SHAP values that reduce the probability of the class. Due to the different training and testing sets, those two plots vary in a range of aspects. In Figure 10(a), the values are quite large and clustered within the top few bytes, which is reasonable, as a majority of flows are transmitted in form of HTTP and the header fields of URI are quite distinguished. In Figure 10(b), the most clustered part is between bytes 100 and 170, where the Cipher Suites and SNI extensions are generally located. Besides, the distribution also shows a greater fluctuation with positive values across the whole input. As is shown in the previous examples, it is common that an app's traffic has no exclusive signatures in the flows and the model needs to learn more complex features from multiple patterns to recognize them. Sometimes those patterns even contribute to the opposite because of the ambiguous flows. In most cases, the top 300 bytes, which have been tested in the evaluation section, hold the useful and important patterns needed for APP-ID.

Through model interpretation, we can see that the 1D CNN model for APP-ID is powerful at discovering app signatures whether they are prominent or not and utilizing them to learn advanced features that a rule-based classifier would hardly manage to do. Still, the model is challenged by the fact that modern mobile apps have been integrating more and more functions and services, which increases the difficulty of rigid traffic classification. In general, given sufficient data, a DL classifier is promised to learn the differences between different classes as long as they exist among the inputs. The knowledge and features a DL model can learn from raw traffic data are meaningful and robust. Most importantly, it is completely in an automatic way.

6. Conclusion

In this paper, we propose a mobile traffic collection framework to construct dataset for APP-ID. We develop NetLog for Android smartphones and combine with a cloud server to collect real network traffic with accurate ground truth labeling and high scalability. A new approach based on deep learning technology for APP-ID is proposed and evaluated on our dataset. It takes advantage of DL methods to process raw packet payloads to automatically learn useful traffic representations for classification. Specifically, three most representative DL architectures, namely, AE, CNN, and RNN, are introduced to carry out the task. We have designed the dedicated classifiers, a SDAE, a 1D CNN, and a bidirectional LSTM network, respectively. In our systematic evaluation experiments, the CNN and LSTM show an

impressive performance improvement over two baseline classifiers, RF and MLP. To further understand how deep learning is qualified for the task of APP-ID, sample-specific interpretations of the best-performing 1D CNN model are performed with state-of-the-art tools. In summary, a DL classifier can automatically learn important and effective features for each app from the uppermost bytes of its raw traffic flows. The approach also shows promise for classifying the majority of the real-world mobile traffic, provided that mass ground truth data can be achieved through the methodology of our traffic collection.

Data Availability

The related dataset in this paper is not made public due to privacy and ethical issues. Requests for the original data, 6 months after publication of this article, will be considered by the corresponding author.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China under Grant no. 61806216.

References

[1] A. Dainotti, A. Pescape, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Network*, vol. 26, no. 1, 2012.

[2] Cisco Visual Networking Index, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper 1*, 2018.

[3] A. Ghosh, P. K. Gajar, and S. Rai, "Bring your own device (byod): security risks and mitigating strategies," *Journal of Global Research in Computer Science*, vol. 4, no. 4, pp. 62–70, 2013.

[4] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysisfication via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2018.

[5] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 439–454, IEEE, Saarbrucken, Germany, March 2016.

[6] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic-fication approaches for classifying mobile app traffic," *Journal of Network and Computer Applications*, vol. 103, pp. 131–145, 2018.

[7] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for internet traffic classificationfic classification," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223–239, 2007.

[8] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classificationfic classification," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1257–1270, 2015.

[9] K. Al-Naami, S. Chandra, A. Mustafa et al., "Adaptive encrypted traffic fingerprinting with bidirectional dependence," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 177–188, ACM, Los Angeles, CA, USA, December 2016.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[11] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler: towards automatic fingerprinting of android apps," in *Proceedings of the INFOCOM*, vol. 13, pp. 809–817, Turin, Italy, April 2013.

[12] S. Miskovic, G. M. Lee, Y. Liao, and M. Baldi, "Appprint: automatic fingerprinting of mobile applications in network traffic," in *Proceedings of the International Conference on Passive and Active Network Measurement*, pp. 57–69, Springer, Boston, MA, USA, March–April 2015.

[13] Q. Xu, Y. Liao, S. Miskovic et al., "Automatic generation of mobile app signatures from traffic observations," in *Proceedings of the 015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 1481–1489, IEEE, Kowloon, Hong Kong, April–May 2015.

[14] H. Yao, G. Ranjan, A. Tongaonkar, Y. Liao, and Z. M. Mao, "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, ACM, Paris, France, September 2015.

[15] G. Ranjan, A. Tongaonkar, and R. Torres, "Approximate matching of persistent lexicon using search-engines for classifying mobile app traffic," in *Proceedings of the INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, IEEE, San Francisco, CA, USA, April 2016.

[16] Y. Chen, W. You, Y. Lee, K. Chen, X. Wang, and W. Zou, "Mass discovery of android traffic imprints through instantiated partial execution," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 815–828, ACM, Dallas, TX, USA, October 2017.

[17] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: inferring app usage over encrypted data traffic," in *Proceedings of the 2015 IEEE Conference on Communications and Network Security (CNS)*, pp. 433–441, IEEE, Florence, Italy, September 2015.

[18] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, vol. 24, no. 11, pp. 1–10, 2015.

[19] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proceedings of the 2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 43–48, IEEE, Beijing, China, July 2017.

[20] M. Lotfollahi, R. Shirali, M. J. Siavoshani, and M. Saberian, "Deep packet: a novel approach for encrypted traffic classification using deep learning," 2017, http://arxiv.org/abs/1709.02656.

[21] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for Internet of thingsfic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18042–18050, 2017.

[22] Cisco, "joy", GitHub repository, https://github.com/cisco/joy.

[23] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: learning

useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[24] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[25] Y. Bengio, P. Simard, P. Frasconi et al., "Learning long-term dependencies with gradient descent is difficultficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the Advances in Neural Information Processing Systems*, Long Beach, CA, USA, December 2017.

[28] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, Sydney, Australia, August 2017.

[29] L. S. Shapley, "17. A value for n-person games," in *Contributions to the Theory of Games (AM-28), Volume II*, pp. 307–318, Princeton University Press, Princeton, NJ, USA, 1953.