Encrypted TLS Traffic Classification on Cloud Platforms

Xiaochun Yun, Yipeng Wang, *Member, IEEE*, Yongzheng Zhang, Chen Zhao, and Zijian Zhao, *Graduate Student Member, IEEE*

***Abstract*— Nowadays, encryption technology has been widely used to protect user privacy. With the explosive growth of mobile Internet, encrypted TLS traffic rises sharply and occupies a great share of current Internet traffic. In reality, the classification of encrypted TLS traffic on cloud platforms brings a new challenge to traditional encrypted traffic classification methods, because some information such as certificates in the TLS flows is no longer effective. In this paper, we apply deep learning technology to the problem of encrypted TLS traffic classification on cloud platforms, and propose NeuTic, which takes the packet sequence of each TLS flow as the input, and effectively classifies raw TLS flows generated by many "cloud" applications. Our approach is able to automatically capture the long-range dependencies between elements in the packet sequences for robust and accurate encrypted TLS traffic classification. In NeuTic, we first convert each TLS flow into three attribute sequences. Then, we train a multi-application traffic classification model using our newly designed deep learning model. Finally, we use the well-trained classification model to classify new incoming TLS flows. We conduct comprehensive experiments on real-world application traces covering multiple "cloud" applications from three different companies. In addition, we compare our experimental results of NeuTic with two deep learning-based methods for encrypted traffic classification. NeuTic outperforms the state-of-the-art approaches in classification accuracy.**

***Index Terms*— Encrypted traffic classification, cloud platform, self-attention mechanism, network security.**

I. Introduction

A. Background and Motivation

APPLICATION traffic classification refers to the task of associating application traffic with their generating applications, and it has many practical uses in the area of networking and network security, such as quality-of-service (QoS), network measurement, tunnel detection and intrusion detection and prevention *etc* [1]–[4]. For network management, network operators need to understand and classify the traffic mixed by different applications for a better QoS of networks and network provisioning, such as different priority strategies. The research in this field has attracted the widespread attention in both academia and industry over the past decade, and the continuous evolution of advanced traffic analysis approaches, such as [4]–[14] justifies this lasting interests. Nowadays, encryption technology has been widely used to protect user privacy. With the explosive growth of mobile Internet, encrypted TLS traffic rises sharply and occupies a great share of current Internet traffic. In reality, the classification of encrypted TLS traffic on cloud platforms brings a new challenge to traditional encrypted traffic classification methods.

B. Limitation of Prior Art

This paper concerns the accurate classification of TLS[1] flows for different mobile applications on the cloud platform. For a mobile application that uses TLS as its encryption basic, the real traffic generated by the application is hidden in the message of TLS protocol by the encryption algorithm, and the content of the application traffic is not plaintext information any more. Recently, Chen *et al.* propose MAAF [15], which uses X.509 certificates embedded in the handshake of encrypted TLS flows to classify the encrypted flows of different applications. Specifically, Chen *et al.* use the "organizationName" filed and the "commonName" field in the X.509 certificate of each TLS flow for application classification. We find that MAAF is very effective for classifying mobile applications from different companies. In practice, we notice that different applications developed by the same company provide their corresponding services for users on the same cloud platform. At the same time, the certificate information embedded in the TLS flows of these "cloud" apps is usually the same. Take "Alipay", "Amap", "Taobao", "Tappiaopiao", and "Youku" five apps from the Alibaba corporation as an example. By parsing the X.509 certificates of the TLS flows for the above apps, we notice that many certificates in the TLS flows of these applications have the same "commonName" field and the "organizationName" field, such as "**.alicdn.com*" for the "commonName" field and "*Alibaba (China) Technology Co., Ltd.*" for the "organizationName" field. Therefore, MAAF cannot make a right distinction to such TLS flows on the cloud platform.

An alternative is that we can use the packet type sequence or the message type sequence of a TLS flow to perform encrypted traffic classification. Different from directly inspecting the

Manuscript received 24 May 2020; revised 12 April 2022 and 25 May 2022; accepted 22 June 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Ren. Date of publication 19 July 2022; date of current version 16 February 2023. *(Corresponding author: Yipeng Wang.)*

Xiaochun Yun, Yongzheng Zhang, and Chen Zhao are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China.

Yipeng Wang and Zijian Zhao are with the Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China (e-mail: yipeng.wang1@gmail.com).

Digital Object Identifier 10.1109/TNET.2022.3191312

[1]TLS, Transport Layer Security, is a fundamental cryptographic protocol suite widely used in the secure communication of today's Internet and mobile Internet.

content generated by mobile applications, the methods that based on sequence information attempt to understand the generation mechanism of different application traffic by observing the state transitions in each TLS flow. However, prior sequence-based methods also have some limitations. (1). First, some sequence-based methods only have short-term memory but no long-term memory for elements in a given sequence. For example, some sequence-based methods use first-order or second-order homogeneous Markov chain models to construct application fingerprints for classifying SSL/TLS traffic [16]–[18]. The Markov chain models for each application are built from ordered SSL/TLS packet/message sequences. Recall that the state transitions in first-order/second-order homogeneous Markov chain can only consider two/three adjacent states. Apparently, it is difficult to capture long-term relationships between states in a given SSL/TLS flow using such a low-order Markov chain model. Therefore, these models do not perform very well in the task of encrypted traffic classification. (2). Second, in recent years, methods based on recurrent neural network (RNN) have been applied to encrypted SSL/TLS traffic classification. However, learning long-range dependencies in sequential data is still a challenging problem for RNNs. Theoretically, advanced RNN models, such as LSTMs and GRUs, can have long-term memory. However, in practice, we find that this kind of long-term memory may become blurred as the distance between elements in the sequential data increases.

C. Proposed Approach

In this paper, we propose NeuTic, a **NEU**ral network-based TLS **T**raff**I**c **C**lassification method, which takes the packet sequence of each TLS flow as the input, and effectively classifies raw TLS flows generated by many "cloud" applications. In practice, we notice that packet sequences can more accurately reflect how the application flow is generated than message sequences. In addition, NeuTic is based on a newly designed deep neural network, and the major structure of the neural network model includes four key components, including input embedding, multi-kernel convolution, sequence self-attention, and classification output. NeuTic aims at classifying encrypted TLS flows. As shown in Fig. 1, NeuTic has two phases, (i) the training phase, and (ii) the classification phase. Specifically, the training phase aims to build a TLS traffic classification model for all target applications, and it has two major modules, including (1) flow processing, and (2) neural training. First, the flow processing module converts each labeled TLS flow into a packet sequence, where each packet sequence is composed of three attribute sequences. Then, according to the input sequences, the neural training module builds a TLS traffic classification model. In addition, the classification phase aims to classifies new incoming TLS flows, and it also has two major modules, including (1) flow processing, and (3) neural classification. Specifically, to classify a new incoming TLS flow, the flow processing module in the classification phase also first extracts three attribute sequences from the packet sequence of each TLS flow. Then, the neural classification module works on the three attribute sequences and leverages

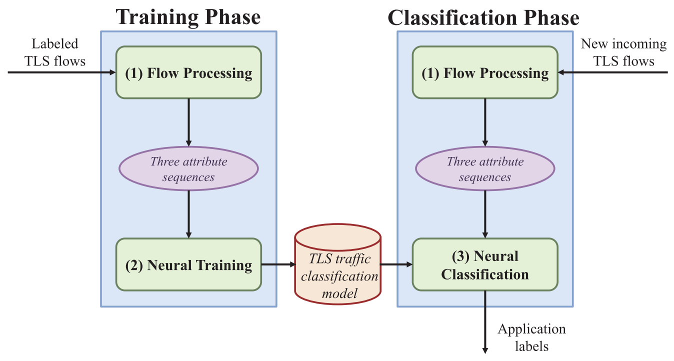


Fig. 1. Architecture of NeuTic.

the TLS traffic classification model generated by the training phase to assign a proper application label to the TLS flow to be classified.

D. Novelty and Advantages of Our Approach

NeuTic is composed of a newly designed neural network, and the design logics of NeuTic is based on the following two considerations. First, NeuTic extracts different raw fields from the IP packets of each flow, and uses them as the input to the neural network model. The designed neural network model can autonomously choose inputs that are more helpful in classifying encrypted TLS traffic. Second, NeuTic attempts to establish a relationship between any two elements in a given sequence in order to obtain better classification features. The prior state-of-the-art approaches can capture the relationship between adjacent elements in a given sequence, but it is difficult for them to establish the relationship between distant elements. Those methods have a limited ability to characterize sequential data, and thus miss the opportunity to achieve more accurate classification results. In contrast to these methods, NeuTic aims to better capture the relationships between elements in an input sequence.

E. Key Contributions

We highlight our key contributions as follows.

- We propose a novel deep learning-based method, NeuTic, for encrypted TLS traffic classification. NeuTic is purely based on the packet sequence of each TLS flow, and thus it does not need to assemble IP packets into TLS messages. NeuTic is very robust for application flows that use TLS as the encryption protocol.
- We design a novel neural network model that is equipped with a multi-kernel convolution that is able to flexibly produce variable-length features for sequential inputs, and also equipped with a sequence self-attention mechanism that is able to learn long-range dependencies between elements in sequential data. Through the above two design considerations, our proposed neural network model can automatically learn robust features from raw sequential inputs for accurate TLS traffic classification.

- We try to understand and explain what features and what packets are more helpful to identify the fingerprints of encrypted TLS traffic.
- We implement NeuTic and carry out extensive evaluations on real-world TLS flows generated by many mobile "*cloud*" applications. We compare NeuTic with two state-of-the-art encrypted traffic classification methods, RBRN [19] and FS-Net [20], both of which are deep learning-based methods. Our experimental results show that NeuTic outperforms the existing methods in terms of classification accuracy for encrypted TLS traffic classification.

The rest of the paper proceeds as follows. In Section II, we review related work. In Section III – Section V, we present the technical details of each component of NeuTic. Next, in Section VI, we evaluate the whole method with real-world application traces. We compare NeuTic with two existing methods in Section VII. Finally, we conclude our work in Section VIII.

II. Related Work

This paper focuses on the accurate classification of encrypted TLS traffic. Prior methods for encrypted traffic classification generally fall into two categories: traditional machine learning-based methods and deep learning-based methods.

A. Traditional Machine Learning-Based Approaches

In 2014, Korczyński *et al.,* propose FOSM [16], which first introduces the concept of Markov chain fingerprints to conduct encrypted SSL/TLS traffic classification. The basic idea of FOSM is to model possible sequences of message types observed in single-directional SSL/TLS sessions based on first-order homogeneous Markov chains. Notice that each application corresponds to a Markov chain fingerprint, and FOSM identifies the parameters of the application fingerprint from observed training application traces. For an encrypted flow to be classified, FOSM compares its message sequence with all fingerprints, and takes the fingerprint with the greatest probability as the classification result for the flow. In 2017, Shen *et al.* extend the concept of Markov chain fingerprints to build more distinctive application fingerprints [17]. They propose an attribute-aware encrypted traffic classification method called SOB, which is designed based on the second-order Markov Chains. Compared to FOSM, SOB introduces the certificate packet length and the packet length of the first application data in a session, in addition to the consideration of possible sequences of message types. In 2018, Liu *et al.,* propose a classification system called MaMPF [18], which is designed to build multi-attribute application fingerprints based on first-order homogeneous Markov chains for encrypted SSL/TLS traffic classification. Specifically, MaMPF separately uses sequences of message types observed in single-directional SSL/TLS sessions and sequences of transformed packet length of the corresponding encrypted flows to build application fingerprints. Note that, in [18], MaMPF uses random forest as its application classifier.

B. Deep Learning-Based Approaches

In recent years, deep learning techniques have achieved great success in many fields, such as speech recognition, computer vision and machine translation. Therefore, some people begin to consider using deep learning technology to solve the problem of encrypted traffic classification.

In 2017, Schuster *et al.,* bring a seminal research work to the networking and network security community, using deep learning technology to carry out network traffic analysis [21]. Schuster *et al.,* develop a novel video identification methodology based on convolutional neural networks. The evaluation results on video titles streamed by YouTube, Netflix, Amazon and Vimeo prove that the method has excellent identification performance for videos. In 2019, Liu *et al.,* propose FS-Net [20], which is a recurrent neural network based approach to encrypted SSL/TLS traffic classification. Specifically, FS-Net takes the packet length sequence of each flow as the input, and classifies it to recognize the mobile application carried by each TLS flow. FS-Net adopts a multi-layer encoder-decoder structure, and thus it is a novel deep learning-based method in this field. In 2020, Rezaei *et al.,* design a novel deep learning model [22], which uses adjacent flows to learn the order and patterns of multiple flows to better classify the mobile applications that generate these flows. Also, in 2020, Zheng *et al.,* propose RBRN [19], a network traffic classification model based on convolutional neural networks (CNN). RBRN learns representative features from the raw flows and then classifies them in a unified framework. In 2021, Xiao *et al.,* propose EBSNN [23], an extended byte segment neural network to classify network traffic. EBSNN is built based on the recurrent neural networks with the attention mechanism, and it can classify flows by examining the first few packets. Also in 2021, Nascita *et al.,* propose MIMETIC-ENHANCED, an evolved multimodel deep learning architecture for network traffic classification at biflow level [24]. In 2022, Wang *et al.,* propose Ulfar [25], a multi-scale feature attention approach to network traffic classification. Ulfar takes the byte sequence of each TCP/UDP flow as the input, and uses convolutional neural networks (CNN) as the building block for building the deep learning models. Also in 2022, Zhao *et al.,* propose Festic [26] a few-shot learning based approach to network traffic classification. Festic adopts a hierarchical convolution structure, and thus it can extract both packet-level and flow-level classification features.

C. Discussions

The above work has made important research contributions to the field of encrypted traffic classification. However, these researches do not pay enough attention to the network traffic generated by a large number of "cloud" applications in the Internet, and the new challenges that the "cloud" application traffic brings to encrypted traffic classification.

III. Flow Processing

As shown in Fig. 1, the flow processing module takes the packet sequence of each TLS flow as the input, and it aims to

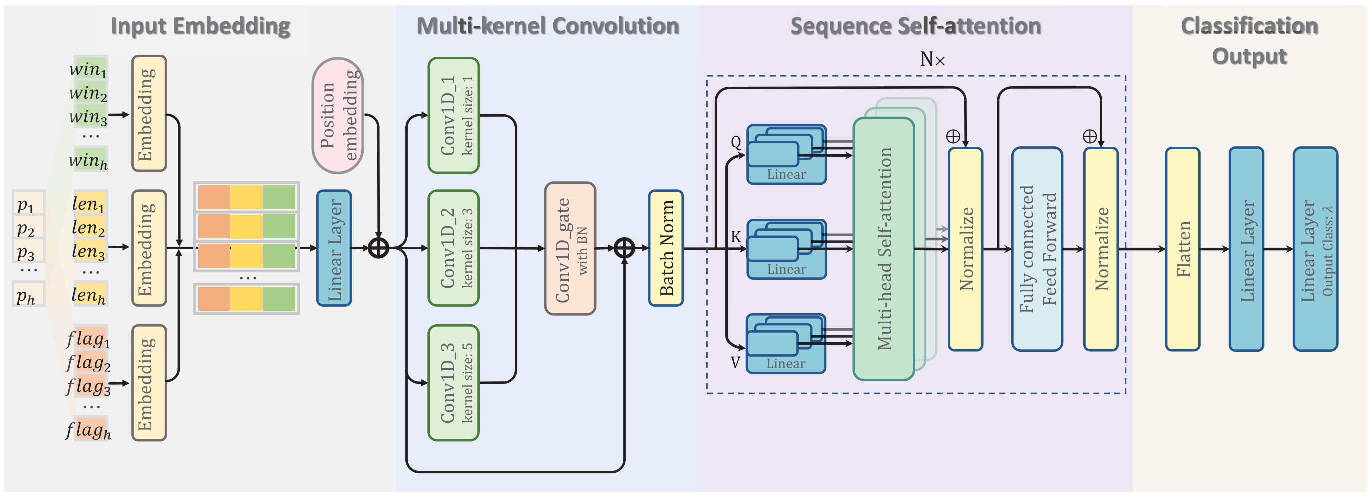


Fig. 2. Structure of neural training.

extract three attribute sequences (*i.e.,* packet length sequence, packet window size sequence and packet TCP flag sequence) of each TLS flow for the next neural training module.

A. Our Findings

Recall that the TLS packet sequences generated by each mobile application are controlled by its application state automata, and each application maintains its own application state automata. To this end, classifying TLS flows from the packet sequences is an effective method in practice. In this paper, instead of using the message type sequences in prior arts [16]–[18], we propose to use the packet sequence to effectively characterize the state transitions of each TLS flow. The advantage of using the packet sequences is that we do not need to assemble IP packets into TLS messages. Therefore, our flow processing scheme allows more scalable analysis in practice.

B. Packet Sequence of a TLS Flow

Let $p_1, p_2, \cdots, p_i, \cdots, p_h$ denote the packet sequence of a TLS flow (denoted $\mathcal{F}$). Notice that we cannot directly use the packet sequence for deep learning modeling. Therefore, for any packet p_i ($i \in [1, 2, \cdots, h]$), this module first extracts three important fields from each packet, including the "*Total Length*" field in the IP header, and the "*Window Size*" field and the "*TCP Flags*" filed in the TCP header. For packet p_i, we use len_i to denote its packet length, win_i to denote its window size, and $flag_i$ to denote its TCP flags. Then, for flow $\mathcal{F}$, it can be represented into three new sequences, S_{len}, S_{win} and S_{flag}, where $S_{len} = \{len_1, len_2, \cdots, len_i, \cdots, len_h\}$, $S_{win} = \{win_1, win_2, \cdots, win_i, \cdots, win_h\}$, and $S_{flag} = \{flag_1, flag_2, \cdots, flag_i, \cdots, flag_h\}$. In this paper, we extract the above three attribute sequences for each TLS flow to represent the packet sequence of the TLS flow. These three attribute sequences (S_{len}, S_{win} and S_{flag}) will be fed into the subsequent module, neural training.

IV. Neural Training

As shown in Fig. 2, the input to the neural training module are the labeled TLS flows, where the packet sequence of each flow is extracted by the flow processing module, and the output of the neural training module is the predicted application labels of these TLS flows. The neural training module has four key components, including input embedding, multi-kernel convolution, sequence self-attention, and classification output. Next, we introduce each key component of our neural network in detail. The source code is available at: https://github.com/auto-ctrl/NeuTic.

A. Input Embedding

For each flow, the purpose of input embedding is to fuse the three attribute sequences from the flow processing module into one feature sequence. First of all, as shown in Fig. 2, for flow $\mathcal{F}$, we feed S_{len}, S_{win}, and S_{flag} three attribute sequences into three embedding layers, respectively. Notice that the embedding layer can embed each element in an input sequence into a vector. Therefore, we will obtain three new embedding sequences, denoted E_{len}, E_{win} and E_{flag}. Specifically, taking packet length sequence S_{len} as an example, $S_{len} = \{len_1, len_2, \cdots, len_i, \cdots, len_h\}$ and it has h elements. The corresponding embedding layer converts each element len_i ($i \in [1, h]$) into a d-dimension vector e_i. For S_{len}, the embedding layer outputs an embedding sequence $E_{len} = \{e_1, e_2, \cdots, e_i, \cdots, e_h\}$, where $e_i \in \mathbb{R}^d$ ($i \in [1, h]$). In this paper, the embedding dimensions of all the three embedding layers are set to be d. Then, we concatenate these outputs (*i.e.,* E_{len}, E_{win} and E_{flag}), and apply a linear transformations to produce the following tensor $\boldsymbol{I} \in \mathbb{R}^{h \times d}$.

$$\boldsymbol{I} = (E_{len} \oplus E_{win} \oplus E_{flag}) \cdot W^{\boldsymbol{I}} + \boldsymbol{b}_{embed} \tag{1}$$

where $W^{\boldsymbol{I}} \in \mathbb{R}^{3d \times d}$ is the parameter matrix of linear transformation, $\boldsymbol{b}_{embed} \in \mathbb{R}^{h \times d}$ is the bias parameter of linear transformation, and $\oplus$ is the concatenation operator. Finally, different from recurrent neural networks and convolutional neural networks, the neural networks used in this paper are based on the self-attention mechanism [27], which cannot naturally make use of the position information of the elements in a given input sequence. Note that without position information, the output of the self-attention network would be the same if two elements in the input sequence exchange their positions. For example, "$a \rightarrow b \rightarrow \underline{c} \rightarrow d \rightarrow e \rightarrow \underline{f}$" and "$a \rightarrow b \rightarrow \underline{f} \rightarrow d \rightarrow e \rightarrow \underline{c}$". Therefore, we use "positional encodings", which explicitly encode the relative/absolute positions of the inputs as vectors. In the paper, we use the following equation to compute the positional encodings (PE):

$$\begin{aligned} PE_{(pos,2j)} &= \sin(pos/10000^{2j/d}) \\ PE_{(pos,2j+1)} &= \cos(pos/10000^{2j/d}) \end{aligned} \tag{2}$$

where "pos" denotes the position index in the input sequence, and "j" denotes the dimension index. In addition, "sin" and "cos" represent the sine and cosine functions, respectively. Finally, the "positional encodings" will be added to $\boldsymbol{I}$, and the new result is denoted by $\boldsymbol{I}'$.

B. Multi-Kernel Convolution

The following component is multi-kernel convolution. For flow $\mathcal{F}$, multi-kernel convolution takes $\boldsymbol{I}'$ from input embedding as the input, and this component outputs a new sequence tensor $\boldsymbol{F}$. Specifically, multi-kernel convolution is composed of 3 parallel 1D-convolutional branches, one gate convolution, and one residual connection. First of all, multi-kernel convolution uses 3 parallel 1D-convolutional branches to produce three different feature maps for flow $\mathcal{F}$, denoted Brn_1, Brn_2 and Brn_3, where the k-th branch is composed of an 1D-convolutional layer of kernel size $2 * k - 1$ and stride 1 as illustrated in Fig. 2. The advantage of the parallel convolutions is that they help to consider each input sequence at different scales. Then, multi-kernel convolution concatenates these outputs (*i.e.,* Brn_1, Brn_2 and Brn_3), and applies a gate convolution layer to produce the following tensor $\boldsymbol{G} \in \mathbb{R}^{h \times d}$.

$$\boldsymbol{G} = \boldsymbol{Conv1d}(Brn_1 \oplus Brn_2 \oplus Brn_3) \tag{3}$$

where $\oplus$ is the concatenation operator. The gate convolution layer is actually another 1-D convolutional layer, and its input is $3 * d$ channels and its output is d channels. For the final result of multi-kernel convolution, we introduce a residual connection layer, which adds $\boldsymbol{G}$ and $\boldsymbol{I}'$ to get the output $\boldsymbol{F}$.

C. Sequence Self-Attention

Notice that sequence self-attention aims to establish the relationship between any two elements in a given sequence, and it is composed of a stack of L identical layers, where each layer contains two sub-layers. The first sub-layer is a multi-head self-attention mechanism, and the second sub-layer is a fully connected feed-forward network. We show the structure of sequence self-attention in Fig. 2. Next, we introduce the two sub-layers in detail.

1) Multi-Head Self-Attention Mechanism: Given the input sequence $\boldsymbol{F} = f_1, f_2, \cdots, f_i, \cdots f_h$ from the multi-branch convolution, where $f_i \in \mathbb{R}^d$ be the d-dimensional vector, the multi-head self-attention mechanism [27] outputs a new sequence of continuous representations $\boldsymbol{O} = o_1, o_2, \cdots, o_i, \cdots, o_h$, where $o_i \in \mathbb{R}^d$. The miracle here is that the multi-head self-attention mechanism can achieve the construction and extraction of the relationship between any two elements in an input sequence through three transformations of the input itself, named "*Query*", "*Key*" and "*Value*". Specifically, to begin with, this mechanism transforms the input $\boldsymbol{F}$ into three matrixes $\boldsymbol{Q}$ (*Query*), $\boldsymbol{K}$ (*Key*) and $\boldsymbol{V}$ (*Value*) using three trainable parameter matrixes (*i.e.,* $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$ and $\boldsymbol{W}^V$), respectively. $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ can be formulated as follows:

$$\begin{cases} \boldsymbol{Q} = \boldsymbol{F} \cdot \boldsymbol{W}^Q \\ \boldsymbol{K} = \boldsymbol{F} \cdot \boldsymbol{W}^K \\ \boldsymbol{V} = \boldsymbol{F} \cdot \boldsymbol{W}^V \end{cases} \tag{4}$$

Notice that $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, and $\boldsymbol{W}^V$ are the model parameters that this sub-layer needs to train. For the above parameter matrices, we have $\boldsymbol{W}^Q \in \mathbb{R}^{d \times d_q}$, $\boldsymbol{W}^K \in \mathbb{R}^{d \times d_k}$ and $\boldsymbol{W}^V \in \mathbb{R}^{d \times d_v}$, where $d_k = d_q$. In general, an attention function can be described as mapping query $\boldsymbol{Q}$ and key-value pairs ($\boldsymbol{K}, \boldsymbol{V}$) into a new output $\boldsymbol{O}'$. Next, according to $\boldsymbol{K}$ and $\boldsymbol{Q}$, we can compute attention matrix $\boldsymbol{A} \in \mathbb{R}^{h \times h}$, and $\boldsymbol{A}$ can be computed as follows.

$$\boldsymbol{A} = Softmax\left(\frac{\boldsymbol{Q} \cdot \boldsymbol{K}^T}{\sqrt{d_k}}\right) \tag{5}$$

where "$Softmax$" denotes the softmax activation function. It is worthy to notice that attention matrix $\boldsymbol{A}$ reflects the relationship between every two elements in input sequence $\boldsymbol{F}$. Finally, we use attention matrix $\boldsymbol{A}$ to extract information from $\boldsymbol{V}$ to form output $\boldsymbol{O}'$, and the new output sequence $\boldsymbol{O}' \in \mathbb{R}^{h \times d_v}$ can be formulated as follows:

$$\boldsymbol{O}' = Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{A} \cdot \boldsymbol{V} \tag{6}$$

In practice, we can generate a series of $\boldsymbol{O}'$ (*i.e.,* O'_1, O'_2, $\cdots$, O'_{head}) using different parameter matrixes in parallel, and concatenate them together to form the final output $\boldsymbol{O}$, which increases the expressive power of our neural network. The output sequence $\boldsymbol{O}$ of the multi-head self-attention mechanism can be formulated as follows:

$$\boldsymbol{O} = (O'_1 \oplus O'_2 \oplus \cdots \oplus O'_i \oplus \cdots \oplus O'_{head}) \cdot \boldsymbol{W}^O \tag{7}$$

where $\boldsymbol{W}^O \in \mathbb{R}^{(head \times d_v) \times d}$ and $head \times d_v = d$. In addition, we have $O'_i = Attention(\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i)$.

2) Fully Connected Feed-Forward Network: The sub-layer following the multi-head self-attention mechanism is the fully connected feed-forward network, which takes the output $\boldsymbol{O} = o_1, o_2, \cdots, o_i, \cdots, o_h$ ($o_i \in \mathbb{R}^d$) of the previous sub-layer as the input. The fully connected feed-forward network is a two-layer fully-connected neural network consisting of an inner hidden layer of dimensionality $d_{ff} = 2048$ and a d-dimensional output layer activated by ReLU [28]. The fully connected feed-forward network can be formulated as follows.

$$FFN(\boldsymbol{O}) = max(0, \boldsymbol{O} \cdot W_1 + b_1)W_2 + b_2 \tag{8}$$

In addition, as shown in Fig. 2, notice that there is an Add&Norm layer between each sub-layer, which involves a residual connection [29] followed by a layer normalization [30]. Specifically, the residual connection provides a shortcut path between sub-layers, and it is basically just taking the input and adding it to the output of the sub-layer. The residual connection is used for an additional gradient path when building deeper neural network models. The layer normalization is a normalization method similar to batch normalization [31], and it is commonly used in deep learning, especially in recurrent neural networks.

D. Classification Output

The last component of our neural network is classification output, which consists of one flatten layer and two linear layers (that is the fully-connected layers).

1) Flatten Layer: In NeuTic, we use a linear layer to do a R-way classification, where R denotes the number of application labels. Notice that the linear layer only accepts 1-dimension vectors as its input. However, our current output from sequence self-attention is a 2-dimension tensor. Therefore, before going through the linear layer, we first flatten our 2-dimension outputs into 1-dimension. Specifically, in our neural network, the output (shape $(h,\ d)$) from sequence self-attention is flattened into vectors of shape $(1,\ h \times d)$.

2) Linear Layers: Before the linear layers, we do a dropout operation with a rate of 0.3 to avoid over-fitting. The outputs to the two linear layers are R_1, and R, respectively, where we have $R_1 = 512$ and R is adjusted according to the specific classification task. The final linear layer uses "Softmax" as the activation function.

V. Classification Phase

Recall that as shown in Fig. 1, the classification phase aims at classifying new incoming TLS flows according to the TLS traffic classification model produced in the training phase. There are two key functional components in the classification phase, including the flow processing module and the neural classification module.

First, the flow processing module also extracts the "*Total Length*" field, the "*Window Size*" field, and the "*TCP Flags*" field information from the first h packet of each TLS flow, and converts them into corresponding sequences. Then, the neural classification module uses the TLS traffic classification model generated by the neural training module to make decisions on each new TLS flow, so that each unlabeled TLS flow can be assigned a proper application label.

VI. Experimental Evaluation

In this section, we aim at assessing the effectiveness of NeuTic in the classification of encrypted TLS traffic on cloud platforms. To this end, we introduce real-world application traces for the evaluation. In the following subsections, we first introduce the application traces used in our experiments. Then, we define proper metrics to evaluate the performances of our method and other prior approaches. Finally, we investigate how NeuTic's performance is affected by different parameter settings.

TABLE I
DATASET-I

Application	Flows	Packets	Bytes	Main Use	Corporation
Amap	16,770	490K	590.43M	Map	**Alibaba**
Eleme	10,337	5,275K	5.85G	Food Ordering	
Taobao	10,485	314K	312.72M	Online Shopping	
Taopiaopiao	10,318	103K	63.92M	Short Video Sharing	
Youku	10,883	1,276K	1.57G	Video	
Baidumap	39,186	1075K	1.21G	Map	**Baidu**
Duxiaoshi	10,103	295K	227.64M	Short Video Sharing	
Haokan	10,220	753K	802.75M	Short Video Sharing	
Baidusearch	10,034	1409K	1.56G	Search	
Youjia	10,339	199K	174.24M	Car News	
Baiduyuedu	20,724	200K	163.59M	Reading	
Dongchedi	10,173	3,941K	5.06G	Car News	**ByteDance**
Douyin	13,273	6,008K	8.03G	Short Video Sharing	
Ixigua	11,596	1,192K	1.26G	Video & Short Video Sharing	
Toutiao	13,570	1,866K	2.24G	News	

A. Data Set

In this paper, we have selected multiple mobile applications from three companies, *i.e.,* Alibaba, Baidu and ByteDance, to evaluate the effectiveness of NeuTic. Next, we report our datasets in detail.

1) ***Dataset-I****:* In *Dataset-I*, we collect a variety of TLS traffic generate by 15 mobile applications from three corporations, and the three corporations are Alibaba, Baidu, and ByteDance. Specifically, (1). first, for mobile applications designed by Alibaba, we select 5 popular mobile applications, which are Amap, Eleme, Taobao, Taopiaopiao and Youku. The type of these applications include map, food ordering, online shopping, short video and video. These applications comprise a variety of categories, and see Table I for details. (2). Secondly, for mobile applications designed by Baidu, we choose a total of 6 popular mobile applications, including Baidumap, Duxiaoshi, Haokan, Baidusearch, Youjia and Baiduyuedu. The type of these applications include map, short video, video, search, *etc*. All of these are well-known applications developed by Baidu corporation. (3). Thirdly, for mobile applications designed by ByteDance, we choose 4 popular mobile applications, including Dongchedi, Douyin, Ixigua and Toutiao. The type of these applications include video, short video, news, *etc*. For each application trace, we report the total amount of flows, packets, bytes and its main use in Table I.

2) ***Dataset-II****:* *Dataset-II* aims to evaluate the classification performance of NeuTic on mobile applications with the same usage category. In *Dataset-II*, we use TLS traffic generate by 6 mobile applications, including Taopiaopiao, Youku, Duxiaoshi, Haokan, Douyin and Ixigua. The main uses of all these mobile applications are short video sharing or video. In addition, Taopiaopiao and Youku are from Alibaba, Duxiaoshi and Haokan are from Baidu, and Douyin and Ixigua are from ByteDance. It is worth noting that the two applications from the same company are similar applications developed under the same certificate. For each application trace, we report its main use, the number of flows and the company it is affiliated with in Table II.

3) Summary: For all the above applications, we collect our application traffic traces under a controlled operating environment. Specifically, our in-laboratory test-bed setup for data collection is as follows. Firstly, we use three smartphones as the end devices to generate mobile application traces, including one iPhone SE2 (running on iOS 14.0), one iPhone 11

TABLE II
DATASET-II

Application	Flows	Main Use	Corporation
Taopiaopiao	10,318	Short Video Sharing	**Alibaba**
Youku	10,883	Video	
Duxiaoshi	10,103	Short Video Sharing	**Baidu**
Haokan	10,220	Short Video Sharing	
Douyin	13,273	Short Video Sharing	**ByteDance**
Ixigua	11,596	Video & Short Video Sharing	

(running on iOS 13.7), and one iPhone 11 (running on iOS 14.3). We execute each mobile application on the smartphones separately. Secondly, our smartphones are connected to a workstation over an access point. The workstation directly connects to the Internet, and it acts as the gateway device. Thirdly, we run tcpdump on this workstation to record the traffic generated by each mobile application.

As with the previous works, in this paper, we consider the unidirectional flows, that is, flows that from a server to a client as well as flows that from a client to a server. For each application in the datasets above, we randomly pick 10K TLS flows. We notice that the TLS versions used by the TLS flows we captured are either TLS1.2 or TLS1.3. Therefore, we have a total of 150,000 samples for *Dataset-I*, and a total of 60,000 samples for *Dataset-II*. For all the two datasets, we carry out 5-fold cross validation to enhance the reliability of our experiments. In addition, for each dataset, the ratio of the training set, the validation set and the testing set is 3:1:1.

B. Evaluation Metrics for Effectiveness

Once an application traffic classification model has been designed, its performance must be evaluated, and proper evaluation metrics must be defined. For a specific application r under analysis, we define the following metrics for further analysis:

- TP_r*, True Positives for application* r: the set of flows where each flow is classified by NeuTic as belonging to application r and is indeed generated by application r.
- FP_r*, False Positives for application* r: the set of flows where each flow is classified by NeuTic as belonging to application r but is not generated by application r.
- FN_r*, False Negatives for application* r: the set of flows where each flow is classified by NeuTic as not belonging to application r but is indeed generated by application r.

Next, we define the following three metrics for application r to quantitatively evaluate the effectiveness of NeuTic:

$$recall_r = \frac{|TP_r|}{|TP_r| + |FN_r|},$$

$$precision_r = \frac{|TP_r|}{|TP_r| + |FP_r|} \tag{9}$$

$$F\!-\!measure_r = 2 * \frac{recall_r * precision_r}{recall_r + precision_r} \tag{10}$$

In addition, for the scenario of multi-category classification, we need to define the accuracy metric to assess the performance of the whole method.

$$Accuracy\ (ACC) = \frac{\sum_{r=1}^{R} recall_r}{R} \tag{11}$$

where R denotes the total number of applications for application classification. Furthermore, we also need to present our experimental results of multi-application classification through a Confusion Matrix, a visualization tool commonly used in the field of artificial intelligence, where each column of the matrix represents instances in a predicted class label, and each row of the matrix represents instances in an actual class label. It is quite convenient to see if the proposed method confuses two classes (*i.e.,* one class is erroneously marked as the other) by means of a Confusion Matrix.

C. Parameter Selection

The training phase of NeuTic involves the following parameters:

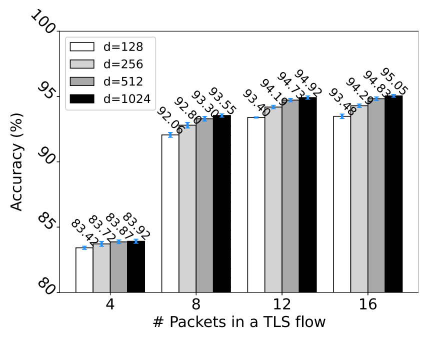
(h): The number of the first few packets of a TLS flow used for classification.
(d): The dimension of the embedding vector for the embedding layer.
(L): The number of times sequence self-attention is repeated.

Next, we present our experimental results for varying values of the above parameters h, d and L.

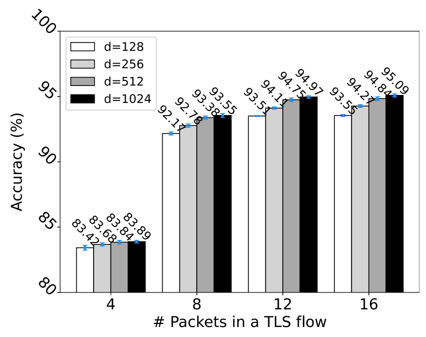
1) Number of the First Few Packets of a TLS Flow Used for Classification, (h): In NeuTic, remember that we use the first h packets of a TLS flow for application traffic classification. Notice that the parameter h affects the performance of NeuTic in two ways. First, the accuracy values of NeuTic generally decrease for lower values of h. Second, we also notice that the time overhead of the application classification model also increases with the number packets considered per TLS application flow. Taking into account the time overhead and the classification accuracy, in our following evaluation experiments for NeuTic, we vary the range of parameter $h \in \{4, 8, 12, 16\}$ as a convenient trade-off.

2) Dimension of the Embedding Vector for the Embedding Layer, (d): Recall that in Section IV, we have three embedding layers in input embedding. For each embedding layer, we use d to denote the dimensions of the embedding vector for the embedding layer. In practice, we experimentally find that the parameter d has an impact on the classification performance of NeuTic. Therefore, in the following evaluations, we vary the ranges of $d \in \{128, 256, 512, 1024\}$ to investigate how NeuTic's performance is affected by different parameters settings.

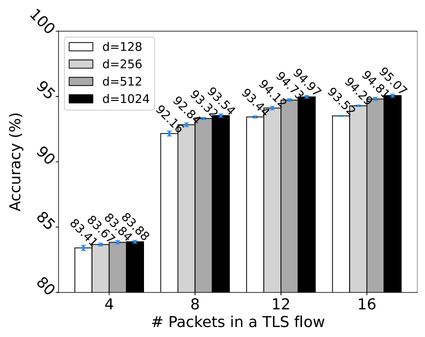
3) The Number of Times the Sequence Self-Attention Is Repeated, (L): The depth of the neural network is very important for classification, and the classification accuracy can gain from considerably increased depth. In practice, we can stack one sequence self-attention on top of another sequence self-attention to build a deeper classification model. For NeuTic, we use parameter L to denote the number of times sequence self-attention is repeated. Notice that, with a small value of L, it looks like that we may not have enough model parameters and therefore will suffer from an under-fitting problem. With a larger value of L, we may have too many model parameters and we encounter an over-fitting problem. Therefore, in the following evaluations, we carry out our



(a) $L = 1$

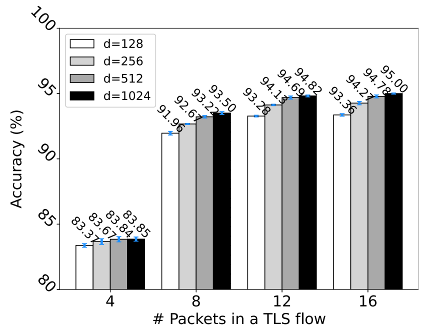


(b) $L = 2$

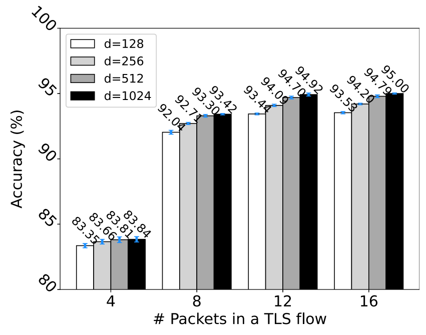


(c) $L = 3$

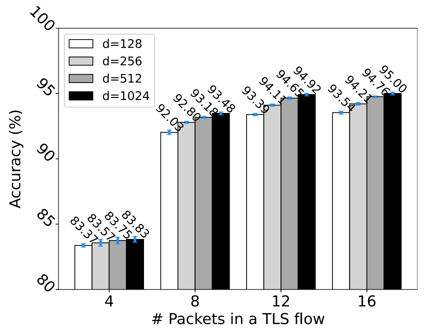
Fig. 3. Accuracy (%) on the validation set of Dataset-I (in the form of mean±standard deviation).



(a) $L = 1$



(b) $L = 2$



(c) $L = 3$

Fig. 4. Accuracy (%) on the testing set of Dataset-I (in the form of mean±standard deviation).

experiments for $L \in \{1, 2, 3\}$ to investigate how NeuTic's performance is affected by different parameters settings.

4) Other Settings: We have trained our neural networks with Adam (adaptive moment estimation) optimizer with a mini-batch of 1024 to optimize the loss function. In addition, the initial value of our learning rate is 1e(−4), and the patience value of the model on the validation set is 30 epoches, then the learning rate is adjusted to 1/10 of the original, and the minimum learning rate is 1e(−5). In this paper, the optimal epoch we choose for the training of each deep learning model is the number of epochs for which each model has a maximum accuracy value on the validation set.

D. Effectiveness Results for Dataset-I

Fig. 3 plots the accuracy values for varying values of parameter h, d, and L on the validation set of *Dataset-I*. Specifically, Fig. 3(a), Fig. 3(b), and Fig. 3(c) three figures plot the accuracy values for varying values of h and d for $L = 1$, $L = 2$ and $L = 3$, respectively. For different parameter settings, we report that the accuracy values vary in the range of $83.41(\pm 0.20)\%$ – $95.09(\pm 0.12)\%$. As shown in Fig. 3, we clearly notice that the accuracy values generally degrade for lower values of h for all possible values of d and L. In addition, we also observe that the accuracy of NeuTic generally degrades for lower values of d. Next, let's take a closer look at the detailed experimental results in Fig. 3. For any fixed values of d and h, we observe the trend that the classification performance of NeuTic with $L = 2$ in most cases outperforms those of $L = 1$ and $L = 3$. In addition, we also notice that when $h = 12$, the classification accuracy is relatively stable, and increasing the value of h does not significantly improve the classification accuracy. In Fig. 4, we present the experimental evaluation results of all the neural network models on the testing set under different parameter settings. The accuracy values vary in the range of $83.35(\pm 0.16)\%$ – $95.00(\pm 0.05)\%$. Based on the above analysis and considering the classification effectiveness and classification efficiency, we choose $d = 1024$, $h = 12$, and $L = 2$ to carry out subsequent evaluation experiments.

Next, for all the 15 applications from Alibaba, Baidu and ByteDance, we report NeuTic's confusion matrix of the classification results on *Dataset-I* in Fig. 5 in the case of $d = 1024$, $h = 12$, and $L = 2$. As shown in the figure, we notice that NeuTic's $recall$ values vary in the range of 84.15% – 99.28% for all the applications. Furthermore, NeuTic achieves an average ACC of 94.92% on *Dataset-I*. From the confusion matrix, we have two important findings. (1). First, as shown in Fig. 5, for applications from different companies, we notice that NeuTic has a very good classification performance. That is to say, NeuTic will not mistake TLS flows generated by one company's (such as Alibaba) application for TLS flows generated by another company's (such as Baidu) application. (2). Second, for apps from the same company, NeuTic performed very well in classifying TLS flows generated by applications from Alibaba and Baidu, while NeuTic's classification performance for TLS flows generated by applications from ByteDance decreases slightly. For example, Ixigua and Toutiao

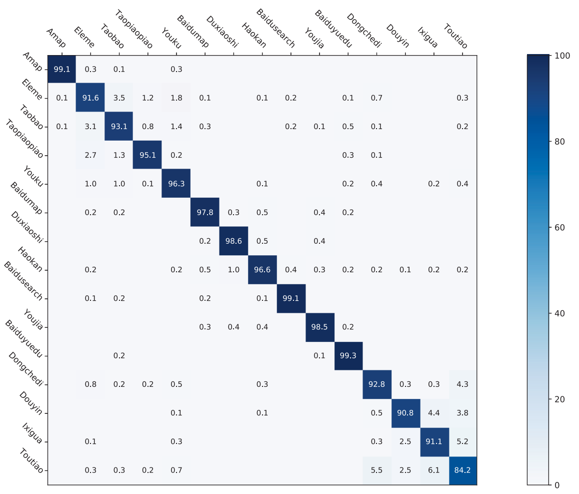
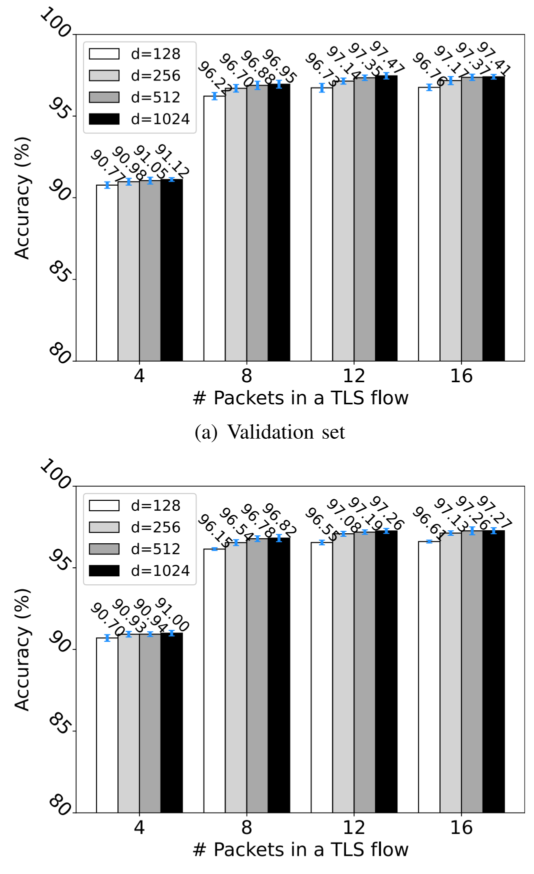


Fig. 5. Confusion matrix of NeuTic on Dataset-I.



(a) Validation set

(b) Testing set

Fig. 6. Accuracy (%) on Dataset-II (in the form of mean±standard deviation).

from ByteDance, we observe that Ixigua traffic is sometimes confused as Toutiao traffic and vice versa, where Ixigua has a $recall$ of 91.08% and Toutiao has a $recall$ of 84.15%.

E. Case Study for Video Applications

Recall that *Dataset-II* is a data set of TLS flows generated by six video/short video mobile applications. Based on the experience gained on *Dataset-I*, in this part we choose a fixed value for L, where $L = 2$. Therefore, in the following evaluations, we only vary the ranges of parameter d and parameter h on *Dataset-II*. More concretely, Fig. 6(a) and Fig. 6(b) show the plots of accuracy for varying values of h and d on the validation set and the testing set of *Dataset-II*, respectively. From Fig. 6(a), we observe that the accuracy values vary in the range of 70.77% – 97.47% for all possible values of h and d. For *Dataset-II*, the optimal parameter values on the validation dataset are $h = 12$ and $d = 1024$, and the corresponding accuracy value is 97.47%. Notice that, like the trend on *Dataset-I*, we observe that NeuTic's accuracy values on *Dataset-II* also generally degrade for lower values of h and d. We also find that compared with $h = 8, 12, 16$, the classification accuracy for $h = 4$ drops a lot for all possible values of d. In addition, the parameters ($h = 12$, $d = 1024$) that perform best on the validation set achieves an average ACC of 97.26% on the testing set.

In Fig. 7, we report the confusion matrix of the classification results on the testing dataset of *Dataset-II*. Specifically, we notice that the $Recall$ values are 99.0% for Taopiaopiao, 99.1% for Youku, 99.1% for Duxiaoshi, 98.0% for Haokan, 93.2% for Douyin, and 95.1% for Ixigua. From the figure, we have two findings. (1). We find that the confusion matrix on *Dataset-II* has good diagonalization performance, which is consistent with our excellent accuracy results. Although the six categories of applications in *Dataset-II* are all video or short video applications, NeuTic's classification performance is still very good. (2). Furthermore, compared to the other four applications, we find that NeuTic performs slightly worse on Douyin and Ixigua, and the TLS flows of these two types confuse each other.

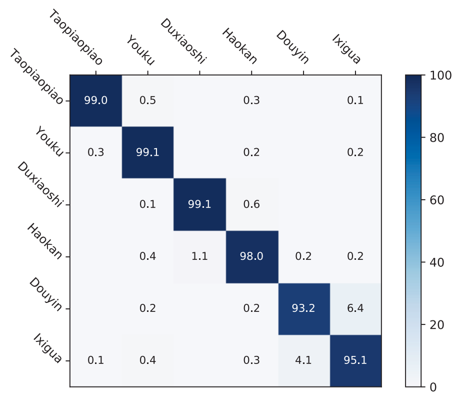


Fig. 7. Confusion matrix of NeuTic on Dataset-II.

F. Interpretability of Our Proposed Method

In this subsection, we use "Lime" [32] to help us to understand what features are more helpful to identify the fingerprint of encrypted TLS traffic. Lime is an open source tool, which can explain the predictions of any classier. Specifically, the input to our neural network model is the three attribute sequences of each flow, including the packet length sequence, the packet window size sequence, and the packet TCP flag sequence. Lime analyzes our neural network model and gives the importance value of each element in the three attribute

TABLE III
AN EXAMPLE OF LIME'S INTERPRETATION FOR THE PACKET SEQUENCE OF "ELEME"

Packet Field		Packet Sequence											
		p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}
PACKET LENGTH	*Value*	488	488	115	46	85	85	93	114	82	78	1492	294
	Importance	2.4111	1.7259	0.9493	2.4054	-0.5142	-0.0820	0.6827	-0.5619	0.2600	0.7242	0.6348	0.0802
PACKET WINDOW SIZE	*Value*	20486	20486	20480	20480	20480	20480	20478	20478	20478	20480	20480	20479
	Importance	1.9564	1.6655	1.6701	0.7915	0.8806	0.6917	-0.2338	0.1726	0.0114	1.1389	0.4611	-0.0320
PACKET TCP FLAGS	*Value*	24	24	24	24	24	24	24	24	24	24	16	16
	Importance	0.0062	0.1509	-0.2987	0.4985	-0.0387	0.4687	0.1080	-0.8745	-0.0048	0.1803	0.0809	0.3303

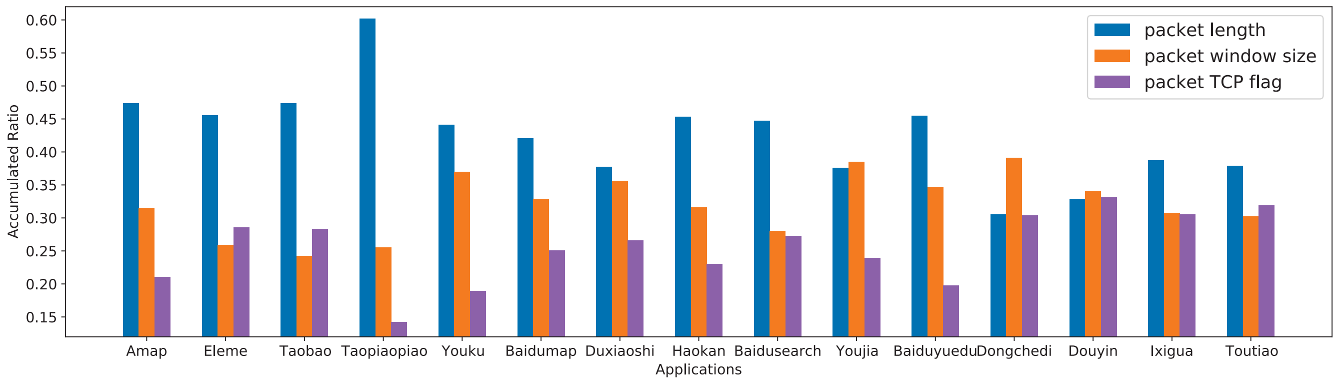


Fig. 8. The importance of different features.

sequences to encrypted traffic classification. The higher the importance value, the greater the contribution of this element to encrypted traffic classification. In this part of the study, we use the first 12 packets of each flow, so the packet sequence of each flow has a total of 36 elements ($=$ 12 packets * 3 attributes). In Table III, we give an example of the results interpreted using Lime. For the packet sequence of each flow, we select the elements with the top 12 importance values to carry out follow-up experimental analysis. Next, we try to interpret the classification results of our neural network model on *Dataset-I*.

1) Which Features Are More Important?: In Fig. 8, we show the importance of three different features (*i.e.,* packet length, packet window size, and packet TCP flag) for each application.

From the above figure, we have the following three findings: (1). First of all, for most applications, the packet length feature has a more important classification contribution than the other two features. Therefore, the packet length feature should be considered first when designing an encrypted traffic classification model. (2). Secondly, the packet window size feature and the packet TCP flag feature are also very helpful for encrypted TLS traffic classification. For different applications, the classification contributions of these two features are different, that is, the packet window size feature is more important for some applications (such as Amap and Youku), and the packet TCP flag feature is more important for other applications (such as Eleme and Taobao). (3). Thirdly, we find that the packet TCP flag feature is not important for maps (*e.g.* Amap and Baidumap) and videos (*e.g.* Taopiaopiao, Youku, Duxiaoshi and Hankan) from Alibaba and Baidu. However, for mobile applications from ByteDance, the packet TCP flag feature is still important.

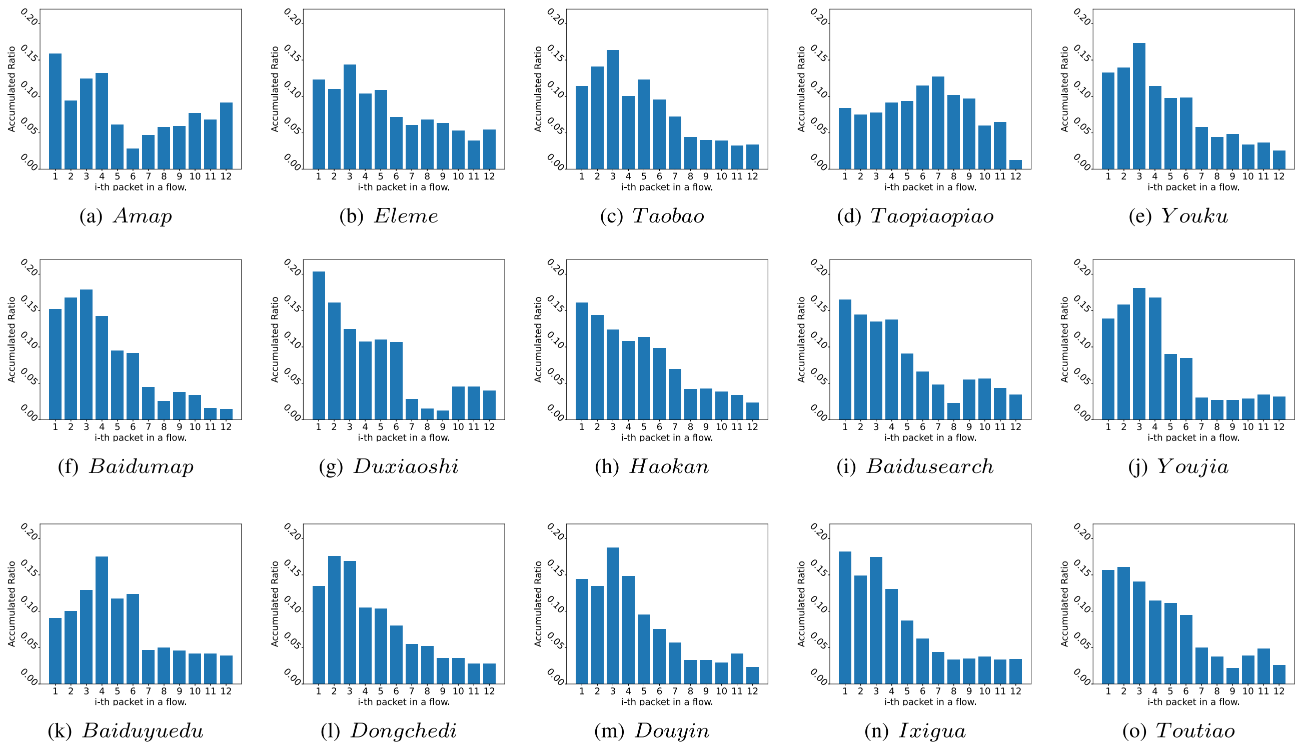
2) Which Packets Are More Important?: In Fig 9, for all the 15 applications, we separately show the importance of the first 12 packets of the flow for classifying encrypted TLS traffic.

From Fig. 9, we have the following three findings: (1). First of all, the importance of packets in the flows varies from application to application. (2). Secondly, the first four packets are very important for encrypted TLS traffic classification. This finding is consistent with our previous experimental results, where using only the first four packets of each flow can achieve a classification accuracy of about 83% on *Dataset-I*. (3). Thirdly, the packets after the first four packets are also helpful for encrypted TLS traffic classification. Increasing the value of parameter h within a certain range can improve the classification accuracy.

G. Dealing With Traffic Obfuscation

In practice, one way to obfuscate a TLS flow is to pad all the packet lengths of the flow to the same length, such as the MTU (Maximum Transmission Unit) value. In this subsection, we test NeuTic for its ability to identify such obfuscated TLS flows. We adjust NeuTic to adapt the detection of obfuscated TLS flows. Specifically, we feed the output of the last layer of our neural network into a softmax layer. The output of the softmax layer is an R-dimensional tensor, and we select a maximum value in the tensor, denoted as $value_{max}$. Then, we set a threshold τ. For each flow to be tested, if $value_{max}$ is greater than τ, NeuTic considers the flow to be a flow of a known class, and if $value_{max}$ is less than τ, NeuTic considers the flow to be a obfuscated flow.

In the experiment, we convert all the TLS flows in *DataSet-I* into obfuscated flows, denoted as *DataSet-I-confusion*. We use



(a) *Amap* (b) *Eleme* (c) *Taobao* (d) *Taopiaopiao* (e) *Youku*

(f) *Baidumap* (g) *Duxiaoshi* (h) *Haokan* (i) *Baidusearch* (j) *Youjia*

(k) *Baiduyuedu* (l) *Dongchedi* (m) *Douyin* (n) *Ixigua* (o) *Toutiao*

Fig. 9. The importance of different packets in a flow.

TABLE IV
EXPERIMENTAL RESULTS OF DEALING TRAFFIC OBFUSCATION

Recall% (Obfuscation)	Precision% (Obfuscation)	F-measure% (Obfuscation)
90.81(±2.01)	86.66(±0.28)	88.62 (±0.94)

DataSet-I and *DataSet-I-confusion* to test NeuTic's ability to identify obfuscated TLS flows. For threshold τ, in our experiment, we set a tentative value of 0.99. The evaluation results are shown in Table IV. We notice the recall, precision, and F-measure of NeuTic for the obfuscated flows are 90.81%, 86.66%, and 88.62%, respectively. It can be seen from the experiment that NeuTic can accurately reject most of the obfuscated TLS flows. This is only a preliminary attempt, and the future work direction is to design a better obfuscated flow filtering algorithm.

H. Discussion

If the developer changes the flow style of a mobile application, which usually means a major upgrade to the version of this application, then our proposed approach may also lose the ability to directly classify the traffic generated by this newly upgraded application. Researchers can consider using DNS resolution records to associate the flow of the upgraded application with the flow of the original application. The research on the classification of new applications is a cutting-edge issue in this field.

VII. Comparisons With Existing State-of-the-Art Methods

In this section, we compare the performance of NeuTic with two existing state-of-the-art methods, including RBRN [19] and FS-Net [20]. Furthermore, we still use the evaluation metrics introduced in Section VI-B to quantitatively evaluate the experimental performances of difference methods.

A. Comparison With RBRN

First of all, we compare NeuTic with the classification algorithm proposed in RBRN. Notice that RBRN uses the byte information in each flow to perform encrypted traffic classification. Specifically, RBRN converts the byte sequence information of each flow into a 2D tensor image and feeds it into the designed deep neural network model. In our comparison experiments, consistent with NeuTic, we also consider the first 12 packets of each flow for RBRN and evaluate RBRN's classification performance on *Dataset-I*. In addition, we mask the IP addresses (both source and destination IP addresses) and TCP port numbers (both source and destination port numbers) in the packets. For *Dataset-I*, columns 2 to 4 of Table V report the cross-validation results of $Recall$, $Precision$ and $F{-}measure$ three metrics of RBRN for each mobile application. Specifically, the cross-validation results of the $Recall$ values vary in the range of 51.79% – 92.84% for different applications, the cross-validation results of the $Precision$ values vary in the range of 53.25% – 95.69% for different applications, and the cross-validation results of the $F{-}measure$ values vary in the range of 51.72% – 94.22% for different applications. We notice that RBRN has an average $Recall$

TABLE V
COMPARISONS OF NEUTIC, RBRN, AND FS-NET ON DATASET-I

Application	*RBRN* *Metrics* Recall (%)	Precision (%)	F-measure (%)	*FS−Net* *Metrics* Recall (%)	Precision (%)	F-measure (%)	*NeuTic* *Metrics* Recall (%)	Precision (%)	F-measure (%)
Amap	92.84 (±2.92)	95.69 (±0.52)	94.22 (±1.40)	98.72 (±0.28)	99.27 (±0.54)	99.00 (±0.27)	99.05 (±0.27)	99.58 (±0.21)	**99.31** (±0.15)
Eleme	64.92 (±9.56)	66.45 (±7.20)	65.22 (±6.41)	88.83 (±1.06)	89.08 (±1.77)	88.94 (±0.48)	91.57 (±0.62)	91.14 (±1.00)	**91.35** (±0.45)
Taobao	68.98 (±9.16)	70.64 (±7.65)	69.41 (±6.39)	91.93 (±0.61)	89.76 (±0.27)	90.83 (±0.26)	93.06 (±0.99)	92.91 (±1.28)	**92.98** (±0.46)
Taopiaopiao	85.21 (±3.24)	88.84 (±1.58)	86.94 (±1.32)	92.21 (±0.46)	97.43 (±0.35)	94.75 (±0.29)	95.11 (±0.99)	97.23 (±0.79)	**96.15** (±0.32)
Youku	73.59 (±5.44)	67.42 (±5.84)	70.36 (±5.62)	94.35 (±1.05)	91.51 (±1.63)	92.89 (±0.57)	96.34 (±0.39)	94.23 (±0.49)	**95.27** (±0.40)
Baidumap	85.65 (±3.51)	88.21 (±7.01)	86.78 (±4.10)	96.09 (±0.29)	96.57 (±0.99)	96.33 (±0.39)	97.82 (±0.25)	98.00 (±0.27)	**97.91** (±0.08)
Duxiaoshi	81.71 (±4.12)	87.71 (±4.89)	84.55 (±3.82)	98.17 (±0.28)	97.70 (±0.32)	97.93 (±0.22)	98.60 (±0.16)	98.15 (±0.63)	**98.38** (±0.25)
Haokan	68.81 (±11.4)	73.75 (±7.45)	71.01 (±8.90)	95.90 (±0.51)	96.40 (±1.03)	96.15 (±0.31)	96.62 (±0.52)	97.50 (±0.40)	**97.05** (±0.26)
Baidusearch	85.39 (±6.11)	85.65 (±8.78)	85.44 (±7.04)	98.36 (±0.22)	97.85 (±0.28)	98.10 (±0.18)	99.05 (±0.19)	98.83 (±0.21)	**98.94** (±0.16)
Youjia	85.39 (±2.70)	82.49 (±10.10)	83.73 (±6.31)	97.58 (±0.37)	97.96 (±0.63)	97.77 (±0.19)	98.46 (±0.37)	98.36 (±0.21)	**98.41** (±0.20)
Baiduyuedu	88.10 (±7.00)	85.55 (±3.91)	86.71 (±4.71)	98.66 (±0.14)	97.67 (±0.39)	98.16 (±0.25)	99.28 (±0.13)	97.97 (±0.39)	**98.62** (±0.18)
Dongchedi	67.65 (±5.76)	66.97 (±7.92)	67.11 (±5.95)	90.45 (±0.25)	89.50 (±1.04)	89.97 (±0.48)	92.83 (±0.77)	91.97 (±0.67)	**92.40** (±0.36)
Douyin	61.83 (±10.3)	64.59 (±7.60)	63.03 (±8.56)	89.05 (±1.19)	89.94 (±1.52)	89.48 (±0.47)	90.81 (±1.14)	93.96 (±0.76)	**92.35** (±0.59)
Ixigua	67.41 (±6.56)	61.23 (±5.31)	64.15 (±5.76)	87.58 (±2.08)	87.53 (±2.13)	87.52 (±0.53)	91.08 (±0.94)	88.85 (±0.81)	**89.95** (±0.56)
Toutiao	51.79 (±9.80)	53.25 (±10.19)	51.72 (±8.01)	81.72 (±1.18)	81.92 (±1.09)	81.81 (±0.45)	84.15 (±1.02)	85.22 (±1.33)	**84.67** (±0.33)
Average	75.28 (±5.48)	75.90 (±5.11)	75.36 (±5.44)	93.31 (±0.12)	93.34 (±0.10)	93.31 (±0.11)	**94.92** (±0.12)	**94.93** (±0.12)	**94.92** (±0.12)

of 75.28(±5.48)%, an average *Precision* of 75.90(±5.11)%, and an average *F−measure* of 75.36(±5.44)%.

In addition, we report the classification confusion matrix of RBRN on *Dataset-I* in Fig. 10. Let's take a closer look at the detailed experimental results for each application. We notice that RBRN's classification approach to improving *Recall* for some applications may reduce *Recall* for other applications. In addition, the classification results of RBRN are not good enough for the following reason. In the design of the deep learning model for encrypted traffic classification, RBRN adopts a neural network structure based on CNN. It is worth noting that the CNN model is good at dealing with the local features of the data. However, it is worth noting that for a flow, the relationship between packets is difficult to be described by local features. As a result, RBRN does not perform well in modeling the relationship between packets in a flow, especially the relationship between distant elements in a packet sequence. Compared to RBRN, NeuTic significantly improves the effectiveness of encrypted TLS traffic classification, where the average *F−measure* increases by about 19.56% on *Dataset-I*. In addition, RBRN is a very complex neural network, which contains dozens of neural network layers, so the algorithm complexity of RBRN is very high, and the whole training process is also very slow.

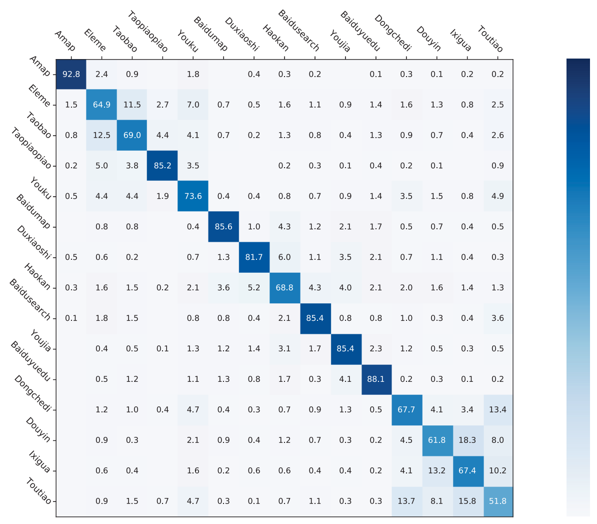


Fig. 10. Confusion matrix of RBRN on Dataset-I.

B. Comparison With FS-Net

Remember that FS-Net classifies TLS flows by using the packet length sequence of each flow as classification features. In this part, we evaluate the classification performance of FS-Net on *Dataset-I*. For *Dataset-I*, columns 5 to 7 of Table V report the cross-validation results of *Recall*, *Precision* and *F−measure* three metrics of FS-Net for each mobile application. Specifically, the cross-validation results of the *Recall* values vary in the range of 81.72% – 98.72% for different applications, the cross-validation results of the *Precision* values vary in the range of 81.92% – 99.27% for

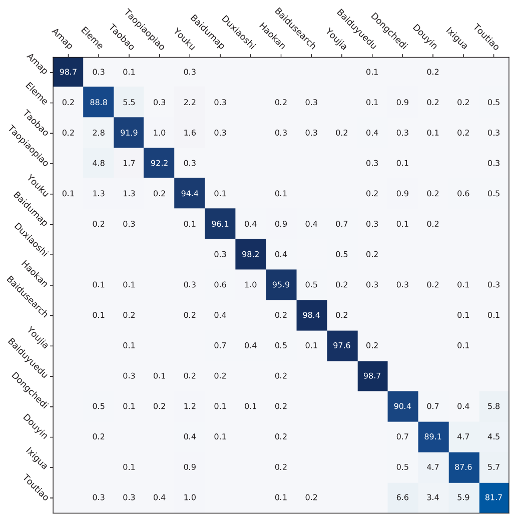


Fig. 11. Confusion matrix of FS-Net on Dataset-I.

different applications, and the cross-validation results of the $F-measure$ values vary in the range of 81.81% – 99.00% for different applications. We notice that FS-Net has an average $Recall$ of $93.31(\pm0.12)\%$, an average $Precision$ of $93.34(\pm0.10)\%$, and an average $F-measure$ of $93.31(\pm0.11)\%$. In addition, we report the classification confusion matrix of FS-Net on *Dataset-I* in Fig. 11. Compared with FS-Net, NeuTic works much better, where the average $F-measure$ increases by about 1.61%. In addition, we also notice NeuTic outperforms FS-Net on the $F-measure$ for all the 15 mobile applications. The main reasons why NeuTic performs better than FS-Net in classification accuracy are: (1) NeuTic considers some other fields of each packet to help classify encrypted TLS traffic, and 2) the deep learning model in NeuTic has stronger memory ability for sequence data.

VIII. Conclusion

This paper represents a novel attempt to construct a TLS traffic classification method for network traffic generated on cloud platforms using advanced deep learning techniques. Our approach is purely based on the packet sequence of each TLS flow, and thus it does not need to assemble IP packets into TLS messages. Our experimental results on real-world datasets show that NeuTic has the ability to achieve an excellent classification performance. In addition, NeuTic outperforms the state-of-the-art methods on encrypted TLS traffic classification with better classification effectiveness.

References

[1] C. Kreibich, "Design and implementation of netdude, a framework for packet trace manipulation," in *Proc. USENIX Annu. Tech. Conf*, 2004, pp. 1–10.

[2] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Comput. Netw.*, vol. 31, nos. 23–24, pp. 2435–2463, 1999. [Online]. Available: https://citeseer.ist.psu.edu/paxson98bro.html

[3] M. Roesch, "SNORT: Lightweight intrusion detection for networks," in *Proc. 13th Syst. Admin. Conf. (LISA)*, Nov. 1999, pp. 229–238.

[4] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, "A semantics-aware approach to the automated network protocol identification," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 583–595, Feb. 2016.

[5] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "KISS: Stochastic packet inspection classifier for UDP traffic," *IEEE/ACM Trans. Netw.*, vol. 18, no. 5, pp. 1505–1515, Oct. 2010.

[6] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and Y. Xiang, "Internet traffic classification by aggregating correlated naive Bayes predictions," *IEEE Trans. Inf. Forensics Security*, vol. 8, no. 1, pp. 5–15, Jan. 2013.

[7] A. Tongaonkar, R. Torres, M. Iliofotou, R. Keralapura, and A. Nucci, "Towards self adaptive network traffic classification," *Comput. Commun.*, vol. 56, pp. 35–46, Feb. 2015.

[8] J. Zhang, C. Chen, Y. Xiang, W. Zhou, and A. V. Vasilakos, "An effective network traffic classification method with unknown flow detection," *IEEE Trans. Netw. Service Manage.*, vol. 10, no. 2, pp. 133–147, Jun. 2013.

[9] Z. Zhang, Z. Zhang, P. P. C. Lee, Y. Liu, and G. Xie, "Toward unsupervised protocol feature word extraction," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 10, pp. 1894–1906, Oct. 2014.

[10] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 1257–1270, Aug. 2015.

[11] D. M. Divakaran, L. Su, Y. S. Liau, and V. L. L. Thing, "SLIC: Self-learning intelligent classifier for network traffic," *Comput. Netw.*, vol. 91, pp. 283–297, Nov. 2015.

[12] Y. Wang, X. Yun, Y. Zhang, L. Chen, and T. Zang, "Rethinking robust and accurate application protocol identification," *Comput. Netw.*, vol. 129, pp. 64–78, Dec. 2017.

[13] N. Hubballi and M. Swarnkar, "$BitCoding$: Network traffic classification through encoded bit level signatures," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2334–2346, Oct. 2018.

[14] N. Hubballi, M. Swarnkar, and M. Conti, "BitProb: Probabilistic bit signatures for accurate application identification," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 3, pp. 1730–1741, Sep. 2020.

[15] Y. Chen, T. Zang, Y. Zhang, Y. Zhou, and Y. Wang, "Rethinking encrypted traffic classification: A multi-attribute associated fingerprint approach," in *Proc. IEEE 27th Int. Conf. Netw. Protocols (ICNP)*, Oct. 2019, pp. 1–11.

[16] M. Korczynski and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2014, pp. 781–789.

[17] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order Markov chains and application attribute bigrams," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 8, pp. 1830–1843, Aug. 2017.

[18] C. Liu, Z. Cao, G. Xiong, G. Gou, S.-M. Yiu, and L. He, "MaMPF: Encrypted traffic classification based on multi-attribute Markov probability fingerprints," in *Proc. IEEE/ACM 26th Int. Symp. Quality Service (IWQoS)*, Jun. 2018, pp. 1–10.

[19] W. Zheng, C. Gou, L. Yan, and S. Mo, *Learning to Classify: A Flow-Based Relation Network for Encrypted Traffic Classification*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 13–22.

[20] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "FS-net: A flow sequence network for encrypted traffic classification," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 1171–1179.

[21] R. Schuster, V. Shmatikov, and E. Tromer, "Beauty and the burst: Remote identification of encrypted video streams," in *Proc. 26th USENIX Secur. Symp. (USENIX Secur.)*, 2017, pp. 1357–1374.

[22] S. Rezaei, B. Kroencke, and X. Liu, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2020.

[23] X. Xiao, W. Xiao, R. Li, X. Luo, H.-T. Zheng, and S.-T. Xia, "EBSNN: Extended byte segment neural network for network traffic classification," *IEEE Trans. Dependable Secure Comput.*, early access, Aug. 2, 2021, doi: 10.1109/TDSC.2021.3101311.

[24] A. Nascita, A. Montieri, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescape, "XAI meets mobile traffic classification: Understanding and improving multimodal deep learning architectures," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4225–4246, Dec. 2021.

[25] Y. Wang, X. Yun, Y. Zhang, C. Zhao, and X. Liu, "A multi-scale feature attention approach to network traffic classification and its model explanation," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 2, pp. 875–889, Jun. 2022.

[26] Z. Zhao, Y. Lai, Y. Wang, W. Jia, and H. He, "A few-shot learning based approach to IoT traffic classification," *IEEE Commun. Lett.*, vol. 26, no. 3, pp. 537–541, Mar. 2022.

[27] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30. Red Hook, NY, USA: Curran Associates, 2017, pp. 1–11.

[28] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statistics*, 2011, pp. 315–323.

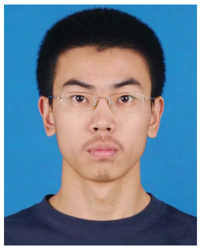
[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[30] J. Lei Ba, J. Ryan Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*.

[32] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should i trust you?': Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144.

Xiaochun Yun received the B.S. and Ph.D. degrees from the Harbin Institute of Technology, China, in 1993 and 1998, respectively. He is currently a Professor with the Institute of Information Engineering, Chinese Academy of Sciences (CAS), China. His research interests include network and information security. He is a member of Advisory Committee for Chinese State Informatization and a member of Subject Matter Expert Group of Information Security Technology for National High Technology Research and Development Program of China. He was honored with the First Prize of the Chinese National Award for Science and Technology Progress in 2002 and 2011.



Yipeng Wang (Member, IEEE) received the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences (CAS), in 2014. He is currently an Associate Professor with the Faculty of Information Technology, Beijing University of Technology. He has published more than 40 research papers in international journals and conferences, such as IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and IEEE INTERNATIONAL CONFERENCE ON NETWORK PROTOCOLS. His research interests are in networking, network security and machine learning, in particular network protocol inference. He has won the Best Paper Award at ICNP for his work on protocol format inference. He serves as a Regular Reviewer for IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *Computer Networks* (Elsevier), and *Neurocomputing* (Elsevier).

Yongzheng Zhang received the B.S. and Ph.D. degrees from the Harbin Institute of Technology, China, in 2001 and 2006, respectively. He is currently a Professor and a Ph.D. Supervisor at the Institute of Information Engineering, Chinese Academy of Sciences (CAS), China. His research interests include network security, particularly cyberspace security situational awareness. He was honored with the First Prize of the Chinese National Award for Science and Technology Progress in 2011.

Chen Zhao received the B.E. degree in information management and information system from Tianjin University and the Ph.D. degree in cyber security from the Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China. Her research interests include network traffic analysis and DNS security issues.

Zijian Zhao (Graduate Student Member, IEEE) received the B.S. degree in computer science from the Beijing University of Technology, Beijing, China, where he is currently pursuing the M.S. degree in computer technology with the Faculty of Information Technology. His research interests include network traffic classification and artificial intelligence.