EBSNN: Extended Byte Segment Neural Network for Network Traffic Classification

Xi Xiao, Wentao Xiao, Rui Li, Xiapu Luo, Haitao Zheng, and Shutao Xia

Abstract—Network traffic classification is important to intrusion detection and network management. Most of existing methods are based on machine learning techniques and rely on the features extracted manually from flows or packets. However, with the rapid growth of network applications, it is difficult for these approaches to handle new complex applications. In this article, we design a novel neural network, the Extended Byte Segment Neural Network (EBSNN), to classify netwrk traffic. EBSNN first divides a packet into header segments and payload segments, which are then fed into encoders composed of the recurrent neural networks with the attention mechanism. Based on the outputs, another encoder learns the high-level representation of the whole packet. In particular, side-channel features are learned from header segments to improve the performance. Finally, the label of the packet is obtained by the softmax function. Furthermore, EBSNN can classify network flows by examining the first few packets. Thorough experiments on the real-world datasets show that EBSNN achieves better performance than the state-of-the-art methods in both the application identification task and the website identification task.

Index Terms—Recurrent neural network, traffic classification, application identification, website identification

1 Introduction

Classifying and fingerprinting network traffic according to their applications or the visited websites [1] is an important task in network management and cyberspace security with many applications [2], [3], [4]. Traditional methods rely on port information or protocol specifications [5]. Since more and more applications do not have a fixed port number or well-documented protocol format, traditional methods cannot get satisfactory results.

Recently researchers employ machine learning methods to address this issue [6]. These methods first extract features of specific protocols at the packet level or the flow level and then use these features to construct a classifier for traffic classification. At the packet-level, each packet will be classified. At the flow-level, the granularity of the classification is flow. Various supervised and unsupervised methods have been used [6] to classify many network protocols. Nevertheless, these methods face two challenging issues. First, since there are enormous new applications, it is time-consuming and error-prone to manually seek new features that work for all applications. Second, the network traffic of modern applications is so complex and dynamic that traditional mining methods cannot generalize well to fingerprint specific applications accurately.

To cope with these challenges, in this paper, we leverage deep learning methods to automatically learn the representation of the network traffic. Deep learning methods have made great progress in many areas [7], including natural language processing, computer vision, etc. Compared with the traditional machine learning based methods, deep learning automatically learns more complex and expressive features with amazing generalization and robustness. Since the network packets can be seen as a kind of natural language between network applications, we propose a novel deep learning neural network, the Extended Byte Segment Neural Network (EBSNN), for traffic classification.

Our proposed network is based on Recurrent Neural Network (RNN) and intends to make full use of all information in network packets, including the side-channel data, which refer to the information that can be learned from the encrypted traffic without decryption [8], such as packet lengths, directions, and metadata. Specially, EBSNN learns side-channel features from packet header. Since the diverse section of a packet may contribute differently to traffic classification, we divide part of its header and all its payload into short byte segments. To find the discrimination information from byte segments, we adopt the attention mechanism. In detail, each segment is fed into the attention encoder composed of RNN units. Subsequently, another attention encoder with the same structure learns the high-level representation of the whole packet. EBSNN employs the softmax function to predict the label of the packet. In EBSNN, the Focal Loss [9] is used to overcome the class imbalance issue in the real data. Moreover, we extend the Byte Segment Neural Network (BSNN) method [10] to classify the whole flow. Specifically, we aggregated the results of the first k non-empty packets in the flow to obtain the label of the flow.

- *Xi Xiao is with the Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong 518057, China, and also with Peng Cheng Laboratory, Shenzhen 518000, China. E-mail: xiaox@sz.tsinghua.edu.cn.*
- *Wentao Xiao, Rui Li, Haitao Zheng, and Shutao Xia are with the Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong 518057, China. E-mail: {xwt20, lir16}@mails.tsinghua.edu.cn, {zheng.haitao, xiast}@sz.tsinghua.edu.cn.*
- *Xiapu Luo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China. E-mail: csxluo@comp.polyu.edu.hk.*

Manuscript received 2 Nov. 2020; revised 14 July 2021; accepted 26 July 2021. Date of publication 2 Aug. 2021; date of current version 2 Sept. 2022. (Corresponding author: Xiapu Luo.)
Digital Object Identifier no. 10.1109/TDSC.2021.3101311

By automatically learning the representations of the network traffic, EBSNN can be applied to many applications. In this paper, we use it for two tasks, namely identifying the applications that send the traffic and determining the target websites being visited according to the captured traffic. We prepare two datasets for the two tasks and conduct extensive experiments to evaluate new method. The experimental results show that our method is superior to the state-of-the-art methods [11], [12], including the machine learning and deep learning approaches for both two tasks. Furthermore, the experiments of application identification at the flow-level demonstrate the scalability of EBSNN that works at both flow-level and packet-level.

This paper is an extension of our previous conference paper [10]. The new contributions in this paper are summarized as follows.

- We design a novel deep learning network, the Extended Byte Segment Neural Network (EBSNN), for traffic classification. EBSNN introduces the aggregate strategy that relies on only the first k packets in the flow to identify the flow. The extension in the flow-level classification and the website identification task demonstrate the scalability of EBSNN.
- EBSNN leverages the side-channel features to improve the performance. The header is an input into EBSNN, from which suitable side-channel features are automatically learned and exploited.
- Compared with the dataset in [10] that consists of only 10 classes, we collect and publish two large real-world datasets from 29 applications and 20 popular websites covering most daily life areas in this work. Further, more base-line methods are implemented for the performance evaluation. Besides the traditional machine learning-based method (Securitas [11]), another deep learning-based approach, DeepPacket [12], is used as the baseline.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 describes the proposed method and Section 4 presents the experiment results. Section 3 describes the proposed method, and Section 4 presents the experiment results. We conclude the paper and discuss the limitations of this work in Section 5.

2 Related Work

Previous studies on network traffic classification have adopted different techniques. In this section, we mainly focus on the machine learning based approaches. Though more and more applications use SSL/TLS to protect their traffic, there is still abundant information that can be learnt from the encrypted traffic, such as packet lengths, directions, metadata, and TCP timestamp. We call such information *side-channel data*. Meanwhile, the payload content also plays an important role in capturing useful information from network traffic even if they are encrypted. According to the information used in traffic classification, these methods fall into two categories: 1) side-channel data-based methods and 2) payload content-based methods.

2.1 Side-Channel Data Based Methods

The side-channel data based methods mainly employ the information of packet headers and traffic flows. Typical side-channel data includes packet length and direction according to the *source* and *target IP addresses* in the traffic, etc. The sequence of the side-channel data from the whole network flow also contain useful information. Thus, existing studies usually first acquire side-channel data sequences and then transform them into features suitable for machine learning [13]. In particular, two categories of features can be extracted from the side-channel data for traffic classification.

2.1.1 Statistical Features

Many statistics can be obtained from the side-channel data sequence, including count, maximum, minimum, mean, standard deviation, variance, skew, median absolute deviation, kurtosis, percentiles, etc. These numbers are calculated from the entire flow or the subflows. A subflow is part of a flow with some constraints. For example, burst is a kind of subflow which consists of consecutive packets with same direction[14]. Roughan *et al.* used Nearest Neighbour, Linear Discriminate Analysis, and Quadratic Discriminant Analysis methods to classify the traffic based on the statistics [15]. Bernaille *et al.* observed that the size and the direction of the first few packets of the TCP connection are important. According to the observation, they proposed a model based on simple K-means [16]. There are many methods based on cluster techniques. Zhang *et al.* [17] utilized flow correlation information to identify the unknown application with flow label propagation and compound classification techniques including K-means clustering. Luis Sacramento *et al.* [18] found that malicious traffics are located at small clusters in real flows when performing clustering, and implemented a novel Network Intrusion Detection System (NIDS) named FlowHacker. Their approach extracts features from flows automatically and leverages two-stage clustering to iteratively detect attacks.

Besides, Taylor *et al.* implemented an application classification framework [13], which exploits statistical features of the packet length sequence, to fingerprint smartphone applications. They evaluated its robustness with different devices, application versions, and time. Recently, in the emerging areas of blockchains[19], [20] such as smart contract and decentralized applications, Shen *et al.* [21] identified encrypted traffic of Decentralized Apps (DApps) on Ethereum[22]. They fused three statistical features (i.e., packet lengths, bursts, and time series) of different dimensions by leveraging kernel functions such as polynomial and radial basis. As for the packet lengths, Kampeas el at. presented an efficient sampling strategy [23] to satisfy the demand of real-time classification in Software-Defined-Network (SDN) environments, which only uses packets of zero payload length such as ACK in TCP flows to restore the whole packet length sequence. They leveraged the first-k a-APDUs in the flow as the feature, where a-ADPU is the sum of the data byte lengths of the consecutive packets transmitted in the same direction.

2.1.2 Sequential Features

The information of the relationships among the packets in the same flow also plays a vital role in network traffic classification. Since the side-channel data can be extracted from the encrypted network flow and form time series, it is

naturally appropriate to use Markov model-based methods. Anderson *et al.* [24] performed binary classification of encrypted malware traffic with three kinds of features: the statistical features of lengths and times, the first-order Markov chain constructed from the sizes of the first 50 packets of a flow, and the feature of TLS handshake metadata. Subsequently, the Multi-attribute Markov Probability Fingerprints (MaMPF) was proposed in [25]. The authors found that the packet length is subject to power-law distribution. Therefore, they first adopted power-law division to transform the packet length sequence into more discriminating Length Block Sequence (LBS). Then, two Markov models are built for Message Type Sequences (MTSs) and LBSs. Finally, the output probabilities of these models are fed into the machine learning classifiers.

Since Recurrent Neural Network (RNN) is suitable for modeling the sequence data, Liu *et al.* designed a new deep learning model named Flow Sequence Network (FS-Net) [26] for the encrypted flow-level classification using the packet length sequence. FS-Net leverages the autoencoder architecture with the reconstruction mechanism to boost the feature learning, where both the encoder and the decoder are two layer Bi-GRU.

Remark 1: Most of existing methods build their models upon specific side-channel features extracted from the packet header. To make use of more coarse and wider side-channel features rather than manually crafted features used in existing studies, our model automatically learns the most suitable side-channel features from network traffic.

2.2 Methods Using Payload Content

The payload of a packet can be treated as a sequence of bytes or bits. To make use of the information provided by payloads, traditional approaches try to find signatures to match the network traffic. These signatures are usually in the form of n-grams or keywords, and the identification of network traffic exploits the appearance frequency of signatures in the payload. Besides these traditional signature based methods, techniques in Natural Language Processing (NLP) such as the topic model are also used. Meanwhile, deep learning is introduced to learn complex but useful representations of the packets as well.

2.2.1 Signature Based Methods

The signature-based methods identify the application protocol from network traffic using signatures extracted from payload. Deri *et al.* developed an open-source high-speed deep packet inspection tool named nDPI [27], which supports over 500 different protocols. By using the signatures extracted from both the packet header and the payload in byte-level, DPI can perform string detection in streaming data without degrading network throughput. It achieved high accuracy and efficiency, but the prior knowledge of each protocol is required. To overcome this limitation, Hubballi *et al.* proposed a bit-level DPI-based signature construction method called BitCoding [28]. To automatically match the bit sequences (i.e., the content of the packet payload) of all the samples with the same label, they used the first-k bits of the payload in the packet (or flow if the length of the first packet is less than k bits) to generate a compressed signature from counting. The signature is similar to the regular expression with only three operators, i.e., $.$, $*$, and $\{n\}$. They also used Relaxed Hamming Distance [28] to measure the signature overlap, which can be decreased by increasing k.

In addition, Haffner *et al.* leveraged three machine learning algorithms, i.e., Naive Bayes, AdaBoost, and Maximum Entropy, to automatically extract signatures from the first n-bytes in a TCP flow [29].

2.2.2 Topic Model Based Methods

Due to the similarity between the word-document relations in NLP and the byte-payload relations in network traffic, some studies use topic model techniques in NLP to find the powerful keywords for network traffic classification. The topic model aims to discover the abstract keywords for the documents. By the same way, some researchers devote to seeking well-generalized keywords to recognize the packets in the network traffic. Securitas [11] is the first topic model based method, which applies Latent Dirichlet Allocation (LDA) and the word bag model to extract the feature of protocols. The feature is the distribution of the keywords in the payloads with same protocol. Then, Support Vector Machine (SVM), C4.5 decision tree or Bayes Network are employed to do classification. In addition, Xiao *et al.* [30] used dynamic cluster topic models to obtain the most common keywords in the cluster for the traffic classification model.

2.2.3 Deep Learning Based Methods

Deep learning has been widely applied in many important fields, including traffic classification and web attack detection [31]. In the scenario of smart home Software-Defined-Network (SDN), Wang *et al.* proposed an SDN inspired Home Gateway Framework (SDN-HGW) [32] that provides fine-grained application-level traffic classification. They developed encrypted data classifiers named DataNets, which consist of three vanilla deep learning models, i.e., multilayer perceptron, Convolutional Neural Network (CNN), and stacked autoencoder. Meanwhile, Lotfollahi *et al.* presented DeepPacket [12] with similar deep learning models. The one-dimensional (1D) CNN was one of the proposed models that achieved the best performance in their experiments. It transforms the 1D byte-vectorized packets into two-dimensional tensors. Giuseppe Aceto *et al.* leveraged multimodal deep learning [33] and fusion [34] techniques to improve the performance. These models either miss the characteristics of network traffic or involve many high overhead modules which introduce unnecessary computational overhead. Therefore, we design a novel neural network which is suitable to the characteristics of network traffic and outperforms DeepPacket [12] as shown by the experimental results.

Remark 2: Compared with side-channel data-based methods, payload content-based methods involve more traffic information, i.e., the payload content of packets. Because of the complexity of network traffic behavior, it is difficult to extract features for traditional machine learning methods. We design a novel deep learning based traffic classification method without the need of feature extraction.

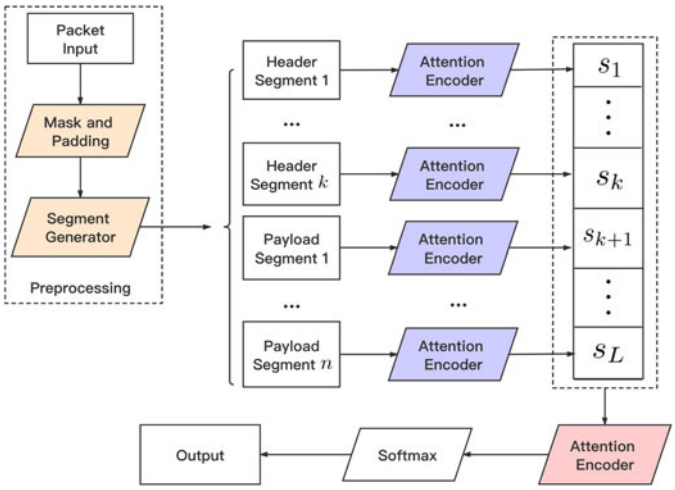


Fig. 1. Overall Structure of EBSNN.

2.3 Deep Learning in Natural Language Classification

Since network traffic can be regarded as language conversations between network applications, the techniques in NLP have been applied to traffic classification, especially deep learning networks in natural language classification. Among them, Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) are two widely used deep learning networks. RNN [35] maintains internal memory of the historical information and is efficient for capturing contextual features from sequential data. Long Short-Term Memory (LSTM) [36] is a special and powerful kind of RNN. It can remove or add memory by introducing forget, input, and output gates. Based on LSTM, Gated Recurrent Unit (GRU) [37] combines the forget and input gates into a single "update gate" and makes some other changes.

To capture the coherence between sentences in a document, RNN was employed in Hierarchical Recurrent Neural Network Language Modeling (HRNNLM) [38] for document modeling. With a two-step training process, HRNNLM is convergent at both sentence-level and word-level. Lai *et al.* introduced a Recurrent Convolutional Neural Network (RCNN) [39] for text classification. The recurrent structure catches contextual information with less noise than traditional neural networks. RCNN can automatically decide which words are more important in text classification.

In addition, the hierarchical network was created to mirror the document structure in Hierarchical Attention Networks (HANs) [40]. Specially, HAN employs two attention layers to make sure that every sentence and word has different importance in document representation.

Remark 3: These studies achieve good performance in natural language classification. Motivated by this, we design a deep learning network with the attention mechanism for network traffic classification.

3 EBSNN

This section describes the architecture of the Extended Byte Segment Neural Network (EBSNN) in detail. The initial objective of EBSNN is to classify the single packet into the corresponding application or website. The overall structure of EBSNN at packet-level is illustrated in Fig. 1. In our model, every single packet is preprocessed and fed into the segment generator and broken into a series of segments including the header segment and the payload segment. Then, the attention encoder transforms each segment to a segment vector. After that, all these vectors of the packet are combined and put into another attention encoder to get a representation vector of the whole packet (In Fig. 1, the attention encoders in different color have different RNN layers). Finally, the representation vector is introduced to a softmax function to calculate the predicted label of the packet.

Details of the proposed method are described in the following subsections: Section 3.1 introduces the prepeocessing details, especially the segment generator; Section 3.2 gives the formal definition of the proposed model; and Section 3.3 describes the extension of our proposed model in flow-level classification.

TABLE 1
Structure of the Headers and the Payload in the Raw Packet

Section	Start position (bit) $\sim$ End position (bit)
Ethernet Header	$1 \sim 112$
IPv4 Header	$113 \sim 112 + 32 \times U$
TCP/UDP Header	$113 + 32 \times U \sim 112 + 32 \times (U + V)$
Payload	$113 + 32 \times (U + V) \sim W$

3.1 Preprocessing

To transform the raw data into the proper representation that is suitable to the model, we first preprocess the raw packet. The packet consists of the header and the payload. The former is composed of the Ethernet II header, the IPv4 header, and the TCP/UDP header. Different from BSNN [10] that only uses payloads, EBSNN also learns side-channel features from the packet header. We first read the packet in binary format and transform it into a sequence of 8-bit integers in the range of $[0, 255]$. From the beginning to the end of the raw packet, there are the Ethernet II header, the IPv4 header, the TCP/UDP header and the payload successively. The length of the IPv4 header and that of the TCP/UDP header can be calculated by the Internet Header Length field in the IPv4 header and the Data Offset field in the TCP header (the Length field in the UDP header).

Table 1 lists the starting and ending positions of these sections in a raw packet, where W is the length of the raw packet; U is the value of the Internet Header Length field, i.e., $117 \sim 120$ bits in the raw packet, and V can be found according to U. Specifically, for the TCP header, V is the value of the Data Offset field, i.e., $177 + 32 \times U \sim 180 + 32 \times U$ bits in the raw packet; while for the UDP header, V is the value of the Length field, i.e., $145 + 32 \times U \sim 160 + 32 \times U$ bits in the raw packet.

An example of a raw packet that contains the TCP header is shown in Fig. 2. One hexadecimal number presents 4 bits in binary. The red numbers indicate the range of each section and each field in bits. In this case, we first get $U = 5$ from $117 \sim 120$ bits in the raw packet. Then $V = 8$ is obtained from $177 + 32 \times 5 \sim 180 + 32 \times 5$ bits. Finally,

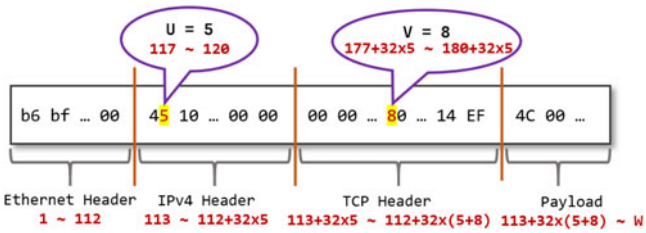


Fig. 2. An example of the structure of the headers and the payload of the raw packet.

based on $U = 5$ and $V = 8$, we calculate the positions of the IPv4 header, the TCP header and the payload according to Table 1.

The 8-bit integer sequence of the packet is then split into the Ethernet header, IPv4 header, the TCP/UDP header, and the payload subsequences. Then, the segment generator preprocesses and breaks these subsequences into byte segments with fixed-length N, which is called segment length. The segment generator provides a simple representation of the packet. Each byte in the segment can be regarded as a character. Therefore, a segment is analogous to a sentence in natural language, and then a packet can be seen as a document.

There are two kinds of byte segments, i.e., header segments generated from the header sequence and payload segments from the payload sequence. Header segments indicate IPv4 header segments and TCP/UDP segments. To get payload segments, in most cases the total number of bytes of a payload is not a multiple of the segment length N, and zeros are padded into the end of the payload. N bytes construct one segment, and thus the number of segments is $\lceil M/N \rceil$, where M is the length of the payload sequence. As for the header segments, there are some additional operations. Although packet header information can be exploited for traffic classification, not all the bits of the header are useful and part of them (e.g., *source* and *destination IP address* in the IPv4 header) might lead to overfitting. Therefore, we first mask some fields in the header with zeros and then generate header segments from the header.

1) Ethernet II header: As this header only contains *EtherType*, *source* and *destination MAC addresses*, we simply abandon the whole Ethernet II header.
2) IPv4 header: Some fields in the header do not provide useful information and are irrelevant to our model, such as *checksum* and *identification*. Therefore, we discard these information and mask these fields with zeros, i.e., replace all the bits in such fields with zeros for all packets: *IP identification* ($32 \sim 37$ bits in the IPv4 header), *IP checksum* ($80 \sim 95$ bits), *source IP address* ($96 \sim 127$ bits), and *destination IP address* ($128 \sim 159$ bits). After that, the segment generator, including the padding, is the same as the segment generator of the payload segment.
3) TCP/UDP header: Similar to the IPv4 header, we replace the *source port* ($0 \sim 15$ bits) and the *destination port* ($16 \sim 31$ bits) in the TCP/UDP header with zeros.

After finishing the aforementioned steps, we get IPv4 header segments, TCP/UDP header segments, and payload segments. Each segment is an integer sequence with the length of N before being input into one attention encoder. The objectives of the attention encoder are to seek the best representation of the packet and understand the packet. Different from the hand crafted side-channel features [41], our EBSNN model automatically learns more suitable side-channel features than hand-crafted methods by using the stochastic gradient descent optimizer such as Adam [42].

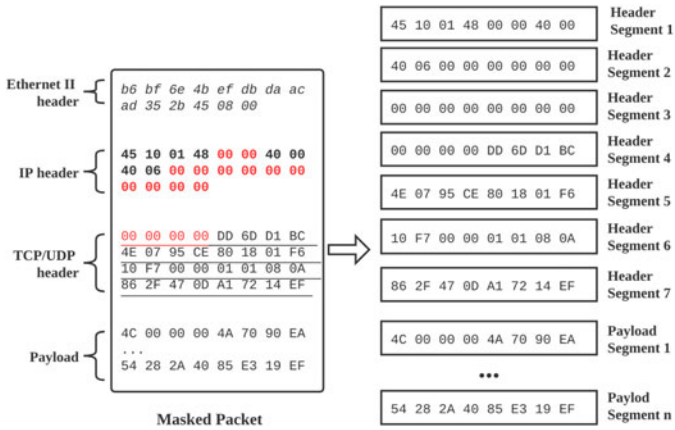


Fig. 3. An example of segment generation for the masked packet.

Fig. 3 shows an example of segment generation with the TCP header. In the figure, the left part is a masked packet in hexadecimal, where two numbers represent one byte. The italic part in the figure is the Ethernet II header which is removed when preprocessing. The bold part in the packet indicates the IPv4 header, and the underlined part indicates the TCP header. Zeros colored in red are masked contents in the packet. In this example, the packet is broken into short segments (see the right part in the figure) with $N = 8$, i.e., one segment involves 8 bytes. In the figure, there are 7 header segments and n payload segments.

3.2 Model

EBSNN achieves a reasonable representation of an input sequence by employing hierarchical RNN units to combine the contextual information of every byte and every segment with two stage design. Moreover, it uses an attention layer to find out which part is important to the whole packet.

The collection of the whole packet can be indicated as $A = \{\{a_{i,n}\}_{n=1}^{N}\}_{i=1}^{L}$, where $a_{i,n}$ denotes the nth byte in segment i, including header segments and payload segments, and L is the number of segments. For applying RNN units to packets, we need to embed bytes to vectors. As described above, the byte $a_{i,n}$ is an integer between 0 to 255, which can be transformed into an one-hot vector in 256-dimension. Then, the one-hot vector is embedded into a dense vector $x_{i,n} \in \mathbb{R}^E$.

To get information from the segments, we employ an RNN structure with the attention mechanism inspired by [43]. This structure is hereafter refered as the *attention encoder*.

At first, we get

$$[h_{i,1}, \ldots, h_{i,N}] = \mathsf{RNN}^{(1)}([x_{i,1}, \ldots, x_{i,N}]), \tag{1}$$

where $\mathsf{RNN}^{(1)}(\cdot)$ denotes the RNN layer (e.g., LSTM, GRU) with hidden size r. The attention output with respect to $x_{i,n}$

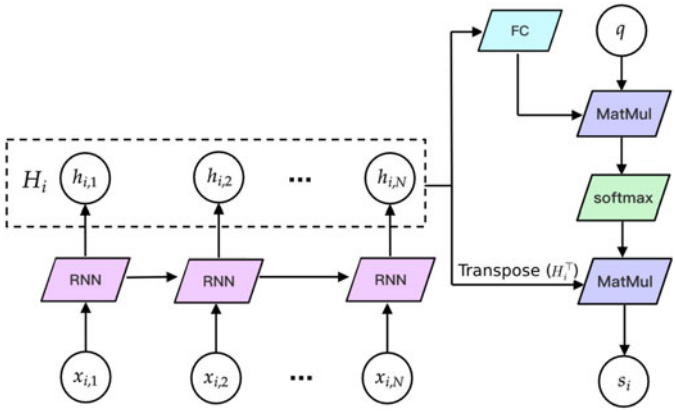


Fig. 4. Structure of the attention encoder.

is $h_{i,n} \in \mathbb{R}^f$, where f is r multiplied by the number of directions of the RNN. The RNN structure in the attention encoder can be either unidirectional or bidirectional.

Next, the outputs of all the bytes in one segment, $h_{i,1}, \ldots, h_{i,N}$, are combined into a segment matrix, $H_i = [h_{i1}, \ldots, h_{iN}]^\top \in \mathbb{R}^{N \times f}$, to represent the whole segment. Specifically, since diverse bytes may be of different importance in traffic classification, the attention mechanism [40] is applied to integrate the outputs.

Formally, the attention output $s_i \in \mathbb{R}^f$ of H_i is given by:

$$\begin{aligned} s_i &= \text{Attention}(q, H_i) \\ &:= H_i^\top \cdot \text{softmax}(\text{FC}(H_i) \cdot q), \end{aligned} \tag{2}$$

where $\text{FC}(\cdot)$ denotes the fully connected layer: $\text{FC}(H_i) := \tanh(H_i W_g + b_g) \in \mathbb{R}^{N \times f}$. In the above formula, $q \in \mathbb{R}^{f \times 1}$ is an abstract representation of the most informative byte in segment i [44], which is randomly initialized. Softmax normalizes the vector into a probability distribution which is defined as:

$$\text{softmax}(z)_i := \frac{\exp z_i}{\sum_{j=1}^{n} \exp z_j}, \tag{3}$$

where $z \in \mathbb{R}^n$ is the unnormalized input vector and the output is also an n-dim vector.

The transformation from $[x_{i,1}, \ldots, x_{i,N}]$ to s_i is called the *attention encoder*: $s_i = \text{AttEncoder}([x_{i,1}, \ldots, x_{i,N}])$. Fig. 4 shows the overall structure of the attention encoder where the input is the segment $[x_{i,1}, \ldots, x_{i,N}]$ and the output is s_i. MatMul denotes matrix multiplication in the figure. Using the same attention encoder with the same RNN layer $\text{RNN}^{(1)}(\cdot)$, each segment is transformed to the new representation. This is the first stage of EBSNN, where the output s_i is the contextual information of all the bytes in one segment. We denote the collection of all segments' representations of the packet as $S = [s_1, \ldots, s_L] \in \mathbb{R}^{f \times L}$. Note that S is a series of vectors just like a single embedded segment $[x_{i,1}, \ldots, x_{i,N}]$. Thus, we adopt another attention encoder with a different RNN layer $\text{RNN}^{(2)}(\cdot)$ to get the attention output of the whole packet from S. At last, a representation vector $d = \text{AttEncoder}([s_1, \ldots, s_L]) \in \mathbb{R}^f$ of the whole packet is obtained. This is the second stage of EBSNN, where the output d is the contextual information of all the segments.

After getting the representation d of a packet, we focus on the classification problem. For C classification, d can be transformed into a C-dimension vector $\tilde{y} = \text{softmax}(Wd + b)$ where $\tilde{y} \in \mathbb{R}^C$. The presentation d is the automated fingerprinting [45], [46] of the packet. Based on the automated fingerprinting d, the last softmax layer of EBSNN completes the identification. The softmax function is utilized to calculate the probability distribution, $\tilde{y}$, with respect to the labels of the packets. The predicted label is the label corresponded to the maximum element in $\tilde{y}$.

Thus, the whole process of our model can be formulated as:

$$\tilde{y} = \text{EBSNN}(A; \theta), \tag{4}$$

where θ is the model parameters of EBSNN.

Algorithm 1. Workflow of the Training Phase

Input: Batch size B, Segment length N, embedding size E, focal loss parameters γ, α, model parameters θ, learning rate λ, and training dataset $\mathcal{D}$

Output: Learned model parameters θ_k

1: Initial model parameters: $\theta \leftarrow \theta_0$
2: Initial step counter: $k \leftarrow 0$
3: **while** θ_k not connverged **do**
4: $k \leftarrow k + 1$
5: Getting data: randomly select trainning batch $\mathcal{B}$ of size B and its ground truth labels y from dataset $\mathcal{D}$
6: Preprocessing: $\mathcal{X} \leftarrow$ mask, zero-pad and segment $\mathcal{B}$ with segment length N
7: Forward propagation: $\tilde{y} \leftarrow \text{EBSNN}(\mathcal{X}; \theta_k)$
8: Backward propagation:
$\theta_{k+1} \leftarrow \theta_k - \lambda \cdot \text{Adam}\ (\nabla_\theta \text{FL}(y,\ \tilde{y};\ \alpha,\ \gamma))$
9: **end while**

Considering the training loss, we adopt the focal loss in EBSNN, because it shows strong advantages on the imbalanced multi-classification problem [9]. The key idea of the focal loss is that the instances which are hard to train are significant to the whole neural network. The focal loss for C-class classification can be simply calculated as follows:

$$\text{FL}(y, \tilde{y}; \alpha, \beta) := -\alpha(1 - \tilde{y}_l)^\gamma \log(\tilde{y}_l), \tag{5}$$

where $y \in \mathbb{R}^C$ is a one-hot vector of the ground truth of the packet, α the class weight, and γ the focusing parameter. l is the label of the ground truth such that $y_l = 1$ and other y_i $(i \neq l)$ are all zeros. $\tilde{y} \in \mathbb{R}^C$ is the probability distribution of the label predicted by our proposed model.

To summarize, the workflow of the training phase of EBSNN is shown in Algorithm 1 where the preprocessing is described in Section 3.1. The mini batch input and the parameters of EBSNN are $\mathcal{X}$ and θ. As for the optimization algorithm, we adopt the Adam optimizer [42] with batch size B.

3.3 Extension to Flow-Level Classification

The initial objective of BSNN is to identify the *single* packet and perform classification at packet-level. In EBSNN, we extend BSNN to make it suitable for the flow-level classification by introducing the aggregate strategy $\text{agg}(\cdot) : \mathbb{R}^{k \times C} \rightarrow \mathbb{R}^C$. Some studies [16], [47] indicate that the first k packets of

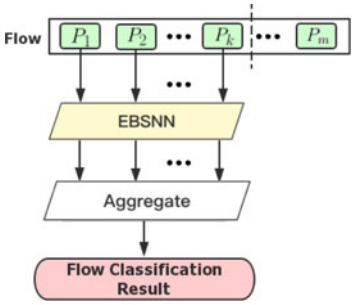


Fig. 5. Workflow of the extension to flow-level classification.

the flow involve the important information about the flow. Thus, the first k packets with the non-empty application-level payloads in the flow can represent the flow. Compared with BSNN, EBSNN provides support at the flow level and can be used to classify the entire flow.

At packet-level, as mentioned in the previous subsection, EBSNN learns the most suitable features as the automated fingerprinting [45], [46] for each packet. Based on the automated fingerprinting d, the last softmax layer of EBSNN completes the identification. The softmax function is utilized to calculate the probability distribution, $\hat{y}$, with respect to the labels of the packets. The overall workflow and architecture of the extension to the flow-level classification are showed in Fig. 5. A flow consists of a sequence of packets including some zero-length packets (e.g., ACK and SYN). Because these zero-length packets do not contain payloads, we discard them from the flow. Let the flow from the dataset be $x = [P_1, P_2, \ldots, P_k, \ldots, P_m]$, where P_i is the ith packet which contains non-empty application-level payload in the flow. Since the first few packets in the flow play a critical role in network flow classification [16], [47], we apply EBSNN on only the first k packets with non-empty application-level payloads in the flow. After that, the k softmax outputs of these packets can be obtained. Finally, we aggregate these outputs to get the label of the flow. The predicted label is the label corresponding to the largest element in the aggregated result. Specifically, these aggregate strategies include the sum of the softmax output and the votes for the packet results, etc.

It is worth noting that the flow-level classification methods such as [16], [47] are mainly based on specific side-channel features such as lengths and directions, which are usually extracted from the header (e.g., TCP header). In addition to the commonly used length and direction features, the packet header includes other features such as the options field in the TCP header and TLS handshake metadata. Compared with these methods, the headers of the first k packets in x are fed into EBSNN so that more side-channel features are involved (e.g., features from the options field) and our method automatically learns the most suitable side-channel features. Thus, EBSNN leverages more comprehensive information from both the side-channel features in headers and the content data in payloads.

4 Expremental Evaluation

To evaluate the performance of EBSNN, we conduct experiments on two large new real-world datasets in this section. At first, the datasets and the metrics are explained. Then, we analyze the impact of the parameters of our model and the structure of the network. Furthermore, the thorough experiments of the application identification task at the packet level are done and the comparison with the other methods are presented in detail. We also conduct the website identification task to evaluate the robustness of EBSNN on the task with more noisy data. Moreover, we perform extensive experiments for application identification at the flow level. The testing time of EBSNN is evaluated and compared with those of some other methods. All the experiments are conducted in a server with 128 GB memory, Intel Xeon Silver 4210 CPU, and NVIDIA 2080Ti GPU.

TABLE 2
Details of **D1** for the Application Identification Task

ID	Application	#Packets	ID	Application	#Packets
1	Vimeo	35267	16	iTunes	18066
2	Spotify	10034	17	Twitter	9314
3	VoipBuster	360403	18	JD	19487
4	Sina UC	155951	19	SOHU	39014
5	Netease Cloud Music	84873	20	Youtube	40856
6	Sina Weibo	49796	21	Youku	99909
7	Baidu	36364	22	Netflix	55189
8	Tudou	89720	23	AIM Chat	17260
9	Amazon	14530	24	Kugou	170167
10	Thunder	64325	25	Skype	632501
11	Gmail	11004	26	Facebook	245434
12	PPLive	181113	27	Google	6767
13	QQ	14045	28	MS-SQL	20882
14	Taobao	31136	29	MS-Exchange	1720
15	Yahoo Mail	38948	-	Average	88071

4.1 Dataset and Metrics

In the experiments, two datasets,[1] **D1** and **D2**, are used in two different tasks, the application identification and the website identification. **D1** is a large real-world dataset for application identification, which is based on 29 kinds of popular applications in many areas such as email, music, video, social network, search and shopping, etc. The datagrams of a specific protocol were caught by tracing the corresponding application processes. Specifically, the traffic collector first obtained the socket information of the target application. Then, datagrams were filtered and saved according to the socket information. Through this way, network traffic can be collected without noises. Note that our dataset consists of both encrypted and unencrypted network traffic data, which reflects the real world network scenario. The detailed information about the number of packets in each application can be found in Table 2. We can see that the packet numbers of various classes in the dataset are quite different, i.e., class imbalance. Such class

1. The data are available at https://drive.google.com/drive/folders/1-I3a3lM6v_ANU6uu_AUmpNYt7rGu3kzt, `dataset_29_D1.tar.gz` and `dataset_20_D2.tar.gz` are dataset **D1** and **D2**, respectively.

TABLE 3
Details of **D2** for the Website Identification Task

ID	Website	#Packets	ID	Website	#Packets
1	JD	150,157	11	Instagram	123,930
2	Netease Cloud Music	152,988	12	iQIYI	193,280
3	TED	262,622	13	QQ Mail	97,661
4	Amazon	145,377	14	Reddit	230,603
5	Baidu	137,859	15	Taobao	227,034
6	Bing	161,689	16	Tieba	125,235
7	Douban	138,904	17	Twitter	229,780
8	Facebook	249,179	18	Sina Weibo	323,014
9	Google	69,251	19	Youku	137,877
10	IMDb	253,404	20	Youtube	315,101

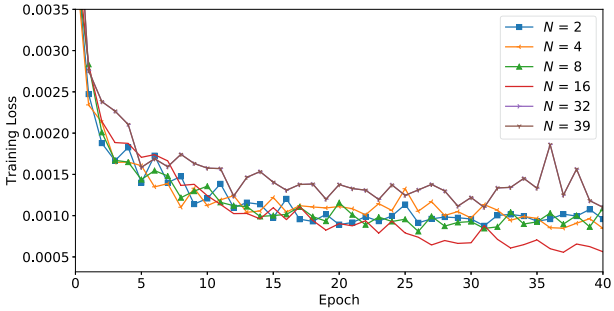
imbalance situation often appears in real-world network scenarios [48].

D2 is another large real-world dataset for the website identification, containing 20 popular websites. We collected D2 in 2020 by running the custom lightweight browser[2] and the network traffic dump tool (i.e., WireShark) in the isolated environment. The isolated environment was constructed with Docker and the network namespace, where only the browser and WireShark run. To collect as much pure network traffic as possible, the custom browser disabled all the noisy network traffic (e.g., search recommendations, browser update check, and account sync). For some websites that require login accounts, such as QQ Mail and Twitter, we manually logged in our personal accounts to obtain real-world network traffic. Compared with application identification, website identification [2], [49] is a task with more noises because many websites share the same static files stored in the same CDN and exhibit similar behaviors. Specifically, **D2** consists of 20 websites: JD, Netease Cloud Music, TED, Amazon, Baidu, Bing, Douban, Facebook, Google, IMDb, Instagram, iQIYI, QQ Mail, Reddit, Taobao, Tieba, Twitter, Sina Weibo, Youku, Youtube, covering almost every aspect of the daily life. Most of their traffic uses https and they cannot be distinguished by destination IP and port. The statistical information is shown in Table 3. To evaluate the classification model, $Precision$, $Recall$ and F_1 $score$ are defined for every target application.

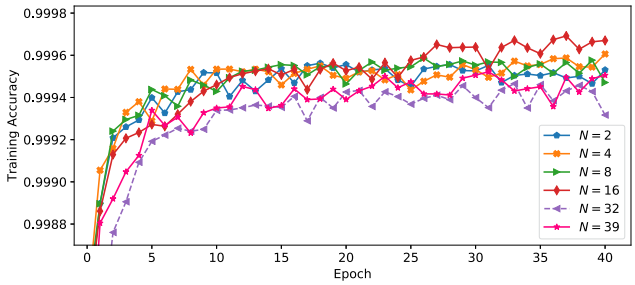
4.2 Parameter Selection

For training EBSNN, some hyperparameters should be set. In all the experiments, the dimensions of GRU and LSTM are 100. Thus, the output size of the attention encoder is 100. Besides, we use a learning rate of 0.001 and a dropout rate of 0.5. The other important hyperparameters in EBSNN can be chosen by conducting experiments, such as the segment length N, the batch size B, the embedding dimension E, and the hyperparameters α and γ of the focal loss. In this subsection, we investigate how to find these optimal hyperparamters.

2. Fork and build based on min browser with some modifications.



(a) Training Loss



(b) Training $Accuracy$

Fig. 6. Training performance of different segment length N.

4.2.1 Segment Length N

The segment length, N can influence the speed and effectiveness of the neural network. Thus, we conduct experiments to seek the proper N. We create different training and test sets with segment length $N = 2, 4, 8, 16, 32$ and 39, and train the model by 40 epochs. Meanwhile, the other parameters are set as $B = 128, \gamma = 2, E = 257$. The RNN unit we used in this part is GRU. EBSNN obtains nearly 100 percent test $Accuracy$ on all the datasets.

The training loss and the training $Accuracy$ with different segment lengths are shown in Figs. 6a and 6b. The abscissas of the two figures are the training steps. The y-axes of Figs. 6a and 6b denote the training loss and $Accuracy$, respectively. The test results are presented in Table 4a.

There are similar stable trends of different segment lengths in the figures of both loss and $Accuracy$. In detail, one too short or too long segment leads to slightly worse performance. It can be seen from Fig. 6a that the loss curve with a smaller N is more unstable than that with a bigger N. The unstable loss curve indicates the danger of traps in local optimum. In general, the segment length of 8 and 16 gets the most stable results. Note that 8 bytes (64 bits) and 16 bytes (128 bits) are commonly used block sizes in block ciphers such as Advanced Encryption Standard (AES) [50]. To achieve better performance, the segment length can be set the same as the block size. Moreover, considering that short segments result in more segments which bring more cost on the attention layer and long segments make the encoding time longer, we choose $N = 16$ in the following experiments.

4.2.2 Batch Size B

Our model is trained on batches during each epoch, where each batch is a subset of the whole dataset. Small batch sizes lead to more frequent update and thus converge quickly.

TABLE 4
Test Results With Different Hyperparameters

Segment Length N	Test *Accuracy*
2	$0.999386 \pm 2.94 \times 10^{-6}$
4	$0.999341 \pm 6.85 \times 10^{-6}$
8	$0.999400 \pm 6.85 \times 10^{-6}$
16	$0.999391 \pm 1.96 \times 10^{-5}$
32	$0.999269 \pm 1.08 \times 10^{-5}$
39	$0.999269 \pm 1.08 \times 10^{-5}$
(a) Test Results of Different N	
Batch Size B	Test *Accuracy*
16	$0.997970 \pm 1.84 \times 10^{-4}$
32	$0.998604 \pm 8.22 \times 10^{-5}$
64	$0.999223 \pm 1.37 \times 10^{-5}$
128	$0.999330 \pm 1.15 \times 10^{-5}$
256	$0.999207 \pm 1.12 \times 10^{-5}$
(b) Test Results of Different B	
Embedding Dimension E	Test *Accuracy*
16	$0.999413 \pm 1.96 \times 10^{-6}$
32	$0.999442 \pm 9.79 \times 10^{-6}$
64	$0.999503 \pm 2.54 \times 10^{-5}$
128	$0.999385 \pm 1.96 \times 10^{-6}$
257	$0.999319 \pm 1.17 \times 10^{-5}$
(c) Test Results of Different E	
γ of Focal Loss	Test *Accuracy*
1	$0.999400 \pm 6.85 \times 10^{-6}$
2	$0.999327 \pm 3.92 \times 10^{-6}$
5	$0.999342 \pm 1.17 \times 10^{-5}$
(d) Test Results of Different γ	

On the contrary, large batch sizes make the learning process converge slowly while bringing more accurate estimation of the training error gradient [51]. Meanwhile, a larger batch requires more GPU memory to store feature tensors and gradient information. Moreover, large batch sizes degrade the generalization performance persistently [51].

We conduct experiments of different batch size B ranging from 16 to 256, while fixing other hyperparameters. Specifically, to investigate the optimal batch size, other hyperparameters are set as $\gamma = 2, N = 8$, and $E = 257$. Fig. 7 presents the training *Accuracy* with respect to different batch sizes. Results in Fig. 7 show that our method with $B = 128$ achieves the best training *Accuracy* in almost all the time. In general, larger batch size gets better training *Accuracy*. However, too large batch size such as 256 may result in degraded generalization, as shown in Table 4b. Besides, Fig. 7 indicates that the result of $B = 128$ is more robust than $B = 256$. Therefore, the optimal batch size is selected as 128.

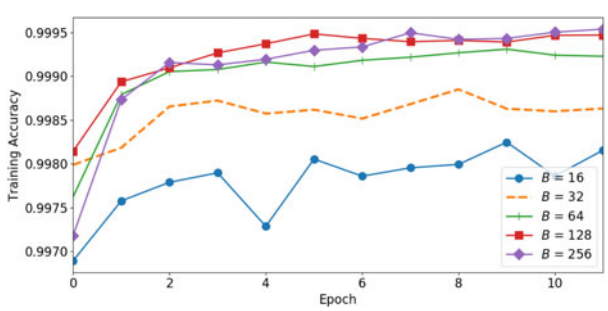


Fig. 7. Training *Accuracy* of different batch size B.

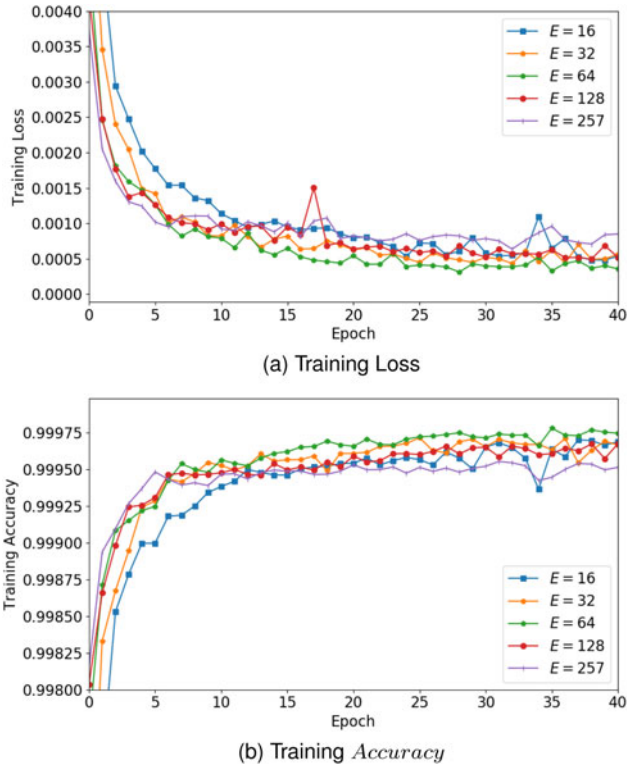


Fig. 8. Training performance of different embedding dimension E.

4.2.3 Embedding Dimension E

EBSNN treats each byte whose value ranges from 0 to 255 as the atomic unit. To make all the samples in the same batch have the same length, we pad flag 256 into the short samples and construct one-hot vectors of 257 dimensions. However, such one-hot vectors are extremely sparse and lack of the semantic relation between bytes. Therefore, it is useful in many cases to transform the one-hot vectors into dense vector representations. This transformation is called embedding. The dimension of the dense vector representation is called the embedding dimension. Different embedding dimensions have distinct influence on the performance. To study the impact, we conduct experiments with respect to the embedding dimension E. Various values of E are selected from $\{16, 32, 64, 128, 257\}$. Meanwhile, we set B, N, γ as 128, 16 and 2.

Fig. 8 presents the training results influenced by the embedding dimension. The test results are also reported in Table 4 c. It can be seen that relatively small ($E = 16$) and large values ($E = 128, 257$) of E lead to slightly worse training performance. The test *Accuracy* represented in Table 4 c also shows the same results. Combining the training results plotted in Fig. 8 and the test results listed in Table 4 c, $E =$

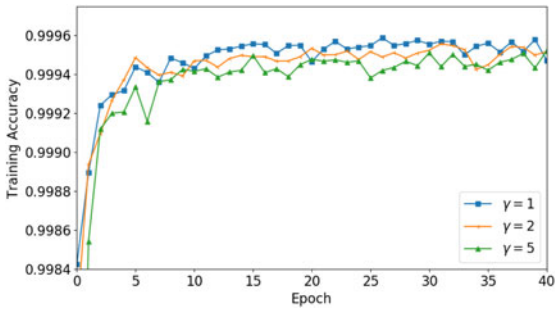


Fig. 9. Training performance of different γ.

64 is the optimal value that achieves the best results in both training and test.

4.2.4 Hyperparameters of Focal Loss

There are two hyperparameters of the focal loss, α and γ. α refers to the class weight. It is more difficult to correctly classify the classes with fewer samples. Thus, they should obtain more weights. Therefore, α can be calculated by one minus the percentage of the class.

In the following part, we study the impact of γ. γ is the focusing parameter and bigger γ lets the neural network focus more on the instances that are hard to train and easy to be misclassified. Generally, γ should be a positive number. When $\gamma = 0$ and $\alpha = 1$, the focal loss becomes the ordinary cross-entropy.

We perform experiments for the selection of γ on the imbalance dataset that contains 29 applications. γ is set as $1, 2, 5$ and the model is trained by 40 epochs. The other hyperparameters are fixed as $B = 128, N = 8, E = 257$.

Note that models with diverse γ calculate the training loss in different ways. Thus, we focus on the training *Accuracy* and the test *Accuracy*. Fig. 9 presents training *Accuracy* with respect to different γ. Note that higher training *Accuracy* may result in more overfitting. Table 4d shows test *Accuracy* of different γ. Although it can be seen from Fig. 9 that diverse γ achieves similar results, Table 4d demonstrates the test *Accuracy* of $\gamma = 1$ exceeds that of the others by a large margin. In addition, the training *Accuracy* of $\gamma = 1$ slightly outperforms that of the others in most epochs. These observations indicate that the optimal value of γ for EBSNN is 1.

According to the aforementioned experiments, we conclude the hyperparameters as follows. In the preprocessing phase, the segment length N is set as 16. In the training phase, the size of a batch is set as 128, i.e., 128 packets are input into our model in one training step. Meanwhile, the embedding dimension is chosen as 64. As for the focal loss, α and γ are one minus the percentage of the class and 1, respectively. Besides, we find out from Figs. 6 and 9 that training with 40 epochs is enough for EBSNN to achieve globally optimal solutions.

4.3 Impact of the Network Structure

The classification results are also affected by the structure of the neural network, such as the RNN unit, the direction of RNN and the input data. Therefore, in this subsection, we analyze the impact of the structure in detail.

TABLE 5
Impact of the Network Structure

Methods	Test *Accuracy*
EBSNN-LSTM	0.9996
EBSNN-GRU	0.9990
EBSNN-BiLSTM	0.9995
EBSNN-BiGRU	0.9989
BSNN-BiLSTM	0.9470
BSNN-BiGRU	0.9405

- *RNN unit*: The encoder of EBSNN involves RNN units. Due to the gradient vanishing problem in ordinary RNNs, we leverage two most popular RNN variant units, namely LSTM and GRU. These variant methods are named as EBSNN-LSTM or EBSNN-GRU according to whether LSTM or GRU is used as the RNN unit in EBSNN.
- *Directions of RNN*: There are two kinds of RNNs, the unidirectional RNN and the bidirectional RNN (BiRNN). The latter learns the representation from two opposite directions, while the former from only one direction. The variants of EBSNN, EBSNN-BiLSTM and EBSNN-BiGRU, adopt bidirectional RNNs, and the others use unidirectional RNNs.
- *Side-channel data*: As aforementioned, many flow-level classification methods adopt side-channel data obtained from the packet header, such as the packet length and the direction. In our previous work, BSNN [10], merely the information from the packet payload is utilized. In contrast, EBSNN leverages the information from not only the payload but also the header (with some masked fields). If the header is ignored, i.e., the side-channel features are not used, EBSNN downgrades into BSNN.

In fact, the difference between EBSNN and BSNN is whether the side-channel features are adopted or not. The difference between EBSNN variants is the used network. The parameters are set to the optimal values obtained from the experiments in Section 4.2. All the experimental results are presented in Table 5. The best *Accuracy* is achieved by EBSNN-LSTM. Therefore, we finally choose EBSNN-LSTM as the default model in the subsequent sections. The conclusions are summarized as follows:

- *LSTM is preferred*: Table 5 shows that the test *Accuracy* of the 1st, 3rd, and 5th rows with LSTM is higher than that of the others with GRU. Compared with GRU, which combines the forget and input gates into a single update gate, LSTM has more gates and thus can learn more complex representations. Therefore, LSTM-based approaches are better than GRU-based ones.
- *Unidirectional RNNs are better than bidirectional ones*: Although the bidirectional RNN achieves good performance in many other fields such as NLP, the improvements provided by bidirectional RNN variants in traffic classification are poor. BiRNNs perform slightly worse than unidirectional ones. It is

perhaps due to that most operations of block cipher mode are done in one single direction [52]. BiLSTM just puts two independent LSTMs together [53]. Roughly speaking BiLSTM requires twice the computation amount of LSTM. EBSNN-BiLSTM does not have any performance improvement over EBSNN-LSTM and even consumes more computational resources than EBSNN-LSTM. Taking these two measures into consideration, unidirectional RNNs are more suitable for network traffic classification tasks than bidirectional ones in the real world, especially in the online scenario.

- *Side-channel features can improve the performance*: Table 5 shows that our new method EBSNN gets higher *Accuracy* than the previous method, BSNN, by 5 percent. We can see from the last four rows that the *Accuracy* of the methods with headers in the 3rd and 4th rows is obviously better than that of those without headers in the 5th and 6th rows. It indicates the side-channel features learned from headers make positive and signification contributions to the classification task.

4.4 Application Identification Task at Packet-Level

In this section, we focus on multi-class application identification task at packet-level. We provide the detailed results of EBSNN and compare it with the state-of-art methods [11], [12]. Securitas [11] is the traditional machine learning approach, which was proposed in 2016 and became one of the best solutions in the field of traffic classification. Meanwhile, DeepPacket [12] is the typical deep learning-based method. Both of these two approaches and EBSNN identify the network traffic at the packet level, i.e., they can recognize the application of one single packet. EBSNN utilizes attention encoders to automatically learn side-channel features from headers and useful features from payloads. Compared with existing methods that depend on the manually constructed fingerprints, EBSNN automatically constructs the most suitable features as the automated fingerprinting [45], [46] of the packet. Based on the automated fingerprinting, the last softmax layer of EBSNN completes the identification. The predicted label is the label corresponded to the maximum element in the output of the softmax layer.

Securitas is a payload-based traffic classification method. It uses the LDA topic model to extract features of a specific application before feeding these features to traditional supervised machine learning methods such as SVM, C4.5 Decision Tree, the Bayes Network, respectively. Securitas is a classic traffic classification method with traditional machine learning, which is dedicated to searching for proper features to construct machine learning classifiers.

DeepPacket exploits the payload content in the packet as input to deep learning models, including the multilayer perceptron, the one-dimension CNN, and the stacked autoencoder. As a framework that leverages deep learning to automatically learn useful representations from packets, DeepPacket opened up a trend to utilize deep learning models in the classification tasks of the network traffic.

4.4.1 Experiment Setup

We conduct experiments on dataset **D1**, and the parameters of EBSNN are set to those in Section 4.2. In the test phase, our method takes a packet as input and outputs its class label. As there are 29 kinds of applications, our method is a multi-class classifier, which can directly classify packets to one of them. In EBSNN-GRU, the attention encoder is constructed with unidirectional GRU units. Similarly, EBSNN-LSTM refers to the EBSNN method with the attention encoders of unidirectional LSTM units.

Besides, we set parameters of Securitas according to [11]. The original Securitas is a binary classifier. For the multi-classification work, it treats the packets of all the non-target applications as the negative samples. Then multiple different Securitas models for different target applications are built for multi-classification. Generally, multi-classification is more difficult than binary-classification on the same data. Thus, the comparison results of EBSNN and Securitas are convincing. In addition, for DeepPacket, the hyperparameters and the preprocessing are the same as [12].

4.4.2 Experimental Results

The experimental results are shown in Table 6, where the best results are in bold. Metrics $R.$, $P.$ and $F_1.$ represent $Recall$, $Precision$, and $F_1\ score$, respectively. Securitas-SVM, Securitas-C4.5 and Securitas-Bayes refer to the Securitas classification methods with SVM, C4.5 Decision Tree and Bayes Network, respectively. DP-SAE and DP-CNN stand for DeepPacket with the stack autoencoder and the one-dimension CNN, respectively.

Table 6 shows that in terms of $Recall$, $Precision$, and $F_1\ score$, EBSNN-LSTM achieves nearly all the best results for all the applications, whereas the other best results are achieved by EBSNN-GRU. Further, EBSNN-LSTM and EBSNN-GRU get the first and second highest $Accuracy$, respectively. Among them, the highest one is 0.9996. Overall, EBSNN gets the best result on all the metrics (i.e., $Recall$, $Precision$, $F_1\ score$, and $Accuracy$) and outperforms the state-of-the-art methods [11], [12]. In the case of encrypted traffic classification, the deep LSTM and GRU networks learn and enjoy non-linear transformations which are analogous to those applied in encryption algorithms, especially the block cipher algorithm. Our method adopts LSTM and GRU and thus achieves the best performance. Similar to the experimental results shown in Section 4.2, the scores of EBSNN-GRU in this experiment are slightly lower than EBSNN-LSTM. Specifically, compared with EBSNN-LSTM, EBSNN-GRU merely gets higher $Precision$ on Baidu and AIM Chat. Meanwhile, better $Recall$ scores are obtained by EBSNN-GRU in regard to only PPLive. With respect to $Recall$, $Precision$, and $F_1 score$, EBSNN-GRU only achieves the same results as EBSNN-LSTM on Google. However, as for $F_1\ score$ which takes both $Precision$ and $Recall$ into account, EBSNN-LSTM outperforms other methods including EBSNN-GRU for all the applications. On the other hand, EBSNN-GRU gets the same results as EBSNN-LSTM for Google. In general, the results indicate that LSTM can capture characteristic pattern better than GRU. One of the reason is that LSTM has more gates than GRU. In fact, GRU combines the forget and input gates into a single update gate. In the view of different applications, all the methods get higher scores on Sina UC and MS-SQL than those on the other applications.

TABLE 6
Experimental Results of the Application Identification Task on Dataset **D1**

ID	Application	Securitas-C4.5			Securitas-SVM			Securitas-Bayes			DP-SAE		
		$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$
1	Vimeo	0.9801	0.9875	0.9838	0.9167	0.9625	0.9390	0.9106	0.9675	0.9382	0.9133	0.9562	0.9343
2	Spotify	0.9556	0.9675	0.9615	0.8990	0.8675	0.8830	0.8990	0.8900	0.8945	0.7682	0.7643	0.7662
3	VoipBuster	0.7813	0.9825	0.8704	0.5783	0.9975	0.7321	0.7007	0.9950	0.8223	0.9908	0.9931	0.9920
4	Sina UC	0.9807	0.8875	0.9318	0.9398	0.8200	0.8758	0.9368	0.8525	0.8927	0.9747	0.9945	0.9845
5	Netease Cloud Music	0.9490	0.9300	0.9394	0.9968	0.7750	0.8720	0.9742	0.8500	0.9079	0.7252	0.9361	0.8173
6	Sina Weibo	0.7073	0.9425	0.8081	0.6365	0.9675	0.7679	0.6812	0.9350	0.7882	0.6595	0.8009	0.7233
7	Baidu	0.6913	0.9350	0.7949	0.6770	0.9325	0.7844	0.6893	0.8875	0.7760	0.6166	0.5921	0.6041
8	Tudou	0.7255	0.9450	0.8208	0.9427	0.5350	0.6826	0.8960	0.5600	0.6892	0.8518	0.8060	0.8282
9	Amazon	0.6893	0.9375	0.7945	0.6437	0.9350	0.7625	0.6692	0.8800	0.7603	0.6489	0.1889	0.2926
10	Thunder	0.7451	0.9575	0.8381	0.8379	0.6075	0.7043	0.7922	0.6575	0.7186	0.7880	0.7755	0.7817
11	Gmail	0.9281	0.6450	0.7611	0.8621	0.5000	0.6329	0.8258	0.5450	0.6566	0.5475	0.1808	0.2719
12	PPLive	0.9529	0.9100	0.9309	0.9969	0.8000	0.8877	0.9971	0.8625	0.9249	0.9619	0.9354	0.9485
13	QQ	0.9016	0.6875	0.7801	0.8773	0.5900	0.7055	0.8796	0.6025	0.7151	0.7227	0.3535	0.4748
14	Taobao	0.7034	0.9250	0.7991	0.6443	0.9600	0.7711	0.6888	0.9350	0.7932	0.7014	0.7132	0.7072
15	YahooMail	0.7490	0.9400	0.8337	0.6736	0.9650	0.7934	0.6878	0.9475	0.7971	0.8619	0.9241	0.8920
16	iTunes	0.7058	0.9475	0.8090	0.6594	0.9775	0.7875	0.6797	0.9550	0.7942	0.6395	0.3501	0.4525
17	Twitter	0.9705	0.9875	0.9789	0.9459	0.9175	0.9315	0.9397	0.9350	0.9373	0.6448	0.8486	0.7328
18	JD	0.6861	0.9125	0.7833	0.6562	0.9400	0.7729	0.6904	0.8975	0.7804	0.5623	0.1737	0.2654
19	Sohu	0.7050	0.9200	0.7983	0.8922	0.4550	0.6026	0.8683	0.5275	0.6563	0.8483	0.8531	0.8507
20	Youtube	0.9777	0.9850	0.9813	0.9442	0.9300	0.9370	0.9398	0.9375	0.9387	0.9416	0.9293	0.9354
21	Youku	0.9079	0.7150	0.8000	0.9261	0.5950	0.7245	0.8958	0.6450	0.7500	0.8315	0.8916	0.8605
22	Netflix	0.9725	0.9725	0.9725	0.9530	0.8625	0.9055	0.9251	0.9575	0.9410	0.9667	0.9776	0.9721
23	Aimchat	0.7446	0.9550	0.8368	0.9696	0.5575	0.7079	0.9405	0.5925	0.7270	0.7118	0.5379	0.6128
24	Kugou	0.9860	0.8775	0.9286	**1.0000**	0.8225	0.9026	0.9941	0.8425	0.9120	0.9926	0.9602	0.9761
25	Skype	0.7538	0.9800	0.8522	0.9157	0.1900	0.3147	0.7180	0.9675	0.8243	0.9534	0.9743	0.9637
26	Facebook	0.9975	0.9875	0.9925	0.9921	0.9375	0.9640	0.9874	0.9800	0.9837	0.9762	0.9911	0.9836
27	Google	0.9850	0.9825	0.9837	0.9475	0.9475	0.9475	0.9478	0.9525	0.9501	0.6290	0.3659	0.4626
28	MS-SQL	0.9900	0.9950	0.9925	0.9924	0.9775	0.9849	0.9949	0.9775	0.9861	0.9594	0.9447	0.9520
29	MS-Exchange	0.9813	0.7645	0.8595	0.9881	0.7238	0.8356	0.9806	0.7355	0.8405	0.8342	0.4826	0.6114
-	Average	0.8553	0.9159	0.8765	0.8588	0.7948	0.7970	0.8528	0.8369	0.8309	0.8008	0.7309	0.7466
-	$Accuracy$		0.8674			0.8082			0.8325			0.9169	

ID	Application	DP-CNN			EBSNN-GRU			EBSNN-LSTM		
		$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$
1	Vimeo	0.9743	0.9820	0.9781	0.9959	0.9967	0.9963	**0.9984**	**0.9991**	**0.9988**
2	Spotify	0.9511	0.9108	0.9305	0.9776	0.9771	0.9773	**0.9935**	**0.9915**	**0.9925**
3	VoipBuster	0.9994	0.9974	0.9984	0.9981	**0.9998**	0.9989	**0.9991**	**0.9998**	**0.9995**
4	SinaUC	0.9998	0.9992	0.9995	0.9999	0.9999	0.9999	**1.0000**	**1.0000**	**1.0000**
5	Netease Cloud Music	0.9785	0.9928	0.9856	0.9985	0.9994	0.9990	**0.9999**	**0.9997**	**0.9998**
6	Sina Weibo	0.9519	0.9796	0.9656	0.9994	0.9969	0.9981	**1.0000**	**0.9998**	**0.9999**
7	Baidu	0.9227	0.9284	0.9255	0.9865	**0.9993**	0.9929	**0.9999**	0.9985	**0.9992**
8	Tudou	0.9880	0.9899	0.9890	0.9985	0.9992	0.9989	**0.9996**	**0.9998**	**0.9997**
9	Amazon	0.9250	0.8737	0.8986	0.9993	**1.0000**	0.9997	**1.0000**	**1.0000**	**1.0000**
10	Thunder	0.9737	0.9955	0.9845	0.9990	0.9973	0.9981	**0.9994**	**0.9997**	**0.9995**
11	Gmail	0.8653	0.9109	0.8876	0.9886	0.9523	0.9701	**0.9900**	**0.9945**	**0.9923**
12	PPLive	0.9932	0.9829	0.9880	**0.9999**	0.9972	0.9985	0.9998	**0.9999**	**0.9999**
13	QQ	0.9535	0.9203	0.9366	0.9897	0.9918	0.9907	**0.9982**	**0.9975**	**0.9979**
14	Taobao	0.9486	0.9107	0.9293	0.9992	0.9994	0.9993	**0.9997**	**0.9997**	**0.9997**
15	YahooMail	0.9958	0.9854	0.9906	0.9994	0.9992	0.9993	**1.0000**	**1.0000**	**1.0000**
16	iTunes	0.9529	0.9125	0.9323	0.9914	0.9975	0.9944	**0.9994**	**1.0000**	**0.9997**
17	Twitter	0.9726	0.9898	0.9811	**0.9995**	0.9989	0.9992	**0.9995**	**1.0000**	**0.9997**
18	JD	0.8763	0.9251	0.9000	0.9985	0.9990	0.9987	**0.9992**	**0.9997**	**0.9995**
19	Sohu	0.9795	0.9759	0.9777	0.9974	0.9911	0.9942	**0.9999**	**0.9986**	**0.9992**
20	Youtube	0.9806	0.9794	0.9800	0.9930	0.9969	0.9950	**0.9980**	**0.9978**	**0.9979**
21	Youku	0.9863	0.9873	0.9868	0.9990	0.9982	0.9986	**0.9995**	**0.9999**	**0.9997**
22	Netflix	0.9958	0.9922	0.9940	0.9989	0.9982	0.9985	**0.9994**	**0.9991**	**0.9992**
23	Aimchat	0.9206	0.9299	0.9252	0.9606	**0.9967**	0.9783	**0.9945**	0.9962	**0.9954**
24	Kugou	0.9982	0.9942	0.9962	0.9996	0.9996	0.9996	0.9999	**0.9999**	**0.9999**
25	Skype	0.9965	0.9983	0.9974	0.9997	0.9985	0.9991	**0.9998**	**0.9993**	**0.9996**
26	Facebook	0.9994	0.9996	0.9995	0.9998	0.9999	0.9999	**0.9999**	**1.0000**	**1.0000**
27	Google	0.9856	0.9608	0.9731	**0.9978**	**1.0000**	**0.9989**	**0.9978**	**1.0000**	**0.9989**
28	MS-SQL	0.9947	0.9966	0.9957	0.9990	0.9995	0.9993	**1.0000**	**1.0000**	**1.0000**
29	MS-Exchange	0.9788	0.9390	0.9585	**0.9971**	0.9971	0.9971	**0.9971**	**1.0000**	**0.9985**
-	Average	0.9668	0.9635	0.9650	0.9952	0.9957	0.9954	**0.9987**	**0.9990**	**0.9988**
-	$Accuracy$		0.9887			0.9990			**0.9996**	

4.4.3 Discussion

As shown in Table 6, Securitas has bad performance for all the target applications. For Securitas-SVM, the reason is perhaps that SVM is not good at dealing with the imbalanced classification problem [54]. Note that in the experiments the sample numbers of different applications are quite imbalanced when it comes to binary classification.

Moreover, it is worth mentioning that Securitas is a binary classifier and we build multiple Securitas models for every target class. EBSNN and DeepPacket classify all the packets to multiple classes directly. Comparing with traditional machine learning methods (Securitas), deep learning methods (DeepPacket and EBSNN) obtain better performance. The reason may be that deep learning methods learn

TABLE 7
Experimental Results of the Website Identification Task on Dataset **D2**

ID	Application	Securitas-C4.5			Securitas-SVM			Securitas-Bayes			DP-SAE		
		$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$
1	Reddit	0.9020	0.8975	0.8997	0.7946	0.9575	0.8685	0.7780	0.9725	0.8644	0.3067	0.5104	0.3832
2	Facebook	0.8475	0.8475	0.8475	0.7377	0.9775	0.8409	0.8005	0.7725	0.7863	0.6638	0.5459	0.5991
3	Netease Cloud Music	0.9237	0.9075	0.9155	0.8247	0.8000	0.8122	0.8240	0.7725	0.7974	0.6641	0.6120	0.6370
4	Twitter	0.8990	0.8900	0.8945	0.8803	0.8275	0.8531	0.8549	0.8250	0.8397	0.6816	0.4774	0.5615
5	QQ Mail	0.8766	0.8525	0.8644	0.8004	0.9525	0.8699	0.7375	0.9625	0.8351	0.2832	0.6148	0.3878
6	Instagram	0.8730	0.8250	0.8483	0.8116	0.8075	0.8095	0.8104	0.7800	0.7949	0.2362	0.5616	0.3326
7	Sina Weibo	0.8665	0.8600	0.8632	0.7984	0.9700	0.8758	0.7365	0.8175	0.7749	0.6397	0.6336	0.6366
8	iQIYI	0.8699	0.9025	0.8859	0.9151	0.7275	0.8106	0.7804	0.7375	0.7584	0.8459	0.7035	0.7682
9	IMDb	0.9483	0.9175	0.9327	0.9776	0.7650	0.8583	0.9271	0.7625	0.8368	0.8929	0.8995	0.8962
10	TED	0.9620	0.9500	0.9560	0.9834	0.8875	0.9330	0.9464	0.8825	0.9133	0.8113	0.8197	0.8155
11	Douban	0.9173	0.9150	0.9161	0.8881	0.9125	0.9001	0.8356	0.9150	0.8735	0.7562	0.8554	0.8027
12	Amazon	0.8811	0.8525	0.8666	0.7177	0.9850	0.8303	0.7186	0.9450	0.8164	0.4801	0.4238	0.4502
13	Youtube	0.8667	0.8775	0.8720	0.7677	0.9500	0.8492	0.7500	0.8325	0.7891	0.8210	0.6382	0.7182
14	JD	0.9466	0.9300	0.9382	0.8499	0.9200	0.8836	0.7795	0.8750	0.8245	0.8578	0.8621	0.8599
15	Youku	0.9347	0.9300	0.9323	0.9102	0.9625	0.9356	0.8717	0.9000	0.8856	0.6020	0.5717	0.5864
16	Baidu	0.9480	0.9575	0.9527	0.9642	0.9425	0.9532	0.9229	0.9275	0.9252	0.9215	0.9113	0.9163
17	Google	0.9538	0.9300	0.9418	0.8250	0.9900	0.9000	0.8041	0.9850	0.8854	0.2421	0.4288	0.3095
18	Tieba	0.9233	0.9325	0.9279	0.9295	0.9225	0.9260	0.9138	0.7950	0.8503	0.7696	0.8015	0.7852
19	Taobao	0.8867	0.9000	0.8933	0.7918	0.9600	0.8678	0.7742	0.9600	0.8571	0.5177	0.7006	0.5954
20	Bing	0.9128	0.8900	0.9013	0.9197	0.8875	0.9033	0.9054	0.8850	0.8951	0.8542	0.8208	0.8372
-	Average	0.9070	0.8982	0.9025	0.8544	0.9052	0.8740	0.8236	0.8652	0.8402	0.6424	0.6696	0.6439
-	*Accuracy*		0.9029			0.8744			0.8406			0.6716	

ID	Application	DP-CNN			EBSNN-GRU			EBSNN-LSTM		
		$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$	$R.$	$P.$	$F_1.$
1	Reddit	0.9297	0.9269	0.9283	**1.0000**	0.9997	0.9998	**1.0000**	**1.0000**	**1.0000**
2	Facebook	0.9682	0.9365	0.9520	**1.0000**	0.9996	0.9998	**1.0000**	**1.0000**	**1.0000**
3	Netease Cloud Music	0.7852	0.8528	0.8176	0.9911	0.9891	0.9901	**0.9965**	**0.9977**	**0.9971**
4	Twitter	0.9444	0.9670	0.9556	**1.0000**	0.9999	**1.0000**	**1.0000**	**1.0000**	**1.0000**
5	QQ Mail	0.5496	0.7178	0.6225	0.9791	0.9782	0.9786	**0.9955**	**0.9912**	**0.9934**
6	Instagram	0.8506	0.9060	0.8774	**1.0000**	0.9999	0.9999	**1.0000**	**1.0000**	**1.0000**
7	Sina Weibo	0.9311	0.9449	0.9380	0.9978	0.9968	0.9973	**0.9990**	**0.9990**	**0.9990**
8	iQIYI	0.9228	0.9429	0.9327	0.9958	0.9964	0.9961	**0.9991**	**0.9989**	**0.9990**
9	IMDb	0.9622	0.9850	0.9734	**0.9998**	0.9990	0.9994	**0.9998**	**0.9997**	**0.9998**
10	TED	0.9871	0.9708	0.9789	0.9996	0.9992	0.9994	**0.9999**	**0.9997**	**0.9998**
11	Douban	0.8958	0.9154	0.9055	0.9981	0.9981	0.9981	**0.9995**	**0.9992**	**0.9994**
12	Amazon	0.9591	0.9247	0.9416	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
13	Youtube	0.9804	0.9742	0.9773	0.9998	**1.0000**	0.9999	**1.0000**	**1.0000**	**1.0000**
14	JD	0.9422	0.9779	0.9597	0.9935	0.9974	0.9954	**0.9983**	**0.9985**	**0.9984**
15	Youku	0.8772	0.7596	0.8142	0.9874	0.9883	0.9879	**0.9955**	**0.9961**	**0.9958**
16	Baidu	0.9367	0.9776	0.9567	0.9946	0.9973	0.9960	**0.9991**	**0.9991**	**0.9991**
17	Google	0.8916	0.7599	0.8205	0.9993	0.9998	0.9996	**1.0000**	**1.0000**	**1.0000**
18	Tieba	0.8717	0.7593	0.8116	0.9905	0.9925	0.9915	**0.9965**	**0.9984**	**0.9974**
19	Taobao	0.8984	0.8989	0.8987	0.9925	0.9911	0.9918	**0.9969**	**0.9970**	**0.9969**
20	Bing	0.9580	0.9357	0.9467	0.9971	0.9975	0.9973	**0.9992**	**0.9994**	**0.9993**
-	Average	0.9021	0.9017	0.9005	0.9958	0.9960	0.9959	**0.9987**	**0.9987**	**0.9987**
-	*Accuracy*		0.9203			0.9967			**0.9989**	

high-level representations which can characterize the application well.

Moreover, as for the comparison with other deep learning-based methods, EBSNN outperforms DeepPacket in almost all the classes on both datasets. The impressive performance of EBSNN is owing to the ability and the structure of EBSNN to deal with complex sequences. Besides, EBSNN represents better robustness and generalization than DeepPacket according to the result for every application. Compared with BSNN, EBSNN introduces and leverages side-channel features from headers and thus improves the performance by a considerable margin.

4.5 Website Identification Task at Packet-Level

Besides the native application, website is another form to provide the services. Websites use a lot of public resources cached in the Content Delivery Network (CDN) which bring more noise network traffic than the native application. Therefore, it is necessary to evaluate EBSNN in the website identification task. The $Precision, Recall$, and F_1 *score* results of the website identification task on dataset **D2** are shown in Table 7. The overall *Accuracy* results are also displayed in the bottom row of the table. The best results among different methods are in bold. All of the best results come from our proposed method. The overall *Accuracy* of EBSNN, i.e., 0.9989, is significantly better than those of the other methods. Different from the results in the application identification task on dataset **D1**, the average F_1 *score* of Securitas-C4.5 is slightly higher than DeepPacket-SAE and DeepPacket-CNN in the website identification task. We can see that the results of DeepPacket-SAE are even the worst. This means that DeepPacket-SAE is not suitable to the website identification task. The poor performance might be due to the poor robustness against noisy network traffic.

Almost all the websites except QQ Mail have F_1 *score* higher than 0.95. The website QQ Mail is a popular Chinese e-mail website which is similar to Gmail. For the slightly

TABLE 8
Statistical Information of the Flow Dataset

Application	#Flows	#UDP Flows	#TCP Flows	#Packets
Skype	4119	3932	187	121015
PPLive	2666	2441	225	146453
Baidu	762	0	762	29547
Tudou	662	313	349	88784
Sina Weibo	627	1	626	39212
Thunder	501	379	122	28942
Youku	492	340	152	85495
iTunes	467	0	467	17319
Taobao	392	0	392	30897
QQ	317	43	274	13469
Gmail	310	274	36	7147
Sohu	292	165	127	32487

unsatisfactory result of QQ Mail, we find that most of its traffic comes from the mails the account received. The e-mail whose format is HTML (a format of rich text) utilizes large amount of static resources from the sender's server other than the QQ Mail server. For example, if we receive an advertisement e-mail which comes from Apple, when we check this e-mail in QQ Mail, the browser loads many static resources, such as image, CSS and font files, from the Apple server. Thus, there is considerable quantity of noisy traffic from other servers such as Apple, which lead to low $F_1\ score$ for QQ Mail.

4.6 Application Identification Task at Flow-Level

To validate the ability of EBSNN to identify the traffic flows, we select the applications which have more than 290 flows in dataset **D1**. Table 8 shows the statistics of the flow dataset. Most of the applications contain both the connection-oriented (TCP) flows and the connection-less (UDP) flows in the dataset. Similar to the packet dataset, the number of packets and flows among different applications are imbalanced. In the experiments, we adopt the aggregate strategy of the sum of the softmax output. The flow-level experimental results are listed in Table 9.

Table 9 shows that all the applications except Sina Weibo (a Chinese social networking service like Twitter), 0.8861, and Youku (an online video-sharing platform like Youtube), 0.8804, obtain $F_1\ score$ higher than 0.91. The result of Sina Weibo can be explained by that the contents in Sina Weibo often contain shared videos from other applications such as Youku. Specially, PPLive and Skype get best scores which are close to 1.0. Such high scores may be caused by the relatively larger sample amount of these two applications. With the power of Focal Loss, our proposed method can work well on the imbalanced dataset, which achieves average $F_1\ score$ of 0.9532 and $Accuracy$ of 0.9724. When the number of the training samples of the application decreases, the performance of EBSNN does not degrade. For example, applications such as SOHU (0.9913 in $F_1\ score$) and Gmail (0.9667 in $F_1\ score$) still achieve excellent results while the amounts of the training samples are the least in the dataset.

TABLE 9
Experimental Results in Flow-Level Classification Task

Application	$Precision$	$Recall$	$F_1\ score$
Skype	0.9940	1.0000	0.9970
PPLive	1.0000	0.9981	0.9991
Baidu	0.8889	0.9412	0.9143
Tudou	0.8776	0.9699	0.9214
Sina Weibo	0.9459	0.8333	0.8861
Thunder	0.9804	1.0000	0.9901
Youku	0.9529	0.8182	0.8804
iTunes	1.0000	0.9785	0.9891
Taobao	0.8837	0.9744	0.9268
QQ	0.9839	0.9683	0.9760
Gmail	1.0000	0.9355	0.9667
Sohu	1.0000	0.9828	0.9913
Average	0.9589	0.9500	0.9532

According to Tables 8 and 9, it can be concluded that our approach can classify both the connection-oriented flows and the connection-less flows.

4.7 Understanding Classifier Behavior

With the design of the attention encoder in ESBNN, we can easily obtain the importance of different bytes and segments in the packet. In this section, we analyze the behavior of the proposed model through the attention output of EBSNN.

As shown in Formula (2), the output of the attention encoder is analogous to the weighted sum of the input. The attention weight estimates how important one element is for the classification task. In EBSNN, there are two kinds of attention encoders: byte-level and segment-level. The output of the byte-level attention encoder is the weighted sum of all the hidden representations of bytes in the segment, where the weight measures the importance of the byte w.r.t. the segment. Similarly, the weight in the segment-level attention encoder measures the importance of the segment w.r.t. the network packet. Therefore, we can obtain the importance of each byte w.r.t. the network packet by combining the byte-level and segment-level attentions.

Table 10 shows typical cases for explaining the information EBSNN learned. Here, we list the most important segments in the packet. Two hex digits present one byte. The background color indicates how important the byte is. The darker the color, the more important. The text under the flipped brace is the description of the field. The results demonstrate that the proposed EBSNN has successfully captured useful side-channel features, including TLS metadata and length. Case-1 is the TLS Client Hello Handshake packet, whose label is Taobao. TLS server name extension is widely used by almost all mainstream websites [55]. The attention weights show that the most important part is the Server Name field, whose value is "rate.taobao.com". Likewise, Cipher Suite, id-at-commonName, and Common Name String ("Micros" is the beginning part of "Microsoft Corporation") in Case-2 (a TLS Server Hello packet) achieve

TABLE 10
Case Analysis for Explanation on Dataset **D2**

Cases	ID	Label	Explanation
TLS metadata	1	Taobao	Segment 11: 00 00 (Type: server_name) 00 14 (Length) 00 12 (List length) 00 (Type: host_name) 00 0F (Length) 72 61 74 65 2E 74 61 (Server Name: rate.ta) Segment 12: 6F 62 61 6F 2E 63 6F 6D (Server Name (cond.): obao.com) 00 17 (Type: extended_master_secret) 00 00 (Length) FF 01 (Type: renegotiation_info) 00 01 (Length)
	2	Bing	Segment 8: 57 78 C1 49 (Session ID) C0 30 (Cipher Suite) 00 (Compression Method) 00 0D (Extensions Length) 00 05 00 00 (status_request) 00 17 00 (Type: extended_master_secret) Segment 20: 1E 30 1C 06 03 (RelativeDistinguishedName) 55 04 03 (id-at-commonName) 13 15 4D 69 63 72 6F 73 (String: Micros
Length	3	Baidu	Segment 1: 45 (IP Header Len) 00 0A 39 (Total Length) 00 00 00 00 7E (TTL) 06 (Protocol) 00 00 00 00 00 00
	4	iQIYI	Segment 3: C0 (TCP Header Len) 18 (TCP Flag) 5E 0D (Window Size) 85 D0 (Checksum) 00 00 (Urgent Point) 00 00 00 00 00 00 00 00 (TCP Options)

**The background color indicates the importance of the byte. The darker the color, the more important.*

the highest weights, which are coincident with the related work [25]. Case-3 demonstrates that EBSNN successfully treated the well-known Total Length as the important information for the classification.

In addition to the well-known side-channel features, EBSNN also suggests that IP header length, TTL (Time-To-Live), and Protocol fields provide useful information. According to the result, TCP Header Length is the most significant information in Case-4. Because iQIYI enables TCP SACK but disables TCP Timestamps Option, the packet of iQIYI has TCP Options of 28 bytes, including two TCP NOP Options and one TCP SACK Option with 3 SACK blocks. Different websites may have various configurations of TCP Options and thus have different TCP header lengths. The result also indicates that Window Size is helpful. TCP window size is used to control the transfer rate of data between client and server. Different operating systems (even different versions) may adopt diverse default TCP window sizes.

It can be seen that the proposed method can learn the most suitable combination of side-channel features and hidden features from the payload.

4.8 Comparison of Testing Time

The classification speed is an important to the online classification of network traffic. Hence, we compare the classification speed of EBSNN with those of the traditional machine learning approach [11] and also the deep learning method [12]. Since training is conducted only once and can be done offline, running time is measured by testing time. The comparison results are shown in Table 11. The results come from the website identification task at packet-level. The data in columns, "Extracting Features" and "Prediction", of the table are in microseconds per packet (μs/pkt). The column "Prediction" in the table refers to the evaluation time of deep learning models or machine learning algorithms to calculate the predicted label. The term "Extracting Features" denotes the time of extracting features in machine learning algorithms (e.g., SVM).

In the testing phase, the average times to process and identify a single packet in our method are $33.02\,\mu s$ (EBSNN-LSTM) and $46.78\,\mu s$ (EBSNN-GRU). In Securitas, there are three different traditional machine learning classifiers, i.e., SVM, C4.5 Decision Tree and Naive Bayes. The average time of SVM is 47.38 μs/pkt. It is obvious that the classification speed of EBSNN can match with that of some machine learning algorithms and even faster. As for C4.5 Decision Tree and Naive Bayes, they both have very short running time with $0.33\,\mu s$/pkt and 0.23 μs/pkt, respectively. It is worth mentioning that the machine learning-based method Securitas need manually extract features. To extract features from one packet, Securitas takes 984.33 μs on average, which is very time consuming. Hence, our method is about 30 times faster than Securitas. As for DeepPacket, the average times to process and identify one packet are $3.59\,\mu s$ (DP-CNN) and $2.29\,\mu s$ (DP-SAE). Compared with EBSNN, DeepPacket utilizes simpler neural networks with less computations and thus spends less time. In future work, we will optimize EBSNN with less computation resource. In general, EBSNN gets high *Accuracy* in all tasks while keeping acceptable classification speed.

In consideration of real-world scenarios, we also conduct experiments of throughput under limited resources. We choose three devices as the test platform: (1) Laptop with Intel i5 4210U CPU and NVIDIA 860M GPU, (2) NVIDIA Jetson Nano with 128-core NVIDIA Maxwell GPU and Quad-core ARM A57 CPU, (3) NVIDIA Jetson Xavier with

TABLE 11
Comparison Results of Testing Time

Methods	Extracting Features (μs/pkt)	Prediction (μs/pkt)
Securitas-C4.5	984.33	0.33
Securitas-SVM		47.38
Securitas-Bayes		0.23
DP-SAE	-	3.59
DP-CNN	-	2.29
EBSNN-LSTM	-	33.02
EBSNN-GRU	-	46.78

TABLE 12
Throughput Under Limited Resources

Configuration		Throughput
GPU	Laptop nVidia 860M	409.836 Mbps
	NVIDIA Jetson Xavier (GPU)	61.829 Mbps
	NVIDIA Jetson Nano (GPU)	34.897 Mbps
CPU	Laptop Intel i5 4210U	750.550 Kbps
	Laptop Intel i5 4210U w/ onnx	1,815.631 Kbps
	NVIDIA Jetson Xavier (CPU)	297.725 Kbps
	NVIDIA Jetson Nano (CPU)	177.970 Kbps

3840-core NVIDIA Volta GPU and 6-core ARMv8 CPU. NVIDIA 860M GPU is a mid-range and out-of-date graphics card for laptops unveiled in 2014. We obtain the testing time per packet in these platforms. The equivalent throughput is estimated from the testing time according to [56]. The throughput results on different low-end devices are shown in Table 12. If we deploy EBSNN running on CPU, the throughput is relatively low and unsuitable in real-world scenarios. The throughput of Intel i5 4210U is only 750.55 Kbps. This result is evaluated with Pytorch, which is not optimized for inference. With the help of onnx (a high-performance inference engine),[3] the throughput is more than twice that of Pytorch, reaching 1,815.631 Kbps. However, neural networks are naturally better suited to run on GPU. Even on NVIDIA Jetson Nano, the throughput of EBSNN running on GPU is about 34.897 Mbps. If EBSNN runs on NVIDIA Jetson Xavier, the throughput on GPU is 61.829 Mbps. The throughput on the old laptop with NVIDIA 860M GPU jumps to 409.836 Mbps. It can be seen that EBSNN achieves acceptable throughput in many real-time scenarios with the help of GPU in some low-end devices.

5 Conclusion

We propose a new neural network model, the Extended Byte Segment Neural Network (EBSNN), for network traffic classification. Besides the payload of the packet, EBSNN utilizes the side-channel features and employs hierarchical attention networks to learn the high-level presentation. Our method can work well on both packet-level and flow-level for the application identification tasks. It is also suitable for website identification tasks. Experiments on the real world network traffic show that EBSNN achieves an outstanding performance, surpassing the state-of-the-art methods. We have released the source code of EBSNN at GitHub.[4]

In terms of the limitation of this work, our algorithm does not leverage additional techniques to resist noisy condition and adversarial attacks[57], [58] on DNN. We will incorporate pre-processing techniques [59] and detection methods [60] to enhance the robustness of our approach in future work. In addition, we will further study the optimization for less computation resource usages in the future work. Possible solutions include knowledge distillation for sequence model [61].

3. https://onnx.ai/
4. https://github.com/DCMMC/EBSNN

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1800204, in part by the National Natural Science Foundation of China under Grants 61972219, 61773229, and 61771273, in part by the Hong Kong RGC under Projects 152279/16E and 152239/18E, and HK ITF under Project GHP/052/19SZ, in part by the Research and Development Program of Shenzhen under Grants JCYJ20190813174403598, SGDX20190918101201696, and JCYJ20190813165003837, in part by the National Key Research and Development Program of China under Grant 2018YFB1800601, and in part by the Overseas Research Cooperation Fund of Tsinghua Shenzhen International Graduate School under Grant HW2021013.

References

[1] *Cisco WAN and application optimization solution guide*. San Jose, USA: Cisco Systems Inc. (2008). Accessed: Apr. 2, 2018. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/nsite/enterprise/wan/wan_optimization/wan_opt_sg.html
[2] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci, "HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2011, pp. 1–20.
[3] X. Luo, P. Zhou, J. Zhang, R. Perdisci, W. Lee, and R. K. C. Chang, "Exposing invisible timing-based traffic watermarks with backlit," in *Proc. 27th Annu. Comput. Secur. Appl. Conf.*, 2011, pp. 197–206.
[4] J. Zhang, R. Perdisci, W. Lee, X. Luo, and U. Sarfraz, "Building a scalable system for stealthy P2P-botnet detection," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 1, pp. 27–38, Jan. 2014.
[5] Z. Li *et al.*, "NetShield: Massive semantics-based vulnerability signature matching for high-speed networks," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 279–290, 2010.
[6] T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Commun. Surv. Tut.*, vol. 10, no. 4, pp. 56–76, Oct.-Dec. 2008.
[7] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.
[8] G. Stergiopoulos, A. Talavari, E. Bitsikas, and D. Gritzalis, "Automatic detection of various malicious traffic using side channel features on TCP packets," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2018, pp. 346–362.
[9] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, Feb. 2020.
[10] R. Li, X. Xiao, S. Ni, H. Zheng, and S. Xia, "Byte segment neural network for network traffic classification," in *Proc. IEEE/ACM 26th Int. Symp. Qual. Serv.*, Banff, AB, Canada, 2018, pp. 1–10.
[11] 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.
[12] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Comput.*, vol. 24, pp. 1999–2012, 2019.
[13] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 1, pp. 63–78, Jan. 2018.
[14] X. Luo, J. Zhang, R. Perdisci, and W. Lee, "On the secrecy of spread-spectrum flow watermarks," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2010, pp. 232–248.
[15] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, New York, NY, USA, 2004, pp. 135–148.
[16] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proc. ACM CoNEXT Conf.*, New York, NY, USA, 2006, pp. 1–12.

[17] 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. Servi. Manage.*, vol. 10, no. 2, pp. 133–147, Jun. 2013.

[18] L. Sacramento, I. Medeiros, J. Bota, and M. Correia, "FlowHacker: Detecting unknown network attacks in big traffic data using network flows," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./ 12th IEEE Int. Conf. Big Data Sci. Eng.*, New York, NY, USA, 2018, pp. 567–572.

[19] T. Chen *et al.*, "TokenScope: Automatically detecting inconsistent behaviors of cryptocurrency tokens in ethereum," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1503–1520.

[20] Y. Huang *et al.*, "Understanding (mis) behavior on the EOSIO blockchain," in *Proc. ACM Meas. Anal. Comput. Syst.*, 2020, pp. 1–28.

[21] M. Shen, J. Zhang, L. Zhu, K. Xu, X. Du, and Y. Liu, "Encrypted traffic classification of decentralized applications on ethereum using feature fusion," in *Proc. Int. Symp. Qual. Serv.*, New York, NY, USA, 2019, pp. 1–10.

[22] T. Chen *et al.*, "Understanding ethereum via graph analysis," *ACM Trans. Internet Technol.*, vol. 20, pp. 1–32, 2020.

[23] J. Kampeas, A. Cohen, and O. Gurewitz, "Traffic classification based on zero-length packets," *IEEE Trans. Netw. Serv. Manage.*, vol. 15, no. 3, pp. 1049–1062, Sep. 2018.

[24] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, New York, NY, USA, 2017, pp. 1723–1732.

[25] C. Liu, Z. Cao, G. Xiong, G. Gou, S. Yiu, and L. He, "MaMPF: Encrypted traffic classification based on multi-attribute Markov probability fingerprints," in *Proc. IEEE/ACM 26th Int. Symp. Qual. Serv.*, Banff, AB, Canada, 2018, pp. 1–10.

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

[27] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "nDPI: Open-source high-speed deep packet inspection," in *Proc. Int. Wireless Commun. Mobile Comput. Conf.*, Nicosia, Cyprus, 2014, pp. 617–622.

[28] 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.

[29] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: Automated construction of application signatures," in *Proc. ACM SIGCOMM Workshop Mining Netw. Data*, New York, NY, USA, 2005, pp. 197–202.

[30] X. Xiao, R. Li, H.-T. Zheng, R. Ye, A. KumarSangaiah, and S. Xia, "Novel dynamic multiple classification system for network traffic," *Inf. Sci.*, vol. 479, pp. 526–541, 2019.

[31] C. Luo, Z. Tan, G. Min, J. Gan, W. Shi, and Z. Tian, "A novel web attack detection system for internet of things via ensemble classification," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5810–5818, Aug. 2021.

[32] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in SDN home gateway," *IEEE Access*, vol. 6, pp. 55 380–55 391, Sep. 2018.

[33] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "MIMETIC: Mobile encrypted traffic classification using multimodal deep learning," *Comput. Netw.*, vol. 165, 2019, Art. no. 106944.

[34] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "Multi-classification approaches for classifying mobile app traffic," *J. Netw. Comput. Appl.*, vol. 103, pp. 131–145, 2018.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

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

[37] K. Cho *et al.*, "Learning phrase representations using MN encoder-decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, Doha, Qatar, 2014, pp. 1724–1734.

[38] R. Lin, S. Liu, M. Yang, M. Li, M. Zhou, and S. Li, "Hierarchical recurrent neural network for document modeling," in *Proc. Conf. Empirical Methods Nat. Lang. Process.*, Lisbon, Portugal, 2015, pp. 899–907.

[39] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proc. 29th AAAI Conf. Artif. Intell.*, Austin, TX, USA, 2015, pp. 2267–2273.

[40] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proc. Conf. North Amer. Chap. Assoc. Comput. Linguistics: Hum. Lang. Technol.*, San Diego, CA, USA, 2016, pp. 1480–1489.

[41] A. Moore, M. Crogan, A. W. Moore, Q. Mary, D. Zuev, D. Zuev, M. L. Crogan, "Discriminators for use in flow-based classification," Dept. Comput. Sci., Queen Mary and Westfield College, Univ. London, London, U.K., Res. Rep. RR-05–13, Aug. 2005.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 1–15.

[43] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[44] A. Kumar *et al.*, "Ask me anything: Dynamic memory networks for natural language processing," in *Proc. 33rd Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 1378–1387.

[45] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2018, pp. 1928–1943.

[46] V. Rimmer, D. Preuveneers, M. Juarez, T. V. Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.

[47] 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*, 2019, pp. 1–11.

[48] Q. Liu and Z. Liu, "A comparison of improving multi-class imbalance for internet traffic classification," *Inf. Syst. Front.*, vol. 16, no. 3, pp. 509–521, Jul. 2014.

[49] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail," in *Proc. IEEE Symp. Secur. Privacy*, 2012, pp. 332–346.

[50] J. Nechvatal *et al.*, "Report on the development of the advanced encryption standard (AES)," *J. Res. Nat. Inst. Standards Technol.*, vol. 106, no. 3, pp. 511–577, Jun. 2001.

[51] E. Hoffer, I. Hubara, and D. Soudry, "Train longer, generalize better: Closing the generalization gap in large batch training of neural networks," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, Red Hook, NY, USA, 2017, pp. 1729–1739.

[52] P. Rogaway, "Evaluation of some blockcipher modes of operation," 2011. [Online]. Available: https://web.cs.ucdavis.edu/rogaway/papers/modes.pdf

[53] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[54] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Syst. Appl.*, vol. 73, pp. 220–239, 2017.

[55] 21+ SSL statistics that show why security matters so much. Accessed: Aug. 13, 2021. [Online]. Available: https://hostingtribunal.com/blog/ssl-stats

[56] A. J. Pinheiro, P. Freitas de Araujo-Filho, J. de M. Bezerra, and D. R. Campelo, "Adaptive packet padding approach for smart home networks: A tradeoff between privacy and performance," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3930–3938, Mar. 2021.

[57] R. Pang *et al.*, "The tale of evil twins: Adversarial inputs versus poisoned models," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 85–99.

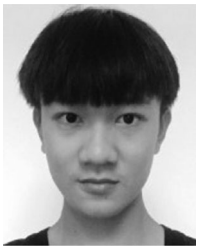
[58] X. Zhang, N. Wang, H. Shen, S. Ji, X. Luo, and T. Wang, "Interpretable deep learning under fire," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1659–1676.

[59] L. Zhang, L. Song, B. Du, and Y. Zhang, "Nonlocal low-rank tensor completion for visual data," *IEEE Trans. Cybern.*, vol. 51, no. 2, pp. 673–685, Feb. 2021.

[60] R. Pang, X. Zhang, S. Ji, X. Luo, and T. Wang, "AdvMind: Inferring adversary intent of black-box attacks," in *Proc. Conf. Knowl. Discov. Data Mining*, 2020, pp. 1899–1907.

[61] M. Huang, Y. You, Z. Chen, Y. Qian, and K. Yu, "Knowledge distillation for sequence model," in *Proc. Interspeech*, Hyderabad, India, 2018, pp. 3703–3707.

Xi Xiao received the PhD degree from the State Key Laboratory of Information Security, Graduate University of Chinese Academy of Sciences, in 2011. He is currently an associate professor with Shenzhen International Graduate School, Tsinghua University. His research interests include information security and the computer network.

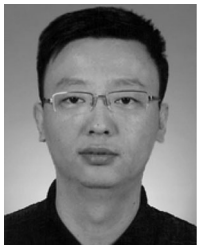


Wentao Xiao is currently studying with Shenzhen International Graduate School, Tsinghua University. His research interests include machine learning, deep learning, and information security.

Rui Li received the master's degree from Shenzhen International Graduate School, Tsinghua University in 2019. He currently working with Alibaba. His research interests include network management, deep learning, and information security.

Xiapu Luo is currently an associate professor with the Department of Computing, The Hong Kong Polytechnic University. His work appeared in top conferences and journals regarding his research interests, which include mobile and IoT security and privacy, blockchain and smart contracts, network security and privacy, and software engineering. He was a recipient of eight best paper awards, including the ACM SIGSOFT Distinguished Paper Award in ICSE'21, Best Paper Award in INFOCOM'18, and Best Research Paper Award in ISSRE'16.

Haitao Zheng received the bachelor's degree from the Department of Computer Science, Sun Yat-Sen University, in 2001, the master's degree from the Department of Computer Science, Sun Yat-Sen University, in 2004, and the PhD degree in medical informatics major from Seoul National University. His research interests include artificial intelligence, Semantic Web, information retrieval, machine learning, and medical informatics.



Shutao Xia received the BS degree in mathematics and the PhD degree in applied mathematics from Nankai University, Tianjin, China, in 1992 and 1997, respectively. Since January 2004, he has been with the Graduate School of Shenzhen, Tsinghua University, Guangdong, China, where he is currently a Professor. His research interests include coding theory, information theory, and networking.