RBLJAN: Robust Byte-Label Joint Attention Network for Network Traffic Classification

Xi Xiao, Shuo Wang, Guangwu Hu, Qing Li, Kelong Mao, Xiapu Luo, Bin Zhang and Shutao Xia

Abstract—Network traffic classification plays a crucial role in network management and cyberspace security. As the Internet evolves with new applications and protocols, traditional machine learning-based methods relying on feature mining have become obsolete. Instead, deep learning-based methods are becoming more popular in the field of traffic classification due to their end-to-end processing approach. However, the vulnerability of neural networks to adversarial examples significantly compromises their performance. In this paper, we propose Robust Byte-Label Joint Attention Network (RBLJAN), an efficient and robust deep learning-based framework for encrypted network traffic classification at both the packet-level and the flow-level. RBLJAN comprises a classifier and an adversarial traffic generator. The classifier utilizes mechanisms such as header-payload parallel processing and byte-label joint attention learning to capture implicit correlations between bytes and labels, enabling the construction of powerful packet representations. The generator produces adversarial examples that are fed to the classifier to enhance its robustness. Experimental results demonstrate that RBLJAN achieves over 99% average F1-score on real-world legitimate traffic datasets and achieves 97.86% average F1-score on malware identification. Moreover, RBLJAN exhibits superior performance in terms of detection speed and robustness compared to state-of-the-art methods in real-world scenarios.

Index Terms—Traffic classification, deep learning, joint attention, adversarial learning

1 Introduction

Traffic classification (TC) identifies network traffic of different types during transmission, which is a vital process for network management and anomaly detection. However, nowadays, the influencing factors of Internet applications and behaviors are complex, dynamic, and interrelated. At the same time, Internet data is huge-amount, encrypted, and high-noise, making it challenging to extract useful information effectively [1]. Consequently, efficient and robust TC technologies of encrypted traffic are of great significance in the current network environment.

The wide adoption of encryption techniques and the emergence of increasingly complex network applications put forward new requirements for TC approaches [2]. Traditional methods based on port numbers or packet pattern matching [3] have become obsolete. Substantial progress has been achieved since the introduction of Machine Learning (ML) into TC tasks [4]. However, ML-based methods are highly dependent on manually crafted features, which are time-consuming and error-prone [5]. With the proliferation of Internet applications and the emergence of complex encryption technologies, feature mining becomes increasingly challenging, leading to a gradual decline in the performance of ML-based models. With the rise of Deep Neural Networks (DNN) in various recognition or classification tasks, Deep Learning (DL) based methods become popular in various TC tasks [5]. DNN can directly take the original data as input and process in an end-to-end way. They usually treat packets or flows as gray-scale images [6] or text [7] and adopt neural networks, e.g., Convolutional Neural networks (CNN) [2] or Recurrent Neural Networks (RNN) [7] to learn abstract features of network traffic, which makes it applicable to encrypted traffic identification. However, it is unreasonable to make an analogy between network traffic and images or text, because packets is specific sequences of bytes without a strong spatial representation like images or a large vocabulary like text since there are only 256 different bytes [8]. In addition, network traffic is affected by the dynamic network environment, causing data imbalance, heavy noise, and diversification. Besides, Lightweight DNN learns less knowledge and has lower performance, while complex DNN is deployed with high hardware requirements and time-consuming. In addition, recent studies show that DNN is susceptible to various attacks (e.g., malicious manipulation [9]), resulting in suboptimal performance. Therefore, in the current complex network environment, the design of a DL-based TC algorithm with high robustness, accuracy, and efficiency is an urgent problem in the current complex network environment [10].

- *Xi Xiao is with the Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong 518055, China, and also with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China. E-mail: xiaox@sz.tsinghua.edu.cn.*
- *Shuo Wang and Shutao Xia are with the Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, Guangdong 518055, China. Email: {wangs22@mails, xiast@sz}.tsinghua.edu.cn*
- *Qing Li and Zhang Bin are with the Peng Cheng Laboratory, Shenzhen, Guangdong 518055, China. E-mail: {liq, bin.zhang}@pcl.ac.cn.*
- *Guangwu Hu is with the School of Computer Science, Shenzhen Institute of Information Technology, Shenzhen, Guangdong 518172, China. E-mail: hugw@sziit.edu.cn.*
- *Kelong Mao is with the Gaoling School of Artificial Intelligence, Renmin University, Beijing 100086, China. E-mail: mkl@ruc.edu.cn.*
- *Xiapu Luo is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China. E-mail: csxluo@comp.polyu.edu.hk.*

Manuscript received X XX. 202X; revised X XX. 202X; accepted X XX. 202X. Date of publication X XX. 202X; date of current version X XX. 202X. (Corresponding author: Qing Li.)
Digital Object Identifier no. XX.XXXX/TDSC.202X.XXXXXXX

In this paper, we propose Robust Byte-Label Joint Attention Network (RBLJAN) for efficient network traffic classification. Specifically, at the packet level, RBLJAN consists of two components, i.e., a classifier and an adversarial traffic generator. The classifier first divides a packet into the header part and the payload part. For each part, the bytes and all candidate labels are embedded into a joint space for the calculation of a byte-label similarity matrix, which is then fed to four meticulously designed attention encoders to learn the attention scores of each byte and label. In this way, we obtain four packet representations by calculating the weighted sum of the embedded vectors based on the attention scores. Finally, these representations are splices and connected to each label for classification. At the flow level, RBLJAN performs flow classification by adjusting the model structures. The learning and classification process at the flow level follows a similar procedure to that of packet-level classification. To enhance the robustness of RBLJAN, motivated by Generative Adversarial Networks (GAN) [11], an adversarial traffic generator is introduced. It takes random vectors and original packets as input and generates adversarial packets on which the classifier could make errors. In the training process, the classifier receives input from both the original packets and the generator, which is trained to accurately determine the labels of these packets. This adversarial training approach strengthens the robustness of RBLJAN, enabling it to handle noise and attacks effectively.

We conduct extensive experiments on four benchmark TC tasks at the packet level and evaluate RBLJAN on malware identification at the flow level. Besides, we evaluate the robustness of the baselines and RBLJAN by conducting experiments on datasets with random noises. Results show that our method could handle various network TC scenarios and achieve high performance.

The contributions of this paper (which is an extended version of our previous conference paper [8]) can be concluded as follows:

- We design a novel deep learning network, the Robust Byte-Label Joint Attention Network (RBLJAN), for traffic classification. RBLJAN has the superiority of effectiveness, detection speed, and robustness, and works well at both packet and flow levels.
- RBLJAN leverages the improved joint attention encoders, header-payload parallel processing, and the improved GAN mechanism to improve its performance. RBLJAN is designed according to the characteristics of network traffic and is more interpretable.
- We implemented multiple baseline methods for the performance evaluation. In addition to the early approaches [12] [7] [13], some SOTA methods [14] [15] [16] [17] [18] are also used as the baselines. Furthermore, we apply RBLJAN in four classification scenarios to prove its feasibility and universality.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 provides the details of RBLJAN. Then, we present the experimental results in Section 4. Finally, we conclude the paper of our work in Section 5. Also, Table 1 summarizes the acronyms used in the text for readability.

TABLE 1
List of the acronyms used in the manuscript.

Acronym	Definition
TC	Traffic Classification
ML / DL	Machine Learning / Deep Learning
DNN	Deep Neural Networks
MLP	Multilayer Perceptron
CNN	Convolutional Neural networks
1D-CNN	one-dimensional CNN
RNN	Recurrent Neural Networks
ResNet	Residual Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
Tree-RNN	Tree Structural RNN
GAN	Generative Adversarial Networks
RBLJAN	Robust Byte-Label Joint Attention Network
DPI	Deep Packet Inspection
LDA	Latent Dirichlet Allocation
SAE	Stacked Auto Encoder
TTL	Time-to-Live
SC	Similarity Calculation
CDN	Content Distribution Networks
C / G	Classifier / Generator
FLOPS	Floating Point Operations Per Second

2 Related Work

Typical works on TC involve many different techniques. In this section, we first show the summary of previous works in Table 2, then introduce them in three main categories: port-based and packet inspection-based methods, traditional machine learning-based methods, and deep learning-based methods.

2.1 Port-based and Packet Inspection-based Methods

At an early age, the methods on TC are large to perform the port matching. Obviously, using the individual port features is simple and fast, however, due to the prevalence of dynamic port and port camouflage techniques, port-based classification methods can only achieve low accuracy and are no longer suitable for the current network environment. The next generation of TC is known as Deep Packet Inspection (DPI) methods. DPI focuses on mining the patterns or keywords manually in the packet payload. Libprotoident [3], nDPI [19] and BlindBox [20] are popular DPI techniques, which leverage predefined patterns to distinguish protocols from others. However, this means they need to update patterns whenever a new protocol is released, which is time-consuming. Moreover, with the wide use of more complex encryption and anonymous techniques, extracting protocol patterns is more difficult and prone to be invalidated, thus making DPI lose effectiveness and encounter great challenges.

2.2 Traditional Machine Learning-based Methods

In recent academic research, significant attention has been paid to traditional ML-based techniques to solve TC problems. ML-based methods typically requires manual feature extraction and leverage ML-based classifiers to determine traffic classes.

Traffic side-channel features generally refer to the specific fields in packets or flows, such as protocol type,

TABLE 2
Summary of previous works. ∗, ○ and ● indicate that this reference processes header and payload separately, uses multimodal learning and considers model's robustness, respectively.

Method Based	Input Features	References	Technique	Merits and Defects
Port/DPI	port/packet byte	[3] [19]∗ [20]	pattern matching	fast, but unable to handle encrypted traffic, hard to extract and update patterns
Machine Learning	bit/byte stream	[21] [12] [22] [23] [24]●	semi-supervised learning, topic models, clustering, ML-based classifiers, ensemble model	hard to extract distinguished traffic features, ML-based models' ability to express complex problems is limited
	statistical features	[25] [26]∗ [27] [28]○●		
	side-channel features	[29] [30] [31] [24]●		
Deep Learning	pseudo-image	[32] [2] [6] [33] [11]○● [34]	CNN, GAN	an end-to-end way without manual feature extraction, but treating traffic as images or text is unreasonable, lack of interpretability and robustness consideration
	bit/byte stream	[35] [36]○ [37]○ [38]○ [39]∗ [40] [13] [14] [7] [16] [41]∗○ [15] [17]○ [42]○ [8] [43]○ [44] [45] [46] [47] [48] [49]○	CNN, RNN, MLP, GAN, GNN, ResNet, contrastive/incremental/ multimodal learning	
	statistical features	[50]○● [51]○ [52]○ [53]∗ [18]∗●		
	side-channel features	[40] [54]○ [47]○		
	bit/byte stream	**RBLJAN(This paper)**∗○●	**CNN with joint attention, GAN**	**accurate, robust, fast and interpretable**

TCP header length and payload length. Based on side-channel features extracted from packets or flows, existing works take these features as inputs and train various ML-based models for TC, including K-Means, Gaussian Mixture Model [29], random forest [30], Markov models [31] and clustering algorithms [24]. Besides, some studies perform statistical calculations on these side-channel features and obtain traffic statistical features. These statistics typically perform on spercific fields in traffic flows, including count, mean, extremum, standard deviation, etc. Researchers leverage statistical features to feed ML-based classifiers, such as SVM [25], random forest [27], semi-supervised models [26] and ensemble models [28]. In addition to using these features, some works directly feed the raw bit/byte stream of traffic into ML-based models. They convert traffic into byte sequences and learn traffic characteristics by appling ML-based methods, such as Bayes, semi-supervised learning [21], Latent Dirichlet Allocation (LDA) [12], topic models [22] and clustering techniques [23].

Note that with the large emergence of more complex new network applications, it is difficult and time-consuming to extract distinguished features from network traffic. In addition, ML-based models are generally shallow structured algorithms. When the number of training samples is limited, their ability to express complex problems is limited, and their generalization ability is also constrained [55].

2.3 Deep Learning-based Methods

Deep learning has achieved considerable progress in many fields related to classification problems, such as computer vision and natural language processing. Since network traffic can be considered as images or texts, the field of TC can leverage DL-based techniques to improve its performance. DL-based methods do not require manually extracting features and can directly utilize the raw byte sequence of the network traffic in an end-to-end way.

Initially, researchers made an analogy between network traffic and images and applied CNN for malware TC [32] [2] [6] [34]. As network traffic can be viewed as a sequence of bits or bytes, a large number of studies process traffic as bit/byte sequences and apply DL-based models for TC. Typical deep learning models include SAE [13] [38], CNN [14] [39] [33] [44], RNN [16] [16] [35] [40] [15] and GNN [53] [18]. In addition, some works combine different models to perform network traffic classification [36] [37] [38] [41] [17] [54] [42] [43] [50] [51] [52] [47] [49]. They leveraged various networks to learn potential features of network traffic from multiple perspectives (i.e., multimodal learning) and achieved significant results. Moreover, with the rapid development of DNN, attention network [56] [44] [8] [45] [49], contrastive learning [45] [18], incremental learning [46] [48] are also applied in network TC tasks.

It is true that DL-based methods have the advantage of automatic feature extraction and are occupying the TC field [10]. However, researchers demonstrated that DNN is difficult to interpret [57] and is highly vulnerable to small perturbations on the input data [58]. Recently, in [59], a set of adversarial network traffic examples such as Start_RandPad and End_AdvPad are proposed and utilized to evaluate the robustness of network traffic classifiers. Based on GAN, Liu et al. proposed TrafficGAN [11] by transforming traffic into images to identify malware traffic. In addition, Li et al. [51] developed their model based on feature extraction and GFDA-WGAN, which detects unbalanced attack traffic and traffic with noise. However, state-of-the-art methods pay little attention to the interpretability and rationality of their methods that enhance effectiveness and robustness [60]. To cope with these challenges, we design an improved generative adversarial network with byte-label joint attention learning for efficient and robust traffic classification.

3 Methodology

This section describes the architecture of our proposed method. We first show the model overview in Section 3.1, then discuss the data preprocessing phase in Section 3.2. Section 3.3-3.5 gives the whole framework of RBLJAN at the packet-level, including the classifier, the adversarial traffic generator, and the training phase. Moreover, we extend our model to the flow-level classification in Section 3.6.

3.1 Model Overview

As mentioned, RBLJAN aims to classify network traffic into specific classes (e.g., application, website) according to user's requirements. As shown in Figure 1, RBLJAN contains a classifier and a generator, which are trained through a generative adversarial mode. The classifier first divides a packet into the header part and the payload part. For each part, it embeds the bytes and all candidate labels into a joint space. Then it uses attention encoders that are meticulously designed to learn attention scores for each byte and each label. Finally, it obtains a packet representation vector, which is put into a linear space to obtain the prediction of each label. The generator takes the preprocessed packets and random vectors as input. Then through linear connections and reshaping operations, it feeds the inner generator network to obtain adversarial bytes, which are inserted into the original packets to form the adversarial packets.

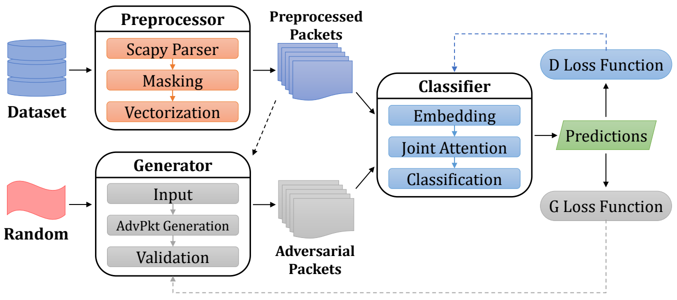

Fig. 1. The overall structure of RBLJAN, which includes three modules, i.e., the preprocessor, the generator, and the classifier. The entire model is trained through a generative adversarial mode.

3.2 Data Preprocessing

The raw network traffic is composed of network packets and is generally stored in the format of *.pcap* or *.pcapng* files. Specifically, a packet is generally encapsulated by the Ethernet II header, the IP header, the TCP/UDP header, and the payload. We leverage *Scapy*[1] to transform the raw traffic into the proper format that is suitable to our model. The preprocessing includes the masking phase and the vectorization phase.

3.2.1 Masking

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 the input of our model.

1) *Ethernet II header*: This header contains *EtherType*, *Source* and *Destination MAC Addresses*, which are useless for the classification task, so we remove this part.

2) *IP header*: Some fields in this header play an important role in TC, i.e., *Total Length*, *Protocol*, *Time-to-Live* (*TTL*), while some of them do not provide useful information, such as *IP addresses*, *Checksum* and *Identification*. Therefore, we discard these fields and mask them with zeros.

3) *TCP/UDP header*: We set the random and useless port numbers to zero and keep some *Well-known Ports*[2]. Notice that although TCP/UDP ports are slightly related to the local network configuration and may lead to overfitting, some well-known ports also provide useful information for TC tasks. For example, the standard HTTP port is 80, and the default port of SSL is 443.

4) *Payload*: Even if the payload is encrypted in a pseudo-random-like format [8], its inner relationship between bytes may still exist, especially the similarity between bytes and labels. Since the payload is essential for TC tasks, for legitimate traffic TC tasks, only packets with payload are kept for experiments. However, in the case of malware TC tasks, we keep these packets without a payload. This is because several types of common network attacks, such as Port Scan and DDoS, exploit packets without payload.

1. https://scapy.net/
2. https://www.iana.org/

3.2.2 Vectorization

After finishing the masking phase, we get a binary packet sequence. To facilitate the calculation, we treat the traffic as bytes and convert them to integers from 0 to 255 (8 bits). As the packet header is largely composed of side-channel features of the packet, the payload is usually user data that is generally encrypted [16]. They provide different information for classification tasks, so we divide the packet into the header part and the payload part, which are two vectors consisting of integers from 0 to 255.

In addition, as our model requires that the input data must be uniform in length, we counted the distribution of the length of all packets in the X-APP dataset [16]. We found that the length of most packets are less than 1500 bytes and the header size is below 50 bytes, so we unify the length of the header and the payload to 50 bytes and 1450 bytes by truncating the byte sequence or padding with 256 (a number unrelated to other bytes), respectively.

Finally, the packet is processed into a byte sequence vector and is defined as:

$$(H, P) = (\{b_i\}_{i=1}^{M}, \{b_i\}_{i=M+1}^{N}), \tag{1}$$

where b_i refers to the i-th byte of the packet, M and $N-M$ represent the size of the header part H and payload part P respectively.

3.3 Packet-level Classifier Design

The overall structure of RBLJAN for the packet-level classification is illustrated in Figure 2. RBLJAN can capture the implicit and complex relations between the packet byte sequence and candidate labels due to its effective header-payload parallel processing and joint attention mechanism. On the whole, our classifier is composed of three modules, i.e., the embedding module, the joint attention module, and the classification module. In this subsection, we introduce the details of these three modules.

3.3.1 Embedding Module

The input of our classifier contains three components, i.e., the header $H = \{b_i\}_{i=1}^{M}$, the payload $P = \{b_i\}_{i=M+1}^{N}$, and T candidate labels $L = \{l_i\}_{i=1}^{T}$. We first embed each byte b_i and each label l_i into two joint embedding spaces, i.e., the Header-Label joint space and the Payload-Label joint space. Specifically, each joint space contains two parallel layers, i.e., the byte embedding layer and the label embedding layer. The former maps byte b_i to byte embedding vector $B_i^{'}$, and the latter maps label l_i to label embedding vector L_i respectively, where $B_i^{'}, L_i \in \mathbb{R}^E$. We use $B^{'}$, L to denote $\left(B_1^{'}, \ldots, B_S^{'}\right)$ and $(L_1, \ldots, L_T)$ respectively, where $S = M \ or \ N - M$. Notice that we use different maps for header embedding and payload embedding, in order to explain our model more clearly, we simply use $B^{'}$ to represent the byte part.

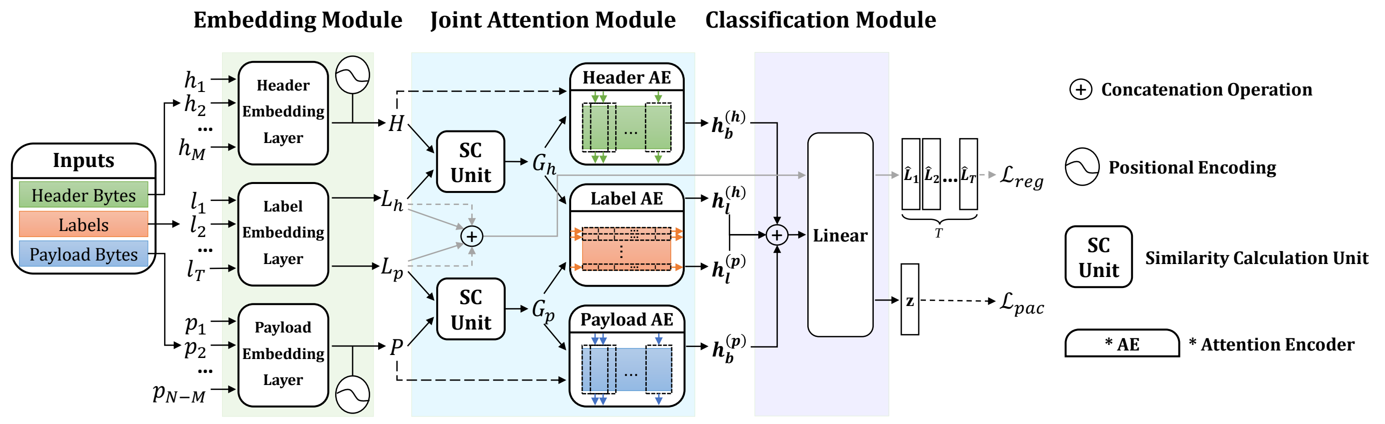

Fig. 2. The Classifier in RBLJAN, classifies network traffic in the form of byte sequences into specific categories.

Besides, the bytes or bits in different positions may have different meanings and importance. For instance, if the *Version* field in the IP header is 6, it indicates it is an IPv6 packet. While a *Protocol* number of 6 means the communication protocol is TCP. Therefore, we introduce the position encoding mentioned in [61] to make use of the absolute positional information. Define $Q_i \in \mathbb{R}^E$ as the position embedding for the i-th byte, then we update the byte embedding by adding $B_i^{'}$ and Q_i. In the following, we use B_i to denote the byte embedding in the i-th position.

3.3.2 Joint Attention Module

This module is the core of RBLJAN, which contains three components, i.e., (1) Similarity Calculation (SC) unit, (2) byte attention encoder, and (3) label attention encoder. It learns the packet representations from byte vectors B_i and label vectors L_i.

1) Similarity Calculation Unit: It is to calculate the cosine similarity between the i-th byte and the j-th label by:

$$G_{(i,\ j)} = \frac{B_i \cdot L_j}{\|B_i\| \cdot \|L_j\|},\ i \in [1, S], j \in [1, T], \tag{2}$$

where $G = [G_{(i,\ j)}]$ is the byte-label similarity matrix, which will be used by the attention encoders to generate their attention scores respectively.

2) Byte Attention Encoder: It uses 1D-CNN to learn attention scores of each byte from G and gets a representation vector of the byte part as a weighted sum of the bytes.

Notice that the byte attention encoder includes the header attention encoder and the payload attention encoder, both of which have the same structure with different parameters. Therefore, we use the byte attention encoder to represent the two encoders for convenience. Specifically, since a byte in a packet is usually not independent in semantics, we consider a sequence of n consecutive bytes as a unit rather than using a single byte. Thus, an S-byte sequence will have $S-n+1$-byte units. The byte unit vector is obtained by taking an average of all its byte vectors. We employ K kernels convolution with a non-linear activation function to act on the $n \times T$ region of G to produce K attention scores as candidates for each byte unit. For each byte unit i, the j-th candidate attention score is obtained from the following formula:

$$\begin{aligned}\widehat{\alpha}_{ij} = \sigma\left(\text{BN}\left(\text{Conv}_j\left(G_{(i:i+n-1,1:T)}\right)\right)\right),\\ i \in [1, S-n+1],\ j \in [1, K],\end{aligned} \tag{3}$$

where σ is the non-linear activation function ReLU, BN is the batch normalization function to normalize all candidate scores from a batch, Conv_j denotes the convolution operation with the j-th kernel. Then we use max-pooling to choose the largest candidate of each byte unit:

$$\begin{aligned}\widehat{\alpha}_i = \text{MaxPooling}\left([\widehat{\alpha}_{i1}, \ldots, \widehat{\alpha}_{iK}]\right),\\ i \in [1, S-n+1].\end{aligned} \tag{4}$$

Finally, by using the weight normalization function softmax [8], we obtain the byte attention vector $\alpha = [\alpha_1, \ldots, \alpha_{S-n+1}]$. We sum up all byte unit vectors weighted by the byte attention vector to obtain the byte part of the packet representation, which consists of the header representation and the payload representation:

$$\begin{aligned}h_b^{(h)} &= \sum_{i=1}^{M-n^{(h)}+1} {\alpha_i}^{(h)} \left(\tfrac{1}{n^{(h)}} \sum_{j=i}^{i+n^{(h)}-1} H_j\right),\\ h_b^{(p)} &= \sum_{i=1}^{N-M-n^{(p)}+1} {\alpha_i}^{(p)} \left(\tfrac{1}{n^{(p)}} \sum_{j=i}^{i+n^{(p)}-1} P_j\right),\end{aligned} \tag{5}$$

where $(*)^{(h)}$ and $(*)^{(p)}$ represent the corresponding parameters of the header and the payload respectively.

3) Label Attention Encoder: This encoder employs a similar calculation method as the byte attention encoder, which learns the label attention vector from G and generates the label part of the packet representation as the weighted sum of the label vectors.

Notice that the input of each attention encoder is a matrix (i.e., G) with S rows and T columns. For the byte attention encoder whose input channel is the number of candidate labels T (e.g., 20), which is smaller. While for the label attention encoder, its input size increased to S (e.g., 1460 bytes of payload), which is so large that simply applying network units to it may result in the loss of local information. Therefore, we divide the input size of G into multiple segments and learn the label attention separately, and then combine the results to obtain the label part of the packet representation.

Specifically, we divide the S channels into s groups (so $S^{'} = S/s$). We first adopt a fully connected layer to act on each group to obtain s attention scores for each label. Then another fully connected layer is used to connect these groups to their corresponding label. Therefore, we get a T-dimension attention vector β by employing the softmax on the output of the two-stage full connection. Finally, The label part of the packet representation is the weighted sum of all label vectors, which also includes the header representation and the payload representation. It is given by the following two formulas:

$$\begin{aligned} h_l^{(h)} &= \sum_{i=1}^{T} \beta_i^{(h)} L_i^{(h)}, \\ h_l^{(p)} &= \sum_{i=1}^{T} \beta_i^{(p)} L_i^{(p)}. \end{aligned} \tag{6}$$

3.3.3 Classification Module

First, we concatenate the four packet representations learned by the attention encoders to form the whole packet representation:

$$h = \left[h_b^{(h)} || h_b^{(p)} || h_l^{(h)} || h_l^{(p)}\right], \tag{7}$$

where $||$ is the concatenation operation and $h \in \mathbb{R}^{4E}$. Then we employ a linear layer to transform the packet representation into a T-dimension space for classification:

$$z = \text{softmax}\left(Wh + q\right), \tag{8}$$

where $W \in \mathbb{R}^{T \times 4E}$ and $q \in \mathbb{R}^{T}$ are the weight and bias respectively. We adopt *Focal Loss* [62] for model training because it focuses learning on hard and misclassified samples and is superior on the imbalanced multi-classification problem. For a T-class classification, it is formulated as:

$$\begin{aligned} \mathcal{L}_{\text{pac}} &= FL(y, z; \alpha, \gamma) \\ &= -\alpha \left[1 - \text{softmax}\left(z_y\right)\right]^{\gamma} \log\left(\text{softmax}\left(z_y\right)\right), \end{aligned} \tag{9}$$

where y is the label representing the ground truth, z_y denotes the prediction probability of the label y, α and γ are the class weight and the focusing parameter of focal loss. We set α as one minus the percentage of the class and $\gamma = 2$ based on the discussion in [16].

In addition, as RBLJAN classifies traffic based on the similarity between the byte embedding and the label embedding, we hope that the learned packet representation should be more similar to its ground truth label embedding. In other words, the label embedding itself should be correctly classified through the classification module. Therefore, we replaced h in the Formula (8) with $\widehat{L}_t = [L_t^{(h)} || L_t^{(p)} || L_t^{(h)} || L_t^{(p)}]$, where $t \in [1, T]$ and $\widehat{L}_t$ denotes the t-th label representation. Then we add another label regularization loss formulated as:

$$\mathcal{L}_{\text{reg}} = \sum_{t=1}^{T} \text{CEL}\left(\text{softmax}\left(W\widehat{L}_t + q\right), \widehat{y}_t\right), \tag{10}$$

where $\widehat{y}_t$ is a one-hot vector that denotes the t-th class label. CEL denotes the *Cross Entropy Loss*. We use *Cross Entropy Loss* but not *Focal Loss* because each class only has one label vector, which is balanced. Finally, the loss of RBLJAN is the weighted sum of the above two losses:

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{pac}} + \lambda \mathcal{L}_{\text{reg}}, \tag{11}$$

where λ is a hyper-parameter to adjust the proportion of label regularization loss. In this way, RBLJAN not only learns powerful packet representations but also tries to learn discriminative label embedding for each class, ensuring that the label attention mechanism is an effective enhancement for TC tasks.

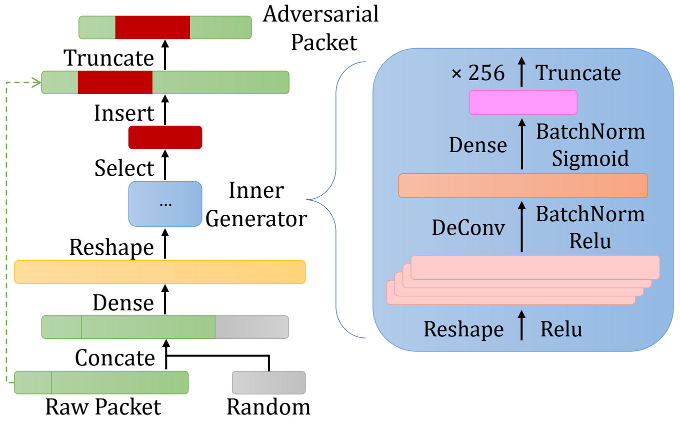

Fig. 3. The overall architecture of the adversarial traffic generator in RBLJAN at the packet level.

3.4 Adversarial Traffic Generator Design

The architecture of the adversarial traffic generator at the packet level is shown in Figure 3. Random vectors and preprocessed packets are used as the input of the model to generate adversarial packets corresponding to the original labels. The inner generator network is based on deconvolution and the output of the generator is a sequence of adversarial bytes, which will be inserted into the original packet to form an adversarial packet.

3.4.1 Inputs

The input of the generator is the original preprocessed packet and a certain length of random noise. In previous works, the input generally contains a label vector [11] indicating the label of adversarial data to be generated, but there are two shortcomings in using the label vector as the input:

1) It leads the data generated by the generator to a single distribution, which is easy to identify by the classifier and cannot improve the generalization of the classifier.

2) It is difficult for the generator to learn the distribution characteristics of the original data, and the adversarial data generated are difficult to attack the classifier. When the classifier is trained to be relatively strong, it is difficult for the generator to make further progress.

Therefore, we take the original packet directly as the generator input and add random noise to expand the generated packet distribution. In this way, the generator can not only learn the data distribution characteristics of the original packet, but also generate a variety of adversarial bytes and optimize the generation of adversarial bytes continuously.

3.4.2 Inner Generator Network

The inner generator is based on deconvolution, which contains a reshape layer, a deconvolution layer, and a dense layer. Specifically, the preprocessed packets and random vectors are first concatenated into one sequence. Then it is transformed through the dense layer and reshaped into a matrix to be input into the deconvolution layer. Finally, through a dense layer, we obtain some adversarial bytes to be inserted into the original packet. We also employ batch normalization and ReLU between each layer to improve the training effect of the generator. Note that, unlike traditional works that directly generate the whole packet, our method does not destroy the structure and functionality of the packet, especially the packet header. It is more reasonable and applicable in the current network environment.

In order to construct adversarial packets, we define the original packet input by the model as $\mathcal{X} = (\mathcal{H}, \mathcal{P})$, and the adversarial bytes generated by G as $\widetilde{x}$. We first insert $\widetilde{x}$ between $\mathcal{H}$ and $\mathcal{P}$ to obtain $\mathcal{X} = (\mathcal{H}, \widetilde{x}, \mathcal{P})$ and then obtain adversarial packets by truncating $\mathcal{X}$ to a length of N. Moreover, in order to make the classifier able to deal with adversarial packets with different lengths and different data distributions, we introduce a selection mechanism similar to the Dropout mechanism. Specifically, we first select a random length of continuous bytes from the generated adversarial data and then determine whether to use the generated adversarial bytes by a random probability p. If used, the continuous bytes are inserted into the original packet, and if not, we randomly generate bytes of that length to insert. We define the function $\text{Random}_{\text{Domain}}(\widetilde{s})$ to generate a random array of length $\widetilde{s}$ in Domain. Then, the process of generating adversarial bytes $\widetilde{x}$ is:

$$\widetilde{x} = \begin{cases} \{\widetilde{x}_i\}_{i=1}^{\widetilde{s}}, & if\ p < 0.5 \\ \text{Random}_{[0,255] \bigcap \mathbb{N}}(\widetilde{s}), & if\ p \geq 0.5 \end{cases}, \tag{12}$$

where $\mathbb{N}$ denotes the natural number domain, $\text{len}(\widetilde{x})$ represents the length of the adversarial bytes $\widetilde{x}$. Define $\mathbb{R}$ as the real number domain, then $p = \text{Random}_{[0,1) \bigcap \mathbb{R}}(1)$ and $\widetilde{s} = \text{Random}_{[1, len(\widetilde{x})) \bigcap \mathbb{N}}(1)$.

3.4.3 Validation

The outputs of the generator are adversarial examples of network traffic. As mentioned above, these adversarial examples should make the classifier predict errors. In other words, when a generated packet is input into the classifier but the prediction is not the ground truth, then the generator achieves its purpose. The loss function of the generator is also Focal loss, which can be described as:

$$\begin{aligned} \mathcal{L}_{\text{adv}} &= \widehat{\text{FL}}(y, z; \alpha, \gamma) \\ &= -\alpha \left[\text{softmax}(z_y)\right]^{\gamma} \log\left(1 - \text{softmax}(z_y)\right), \end{aligned} \tag{13}$$

comparing the above formula with Formula (9), we can see that the prediction probability of the loss function utilization of generators is one minus the probability of the classifier loss function. In this way, the generator will be trained in the opposite direction of the classifier, and this adversarial learning makes our classification model increasingly robust.

3.5 Training Method

As shown in Figure 1, RBLJAN leverages a training method similar to GAN. Overall, the input of RBLJAN training includes three parts, namely, packet byte sequence, all candidate labels, and a certain length of random vectors. Firstly, the packet byte sequence and all candidate labels are sent to the classifier for training and the prediction probability of each label is obtained. Then, the loss function of the classifier is used to reverse propagation and update its parameters. At the same time, the packet byte sequence and random vectors are sent to the adversarial traffic generator to obtain adversarial traffic examples. The prediction results of the classifier and the loss function of the generator are used to reverse propagation and update the parameters of the generator. Since the generator is a simpler network, in one epoch, we train the generator several times for each batch until it converges or reaches a maximum number of iterations, while the classifier updates the parameters only once. Then, the adversarial traffic examples are sent to the classifier for additional training. In this way, the classifier could accurately classify the original packets as well as the adversarial packets into the correct label.

Algorithm 1: The workflow of the Training method of RBLJAN

```
Input: Training dataset $\mathcal{D}$, all candidate labels $\widehat{y}$,
       Batch size $B$, Embedding dimension $E$,
       Focal Loss parameters $\gamma,\ \alpha$, Classifier
       parameters $\theta_c$, Generator parameters $\theta_g$,
       Random size $r$, Learning rate $\eta_c$ and $\eta_g$
Output: Learned model parameters $\theta_c$
1  Initial model parameters: $\theta_c \leftarrow \theta_c^0,\ \theta_g \leftarrow \theta_g^0$
2  Initial step counter: $i \leftarrow 1$
3  while $\theta_c^i$ not converged do
4      Getting data: randomly select training batch $X$
         and its ground truth labels $y$ from $\mathcal{D}$
5      Preprocessing: $\mathcal{X}\leftarrow$clean, mask, zero-pad, unify
         length
6      Initial step counter of G: $k \leftarrow 1$
7      while $\theta_g^k$ not converged do
8          $\mathcal{A} \leftarrow \text{Random}_{[0,1)\bigcap\mathbb{R}}(r)$
9          $G$ Forward propagation: $\widetilde{X} \leftarrow G\left(\mathcal{A}, \mathcal{X}; \theta_g^k\right)$
10         $C$ Forward propagation: $\widetilde{z}, \widetilde{L} \leftarrow C\left(\widetilde{X}; \theta_c^i\right)$
11         $G$ Backward propagation: $\theta_g^{k+1} \leftarrow$
               $\theta_g^k - \eta_g \cdot \text{Adam}_g\left(\nabla_{\theta_g}\widehat{\text{FL}}(y, \widetilde{z}; \alpha, \gamma)\right)$
12         $k \leftarrow k + 1$
13     end
14     $C$ Forward propagation: $\widetilde{z}_1, \widetilde{L}_1 \leftarrow D\left(\mathcal{X}; \theta_c^i\right)$
15     $\mathcal{A} \leftarrow \text{Random}_{[0,1)\bigcap\mathbb{R}}(r)$
16     $G$ Forward propagation: $\widetilde{X} \leftarrow G\left(\mathcal{A}, \mathcal{X}; \theta_g^k\right)$
17     $C$ Forward propagation: $\widetilde{z}_2, \widetilde{L}_2 \leftarrow D\left(\widetilde{X}; \theta_c^i\right)$
18     $C$ Backward propagation: $\theta_c^{i+1} \leftarrow$
           $\theta_c^i - \eta_c \cdot \text{Adam}_c(\nabla_{\theta_c}\text{FL}([\widetilde{z}_1, \widetilde{z}_2], [y, y]; \alpha, \gamma) +$
                       $\lambda\nabla_{\theta_c}\text{CEL}([\widetilde{L}_1, \widetilde{L}_2], [\widehat{y}, \widehat{y}]))$
19     $i \leftarrow i + 1$
20 end
```

Suppose the input of a neural network model is A and the parameter of the model is θ, then the classifier (C) and the generator (G) can be expressed as follows:

$$\begin{aligned} \widetilde{z}, \widetilde{L} &= C\left(A_c; \theta_c\right), \\ \widetilde{X} &= G\left(A_g; \theta_g\right), \end{aligned} \tag{14}$$

where $\widetilde{z}$ is the probability that the classifier predicts each label, $\widetilde{L}$ denotes the T label representations, and $\widetilde{X}$ is the generated adversarial examples. The workflow of the training phase of RBLJAN is shown in Algorithm 1.

3.6 Extend to flow-level classification

The initial objective of RBLJAN is to identify the single packet and perform the packet-level classification, it also can be used to solve traffic flow classification problems as traffic flow is just formed by several continuous packets. Ensemble and voting are common ways to extend classification from the packet level to the flow level. However, they usually first get the output of each packet and then aggregate these outputs to get the label of the flow, which ignores the flow-level features. Instead, we introduce a novel extension approach to decide the class of the flow that can directly apply the model to the flow-level classification. Some studies [29] [18] indicate that the first U packets of the flow involve the important information about the flow. Thus, we use SplitCap[3] to obtain bidirectional flows and select the first U packets to represent the whole flow.

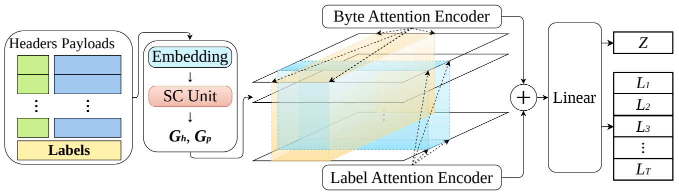

Fig. 4. The architecture of the classifier in RBLJAN at the flow level.

Fig 4 shows the overall structure of the classifier in RBLJAN for the flow-level classification. Similar to the packet-level classifier, the model receives a flow F with U packet sequences and all candidate labels as input and process the header sequences and the payload sequences in parallel. In the following, we use byte sequences to describe both the header and payload for convenience. The embedding module first embeds each byte and each label into a joint space, where we obtain $B \in \mathbb{R}^{U\times S\times E}$ and $L \in \mathbb{R}^{T\times E}$. Then through the SC Unit, we can obtain a 3-dimensional byte-label similarity matrix $\left[G_{(i,\ j,\ k)}\right] \in \mathbb{R}^{U\times S\times T}$. To enable RBLJAN to handle data with an additional dimension, we extend the dimension of the learning unit (e.g., the convolutional kernel, the fully connected layer) in RBLJAN. Specifically, for the byte attention, we extend the byte unit to a 3D-matrix like $n \times T \times U$, then we employ $\widetilde{K}$ kernels convolution with a non-linear activation function to act on $U \times n \times T$ region of $\left[G_{(i,\ j,\ k)}\right]$ to produce $\widetilde{K}$ attention scores as candidates for each byte unit. Then we use max-pooling and softmax functions to obtain the byte attention vector. As for the label attention, we use a three-stage full connection to get label attention scores, in which an added layer is to connect different packets in the flow. As for the generator part, it generates adversarial bytes for all packets in the flow and inserts them into the packets separately. In this way, the generator produces adversarial flows and sends to the flow-level classifier to improve its robustness.

RBLJAN-Flow can directly use the flows as the input of the model, instead of training the model of the packet-level classification in advance. It can learn the correlations between packets (i.e., the flow-level features) and within packets (i.e., the packet-level features). Thus, RBLJAN-Flow leverages more comprehensive information from both the packet-level and the flow-level features, which also performs well in real-world scenarios.

3. https://www.netresec.com/?page=SplitCap

4 Experiments

To evaluate the performance of RBLJAN, in this section, we conduct experiments on four large real-world datasets with four benchmark tasks at the packet-level classification, i.e., application classification, website fingerprinting, malware identification and traffic characterization, and we perform extensive experiments for malware identification at the flow level. We provide a detailed analysis of the robustness evaluation on baselines and RBLJAN. Furthermore, we conduct some ablation studies and show the interpretability of the learned embedding and attention scores of RBLJAN. In the following, we present details of datasets, experimental setup, comparison results and analysis.

4.1 Dataset and Implementation

We evaluate RBLJAN on four real-world datasets, including X-APP [16], X-WEB [16], USTC-TFC [32] and ISCX-VPN [63]. X-APP and X-WEB are two large real-world datasets containing both encrypted and unencrypted traffic packets for application classification and website fingerprinting respectively. Specifically, X-APP contains 29 kinds of popular applications and X-WEB contains 20 popular websites. ISCX-VPN and USTC-TFC are two widely used public encrypted traffic datasets for TC. The former consists of VPN and non-VPN traffic of multiple applications under several behaviors and is frequently updated. According to the 2022 version of this dataset, we divide it into 12 types (6 types of VPN, 6 types of non-VPN) to validate the effectiveness of RBLJAN on traffic characterization. The latter includes 10 types of malware traffic and 10 types of benign traffic. We select the 10 types of malware traffic in this dataset to verify the ability to detect malware traffic of RBLJAN. In general, these four traffic datasets cover various areas including email, music, video, security, chat, search, shopping, etc, which is rich enough for evaluating RBLJAN.

At the packet level, we compare RBLJAN with eight SOTA TC methods: Securitas [12], DeepPacket [13], SAM [14], Tree-RNN [15] and EBSNN [16], ICLSTM [17], CLE-TFE [18] and our previous work BLJAN [8], which are described in Section 2. For RBLJAN, in most cases, we set the embedding dimension E of bytes and labels to 256, set the byte unit lengths of headers and payloads, i.e., $n^{(h)}$ and $n^{(p)}$, to 17 and 65, respectively, set the number of convolution kernels K to 8 on both the header part and the payload part, and set λ to 0.1. We train RBLJAN with Adam Optimizer with a learning rate of 0.001 and batch size of 256, and we employ a dropout of 0.5 on the classification layer to prevent overfitting. An early stopping strategy is performed, i.e., premature stopping if the average F1-score on the validation data does not increase for 5 successive epochs.

Each method is evaluated with a 10-fold cross-validation. We randomly divide the dataset into ten parts and take turns selecting 8, 1, and 1 of them for training, validation and testing. Except for Securitas, due to the large memory consumption of LDA and the limitation of our hardware conditions, we randomly select 4k packets as the positive class and 4k packets as the negative class. The average of the results of 10 experiments is used as an estimate of the performance of each method. All the experiments are conducted on a server with 128 GB memory, Intel Xeon Silver 4210 CPU, and NVIDIA 2080Ti GPU. We carefully tune the parameters of compared baselines and RBLJAN for a fair comparison. To evaluate these models, common metrics including Precision, Recall, F1-score and accuracy are adopted for every target category. When evaluating the robustness of these models, in order to ensure the consistency of the input data, we use the method of inserting random noise to verify all the classification algorithms, instead of using adversarial examples that may cause unfairness during evaluation. We test the accuracy of all methods by inserting random noise of different lengths between the header and the payload, thus evaluating the robustness based on changes in accuracy.

4.2 Performance Comparison at Packet-level

In this subsection, we give a detailed analysis of all baselines and RBLJAN on four datasets at the packet level.

4.2.1 Application Classification

We evaluate the performances of RBLJAN and compared baselines on the application classification on the X-APP dataset. The F1-scores are shown in Table 3. SVM, C4.5 and Bayes refer to the Securitas with SVM, C4.5 Decision Tree and Bayes Network, respectively. DP-SAE and DP-CNN stand for DeepPacket with the stacked autoencoder and the one-dimension CNN, respectively. EBSNN-LSTM and EBSNN-GRU represent EBSNN with the LSTM unit and GRU unit, respectively. For EBSNN, because we use similar data processing methods and the same dataset, our reproduced results are slightly worse than those in their paper, so we directly refer to the experimental results in [16]. Through the results, we draw the following conclusions.

Overall, RBLJAN achieves the best performance on F1-score on average. It is notable that RBLJAN achieves the best F1-score on the detection of 22 out of 29 applications, achieving more than 99.90% on 24 classes. Such good results strongly demonstrate the effectiveness of RBLJAN on application TC tasks. EBSNN, BLJAN, ICLSTM and Tree-CNN also achieve good results in application TC, whose averaged F1-score is slightly lower than RBLJAN. DP-CNN, C4.5 reached F1-scores of 96.5% and 94.2%, which can also deal with the application TC in the current network environment. The performance of DP-SAE is the worst, which only got 74.7%. The performance of SVM and Bayes of Securitas is slightly worse than that of C4.5. SAM only uses the header information, and the average F1-score is only 91.2%. Although CLE-TFE uses powerful GNNs, it needs to use the information of flows when training packet classification. Due to the imbalance of the number of flows of different classes in this dataset, its averaged F1 score is only 90%. Generally, the performance of the ML-based methods is not as high as that of the neural network. SAM only uses header information and ignores the features in the payload, which leads to lower performance. In 29 kinds of applications, the performances of Gmail, Aimchat, and QQ[4] are lower than other categories in all models. These real-time communication applications are very similar in packet features (e.g., *TTL* and *Encryption Type*), which makes all methods have sub-optimal performance in these applications.

In fact, the byte distributions of the header and payload of the packet are totally different. The header part contains different feature fields (such as *Time to Live*, *Protocol*, etc.). RBLJAN can find the discriminative feature fields in the header and give greater weights to provide information for classification. In addition, for the encrypted byte sequences in the payload, the potential features of different labels of sequences still exist, which can be learned through the byte-label similarity matrix. RBLJAN makes the similarity between the bytes of each category and the corresponding labels reach the maximum similarity, so as to provide the label information as much as possible. In this way, RBLJAN integrated the side-channel features and sequence similarity features of packets, which can learn more useful information about packets to obtain better performance.

4.2.2 Website Fingerprinting

In addition to the application TC task, we implement website fingerprinting on the X-WEB dataset. Websites utilize a large number of public static resources stored in the Content Distribution Network (CDN), which makes it contain more network noise than network applications. At the same time, some websites contain various links to other websites, which makes the traffic dataset itself contain more category-cross noise that is hard to identify. Therefore, through website fingerprinting, we can better evaluate the ability of classification models to process noisy data.

Table 4 lists the experimental results of each method on the X-WEB dataset. We can see that RBLJAN outperforms the SOTA methods. In 20 kinds of websites, 13 kinds of RBLJAN get the highest F1-score. Compared with BLJAN, RBLJAN has improved the Performance in all categories. EBSNN-LSTM also performs quite well and gets 11 kinds of the highest F1-score, its average F1-score is slightly worse than RBLJAN. ICLSTM, Tree-RNN and EBSNN-GRU achieve sub-optimal performances and reach over 98% average F1-score. But they are based on multi-layer RNNs, whose detection speed is much slower than RBLJAN (refer to Subsection 4.5). CLE-TFE performs better in this task than application identification because it can better handle the noise in website traffic by learning from flow-level features. For DeepPacket, although it uses 1D-CNN or linear connection layers as an important part of RBLJAN, it only achieves 90.04% and 64.39% average F1-score mainly because it regards packets to images and ignores the differences and potential relationships between bytes. The input of SAM is just the first 50 bytes of the packet and it does not leverage the potential features in the payload, which is inferior to RBLJAN by 5.46%. C4.5 of Securitas is better than SVM and Bayes, but compared with recent DL-based models, its performance is no longer able to deal with the task of current website fingerprinting. It is worth mentioning that Taobao and Youku belong to the core business of Alibaba

4. QQ is a popular instant messaging application in China that provides services such as chat, music, online games, etc

TABLE 3
Performance comparison on X-APP w.r.t. F1-score. AVE denotes the average results of all applications.

Model	Vimeo	Spotify	Voipbuster	Sinauc	CloudMusic	Weibo	Baidu	Tudou	Amazon	Thunder	Gmail	PPlive	QQ	Taobao	YahooMail
C4.5	0.9568	0.9435	0.9887	0.9950	0.9439	0.8631	0.8405	0.9789	0.8936	0.9357	0.9466	0.9504	0.8943	0.8571	0.9517
SVM	0.9249	0.8476	0.9937	0.9926	0.9393	0.8379	0.7620	0.9730	0.6978	0.9109	0.9311	0.9120	0.7334	0.7356	0.8878
Bayes	0.8996	0.7965	0.9925	0.9863	0.8743	0.7663	0.7190	0.9368	0.6944	0.8479	0.9015	0.8910	0.7139	0.3206	0.8117
DP-SAE	0.9343	0.7662	0.9920	0.9845	0.8173	0.7233	0.6041	0.8282	0.2926	0.7817	0.2719	0.9485	0.4748	0.7072	0.8920
DP-CNN	0.9781	0.9305	0.9984	0.9995	0.9856	0.9656	0.9255	0.9890	0.8986	0.9845	0.8876	0.9880	0.9366	0.9293	0.9906
SAM	0.9841	0.9452	0.9971	0.9995	0.8456	0.7939	0.5847	0.9617	0.7253	0.9809	0.8356	0.9891	0.8625	0.9286	0.9588
Tree-RNN	0.9967	0.9869	0.9986	0.9998	0.9953	0.9997	0.9990	0.9964	0.9757	0.9962	0.9262	0.9924	0.9562	**1.0000**	0.9841
ICLSTM	0.9873	0.9705	0.9995	0.9999	0.9983	0.9928	0.9857	0.9979	0.9976	0.9981	0.9519	0.9972	0.9721	0.9987	0.9980
CLE-TFE	0.9783	0.8919	**1.0000**	0.9996	0.9581	0.7873	0.6912	0.9345	0.6772	0.9725	0.7561	0.9954	0.8778	0.8303	0.8894
EBSNN-LSTM	0.9988	0.9925	0.9995	**1.0000**	**0.9998**	**0.9999**	0.9992	**0.9997**	**1.0000**	0.9995	0.9923	**0.9999**	**0.9979**	0.9997	**1.0000**
EBSNN-GRU	0.9963	0.9773	0.9989	0.9999	0.9990	0.9981	0.9929	0.9989	0.9997	0.9981	0.9701	0.9985	0.9907	0.9993	0.9993
BLJAN	0.9963	0.9844	0.9997	**1.0000**	0.9972	0.9966	0.9933	0.9985	0.9945	0.9960	0.9784	0.9979	0.9764	0.9921	0.9970
RBLJAN	**0.9997**	**0.9995**	0.9999	**1.0000**	**0.9998**	**0.9999**	**1.0000**	0.9995	**1.0000**	**0.9997**	**0.9958**	**0.9999**	**0.9979**	0.9998	**1.0000**

Model	iTunes	Twitter	JD	Sohu	Youtube	Youku	Netflix	Aimchat	Kugou	Skype	Facebook	Google	MS-SQL	MS-Exchange	AVE
C4.5	0.8761	0.9666	0.8812	0.9335	0.9799	0.9406	0.9925	0.9425	0.9735	0.9338	0.9913	0.9711	0.9950	0.9927	0.9417
SVM	0.7867	0.9721	0.7883	0.9198	0.9619	0.9554	0.9578	0.9223	0.9849	0.9179	0.9937	0.9815	0.9975	0.9854	0.9036
Bayes	0.6348	0.9721	0.7839	0.8951	0.9165	0.9479	0.9566	0.8582	0.9677	0.8579	0.9778	0.9815	0.9937	0.9780	0.8577
DP-SAE	0.4525	0.7328	0.2654	0.8507	0.9354	0.8605	0.9721	0.6128	0.9761	0.9637	0.9836	0.4626	0.9520	0.6114	0.7466
DP-CNN	0.9323	0.9811	0.9000	0.9777	0.9800	0.9868	0.9940	0.9252	0.9962	0.9974	0.9995	0.9731	0.9957	0.9585	0.9650
SAM	0.6009	0.9979	0.8785	0.9739	0.9719	0.9564	0.9970	0.8756	0.9951	0.9922	0.9981	0.9896	0.9955	0.8577	0.9129
Tree-RNN	0.9945	0.9982	0.9997	0.9905	0.9963	0.9980	0.9990	0.9582	0.9303	0.9780	0.9983	0.9970	0.9992	0.9762	0.9868
ICLSTM	0.9825	0.9837	0.9961	0.9945	0.9898	0.9954	0.9963	0.9690	0.9981	0.9990	0.9998	0.9788	0.9998	0.9855	0.9901
CLE-TFE	0.7576	**1.0000**	0.7431	0.9277	0.9632	0.9504	0.9903	0.7677	0.9980	0.9297	0.9997	**1.0000**	0.9992	0.8542	0.9007
EBSNN-LSTM	**0.9997**	0.9997	0.9995	**0.9992**	0.9979	**0.9997**	0.9992	0.9954	**0.9999**	0.9996	**1.0000**	0.9989	**1.0000**	**0.9985**	0.9988
EBSNN-GRU	0.9944	0.9992	0.9987	0.9942	0.9950	0.9986	0.9985	0.9783	0.9996	0.9991	0.9999	0.9989	0.9993	0.9971	0.9954
BLJAN	0.9891	0.9995	0.9927	0.9893	0.9933	0.9975	0.9985	0.9822	0.9992	0.9995	0.9999	0.9993	0.9993	0.9912	0.9941
RBLJAN	**0.9997**	**1.0000**	**0.9997**	0.9984	**0.9996**	0.9995	**0.9998**	**0.9955**	0.9997	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9971	**0.9993**

TABLE 4
Performance comparison on X-WEB w.r.t. F1-score. AVE denotes the average results of all websites.

Model	Amazon	Baidu	Bing	Douban	Facebook	Google	IMDb	Instagram	iQIYI	JD
C4.5	0.8844	0.9561	0.9641	0.9265	0.8872	0.9212	0.9357	0.9075	0.9264	0.9524
SVM	0.8086	0.9479	0.9637	0.8652	0.8138	0.8886	0.8179	0.8479	0.8506	0.9362
Bayes	0.7851	0.9165	0.9462	0.8488	0.7643	0.8702	0.7909	0.8413	0.8377	0.8727
DP-SAE	0.4502	0.9163	0.8372	0.8027	0.5991	0.3095	0.8962	0.3326	0.7682	0.8599
DP-CNN	0.9283	0.9520	0.8176	0.9556	0.6225	0.8774	0.9380	0.9327	0.9734	0.9789
SAM	0.9999	0.9797	0.9700	0.9284	0.9996	0.9995	0.9979	0.9998	0.8793	0.9778
Tree-RNN	0.9977	0.9983	0.9990	0.9983	0.9972	0.9980	0.9996	0.9948	0.9937	0.9997
ICLSTM	0.9965	0.9856	0.9938	0.9927	0.9974	0.9854	0.9975	0.9930	0.9920	0.9868
CLE-TFE	0.9988	0.9891	0.9788	0.9732	0.9903	0.9933	0.9916	0.9957	0.9726	0.9718
EBSNN-LSTM	**1.0000**	**1.0000**	0.9971	**1.0000**	0.9934	**1.0000**	0.9990	0.9990	**0.9998**	**0.9998**
EBSNN-GRU	0.9998	0.9998	0.9901	**1.0000**	0.9786	0.9999	0.9973	0.9961	0.9994	0.9994
BLJAN	0.9998	0.9885	0.9946	0.9958	0.9997	0.9994	0.9977	0.9996	0.9883	0.9772
RBLJAN	**1.0000**	0.9968	**0.9993**	0.9989	**0.9999**	0.9998	**0.9998**	**0.9998**	0.9982	**0.9998**

Model	CloudMusic	QQMail	Reddit	SinaWeibo	Taobao	TED	Tieba	Twitter	Youku	Youtube	AVE
C4.5	0.9225	0.8619	0.9350	0.9306	0.9627	0.9260	0.9378	0.9384	0.9240	0.9327	0.9267
SVM	0.8530	0.8000	0.8376	0.8578	0.9325	0.8229	0.8229	0.9125	0.8673	0.8609	0.8654
Bayes	0.7626	0.7263	0.7991	0.8472	0.9339	0.7850	0.8428	0.8872	0.7934	0.8360	0.8344
DP-SAE	0.6370	0.3878	0.3832	0.6366	0.5954	0.8155	0.7852	0.5615	0.5864	0.7182	0.6439
DP-CNN	0.9055	0.9416	0.9773	0.9597	0.8142	0.9567	0.8205	0.8116	0.8987	0.9467	0.9004
SAM	0.9059	0.7699	0.9995	0.9004	0.9954	0.9145	0.9995	0.9362	0.7555	**1.0000**	0.9454
Tree-RNN	0.9885	0.9971	0.9213	0.9964	0.9891	0.9957	0.9999	0.9951	0.9797	0.9346	0.9802
ICLSTM	0.9820	0.9377	0.9976	0.9771	0.9981	0.9702	0.9975	0.9898	0.9724	0.9992	0.9871
CLE-TFE	0.8872	0.8310	0.9949	0.9612	0.9946	0.9464	0.9954	0.9751	0.9199	0.9923	0.9677
EBSNN-LSTM	**0.9994**	**1.0000**	**1.0000**	**0.9984**	0.9958	0.9991	**1.0000**	0.9974	**0.9969**	0.9993	0.9987
EBSNN-GRU	0.9981	**1.0000**	0.9999	0.9954	0.9879	0.9960	0.9996	0.9915	0.9918	0.9973	0.9959
BLJAN	0.9821	0.9300	0.9998	0.9789	0.9973	0.9792	0.9999	0.9894	0.9689	0.9996	0.9883
RBLJAN	0.9971	0.9999	**1.0000**	0.9970	**0.9968**	**0.9998**	**1.0000**	**0.9989**	**0.9969**	**1.0000**	**0.9990**

Group[5], and their website traffic contains a large number of similar packets (for example, both contain DNS packets for *alibaba* domain name resolution), which are difficult to identify even through manual detection. Therefore, the fingerprinting results of these two websites in all methods are not very good. In total, RBLJAN achieves very high performance and reaches 99.90% average F1-score, verifying its strong ability and universality for website fingerprinting.

4.2.3 Traffic Characterization

In addition to the fine-level classification of network applications or website contents, we also use the ISCX-VPN dataset to evaluate the ability of traffic characterization of all methods. The experimental results are shown in Table 5.

The performance of all methods is similar to the above tasks and RBLJAN performs the best on average. Compared with the above two tasks, SAM obtains obvious improvement in F1-score. This is mainly because, the *Protocol* and other feature fields in the packet header may provide more useful information for classification. SAM only uses some fields of the header to classify, so it has high practical application value in traffic characterization. Moreover, C4.5 reaches 94.10% average F1-score and its performance approaches or even exceeds some DL-based methods, but note that Securitas is a binary TC method while the other methods are capable of multi-classification directly. Thus, in the real scenario that requires multi-classification, if we have K classes, we have to build K models and adopt an ensemble method, e.g., Bagging, Boosting, and Stacking [64], to integrate their results to get the final predicted class. So when Securitas is extended to multi-classification which is more complex than the current binary classification, its performance may decrease.

RBLJAN not only focuses on learning features from the header part of the packet but also learns sequence features

5. Taobao is a shopping website and Youku is an online video platform in China. Both of them operate as subsidiaries of Alibaba Group Holding Limited.

TABLE 5
Performance comparison on ISCX-VPN w.r.t. F1-score. AVE denotes the average results of all types of traffic.

Model	audio	chat	file	mail	streaming	voip	vpn-audio	vpn-chat	vpn-file	vpn-mail	vpn-streaming	vpn-voip	AVE
C4.5	0.9346	0.8854	0.9630	0.9156	0.9520	0.9438	0.9301	0.9006	0.9650	0.9614	0.9862	0.9545	0.9410
SVM	0.8575	0.7906	0.8777	0.8734	0.9538	0.8973	0.6777	0.8571	0.9367	0.8831	0.9417	0.9557	0.8752
Bayes	0.8344	0.6759	0.736	0.7845	0.8873	0.8973	0.6212	0.7637	0.8526	0.8229	0.9307	0.9487	0.8129
DP-SAE	0.9635	0.7099	0.9655	0.6074	0.9797	0.9611	0.9690	0.7322	0.9536	0.6860	0.9944	0.9647	0.8739
DP-CNN	0.9820	0.8361	0.9859	0.7316	0.9909	0.9509	0.9832	0.8965	0.9873	0.9287	0.9972	0.9559	0.9355
SAM	0.9947	0.8746	0.9993	0.7198	0.9968	0.9897	0.9926	0.9762	0.9994	0.9849	0.9992	0.9918	0.9599
Tree-RNN	0.9759	0.8700	0.9579	0.7441	0.9989	0.9964	0.9936	0.9721	0.9888	0.9820	0.9996	0.9941	0.9561
ICLSTM	0.9939	0.9516	0.9966	0.8980	0.9979	0.9987	0.9953	0.9817	0.9977	0.9892	0.9987	0.9990	0.9832
CLE-TFE	**0.9992**	0.9093	**0.9997**	0.8309	0.9981	0.9981	0.9952	0.9661	0.9992	0.9792	0.9998	0.9992	0.9728
EBSNN-LSTM	0.9976	0.9619	0.9975	0.9201	**0.9996**	0.9974	0.9982	**0.9913**	0.9988	**0.9981**	0.9997	0.9979	0.9882
EBSNN-GRU	0.9840	0.8833	0.9969	0.8759	0.9984	0.9943	0.9964	0.9798	0.9965	0.9911	0.9987	0.9989	0.9745
BLJAN	0.9964	0.9618	0.9978	0.9286	0.9992	0.9993	0.9979	0.9842	0.9982	0.9903	0.9992	0.9996	0.9877
RBLJAN	0.9980	**0.9893**	0.9980	**0.9799**	**0.9996**	**0.9996**	**0.9984**	0.9897	**0.9995**	0.9977	**0.9999**	**0.9997**	**0.9958**

TABLE 6
Performance comparison on USTC-TFC w.r.t. F1-score. AVE denotes the average results of all malware.

Model	Cridex	Geodo	Htbot	Miuref	Neris	Nsis-ay	Shifu	Tinba	Virut	Zeus	AVE
C4.5	0.9222	0.8165	0.7004	0.7640	0.8136	0.9158	0.9397	0.9937	0.7016	0.9188	0.8486
SVM	0.8193	0.7511	0.1943	0.6003	0.4792	0.8252	0.9412	0.9614	0.4718	0.6973	0.6741
Bayes	0.5185	0.7188	0.0769	0.4753	0.1085	0.7290	0.9118	0.9614	0.0387	0.5016	0.5041
DP-SAE	0.8087	0.6653	0.5287	0.4579	0.6809	0.8269	0.9149	0.9711	0.5671	0.5431	0.6965
DP-CNN	0.8621	0.7884	0.6590	0.5348	0.8359	0.9346	0.9384	0.9944	0.7632	0.8553	0.8166
SAM	0.9021	0.7710	0.7986	0.6756	0.8843	0.9685	0.9524	0.9946	0.8360	0.9714	0.8755
Tree-RNN	0.9311	0.9094	0.9474	0.7353	0.9714	0.9968	0.9697	**0.9995**	0.9693	0.9934	0.9423
ICLSTM	0.9601	0.9499	0.9817	0.8502	0.9649	0.9953	0.9877	0.9988	0.9626	0.9852	0.9636
CLE-TFE	**0.9951**	0.9477	0.9742	**0.9536**	0.7689	0.9774	0.9877	0.9991	0.7495	0.9913	0.9345
EBSNN-LSTM	0.9172	0.8945	0.9375	0.8485	0.9805	0.9964	0.9673	0.9993	0.9781	0.9906	0.9510
EBSNN-GRU	0.9333	0.8285	0.8621	0.7941	0.9577	0.9537	0.9671	0.9937	0.9248	0.9333	0.9148
BLJAN	0.9651	0.9544	0.9761	0.8142	0.9530	0.9959	0.9921	0.9988	0.9523	0.9725	0.9574
RBLJAN	0.9679	**0.9703**	**0.9925**	0.8895	**0.9895**	**0.9987**	**0.9962**	0.9975	**0.9887**	**0.9956**	**0.9786**

from the payload part. Compared with other methods (only using the header, or processing the header and the payload as a whole), RBLJAN is more reasonable and efficient in traffic characterization. In addition, as for chat and mail, these two types of traffic have the highest similarity in behavior and traffic features [8]. Compared with other categories, RBLJAN is about 1~2% lower in the performance of these two categories. However, RBLJAN has achieved the highest results on the three metrics of these two categories and outperforms other SOTA methods, which shows that RBLJAN can focus on learning more difficult categories to achieve better performance.

4.2.4 Malware Identification

With increasing attention to network security, the identification of network malware is also a challenging task in the field of network traffic. We select 10 types of malware traffic in the USTC-TFC dataset for the malware identification experimental task. Results of the F1-score of the malwares are shown in Table 6. Note that different from the above legitimate TC tasks that discard packets without payload, we keep these packets in the case of malware identification for the reasons described in Section 3.2.1.

It is notable that RBLJAN outperforms the other SOTA methods on all classes in terms of the F1-score, which demonstrates RBLJAN has a strong ability to identify malware traffic. Among the baselines, the average F1-scores of DeepPacket and ICLSTM have been improved to some degree. This dataset contains 10 traffic classes, which itself reduces the training difficulty of the multi-classification model. For Tree-RNN, it has always been in sub-optimal performance (about 94-98%), but note that in addition to the time overhead brought by its RNN-based network, its tree-like structure requires training multiple models and it requires traversing from the root node to the leaf node to complete the multi-classification. In this 10-class classification task, 7 models need to be trained and at least 3 times of predictions are required to obtain the results. The training difficulty and prediction time of Tree-RNN will continue to increase with the increase of categories.

TABLE 7
Performance of RBLJAN-Flow on USTC-TFC. P., R., and F1. are the abbreviations for Precision, Recall, and F1-score, respectively.

Malware	**P.**	**R.**	**F1.**	**Malware**	**P.**	**R.**	**F1.**
Cridex	0.998	1.000	0.999	Neris	0.977	0.973	0.975
Geodo	0.996	0.995	0.995	Shifu	0.999	0.998	0.998
Htbot	0.996	0.991	0.993	Tinba	1.000	1.000	1.000
Miuref	0.992	0.999	0.995	Virut	0.955	0.972	0.964
Nsis-ay	0.997	0.978	0.988	Zeus	0.995	0.999	0.997
				Average	0.990	0.991	0.990

4.3 Performance of extending RBLJAN to Flow-level

To validate the ability of RBLJAN to identify network bidirectional flows, we select the USTC-TFC Dataset to do flow-level experiments. Most of the malware contains more than 5k bidirectional flows in the dataset and the number of flows is balanced. We also made statistics on the flows of the other three datasets, but the number of their flows is very unbalanced, even though some categories contain only dozens of flows, which is inappropriate for training. USTC-TFC consists of 10 types of encrypted flow traffic and consists of both the connection-oriented (TCP) flows and the connection-less (UDP) flows, which is enough to validate the ability of RBLJAN to identify traffic flows. We set $U = 15$ and the flow-level experimental results are listed in Table 7.

Results show that all the malware except Neris, 97.7%, and Virut, 95.5%, obtain F1-score higher than 99.0%. First, USTC-TFC is a malware traffic dataset, which has more noise traffic and is difficult to classify. We carefully analyzed the composition of this dataset, which contains lots of broadcast traffic and LAN address-related protocols. For Neris and Virut, we traced the source to find the wrong classified data flows and found that most of the wrong classified flows are UDP flows. Besides, Note that for the F1-socre

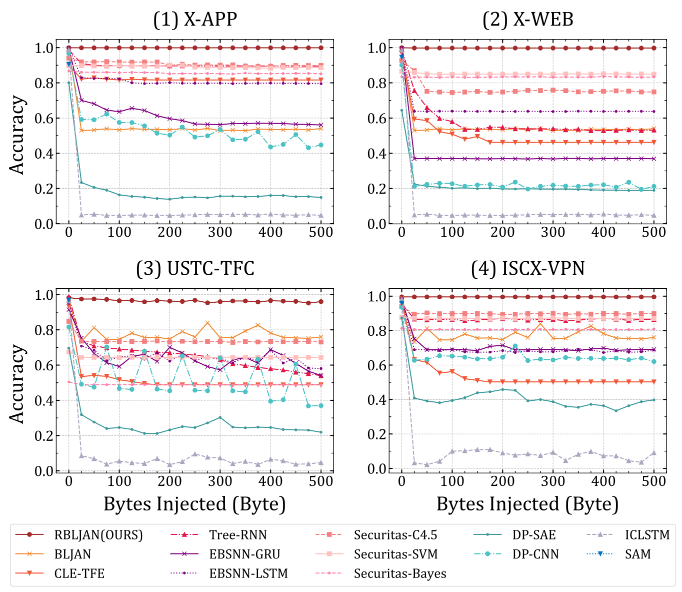

Fig. 5. Robustness comparison on four datasets.

of miuref that is only 88% at the packet level but reaches 99% at the flow level. We found it contains a large number of packets without payload, making it difficult to classify at the packet level. However, at the flow level, RBLJAN can learn flow-level features to improve its accuracy. In particular, the traffic of some LAN address communication protocols, such as MDNS and SSDP, has a small number of packets in the flow with a short packet length. In addition, the payload of these packets contains almost no effective information, which is difficult to classify even by manual methods. But RBLJAN-Flow can still predict accurately in high noise network traffic, which validates the extensibility and effectiveness of the flow-level classification.

4.4 Robustness Comparison at Packet-level

This subsection describes the robustness comparison between baselines and RBLJAN, we first show an overall comparison of robustness on four datasets, then we take USTC-TFC as an example and discuss the robustness of each model on specific categories. Moreover, we investigate the impacts of the inner generator structure to get better performance of RBLJAN.

4.4.1 Overall Comparison of Four Datasets

According to the robustness evaluation method of the classification models in Section 4.1, we randomly insert random bytes of different lengths into the packet. we test the length of inserted bytes in $[0,\ 25,\ 50, \ldots, 500]$, and show the accuracy of each method on the four datasets in Figure 5. Note that for RBLJAN, the model we used in this section is the same as the one in Section 4.2, i.e., both contain the generator and the improved GAN mechanism.

Results show that the accuracy of RBLJAN on the four datasets is stable at more than 95% due to its noise processing mechanism and adversarial learning mechanism. On the contrary, the accuracy of other models significantly reduces after adding random noise due to over-reliance on the single input of the original data. Specifically, when the first 25 bytes are inserted, their accuracy has dropped significantly to around the fixed value. DNN-based algorithms are most affected by random noise, while ML-based methods Securitas are not. Specifically, ML-based Securitas utilizes the topic model of LDA, which leverages the statistical distribution of byte frequency for feature extraction. The inserted random noise conforms to a uniform distribution and has little effect on the statistical characteristics of bytes, so Securitas is less affected by the random noise. But note that the attacker can obtain the features extracted by Securitas, that is, which bytes have a greater impact on the classification results. If these bytes related to classification are inserted, then the byte distribution will change and the accuracy will reduce significantly. DNN-based methods take the original bytes as input and classify them through the parameters of the hidden layer of the neural network. If the relative position or value of the original data is slightly changed, the output will be greatly changed. ICLSTM is most affected with an accuracy close to zero, which means it has almost no robustness. It converts packets into gray-scale images and uses DNN models for classification, resulting in higher sensitivity to noisy data. DeepPacket uses a stacked CNN or linear connection network, which is also heavily affected by random bytes. The DP-SAE method has only about 20% accuracy in application and malware classification, and the performance of DP-CNN is also reduced to a maximum of 60% accuracy. They are no longer able to make a satisfactory prediction of network traffic. Tree-RNN, EBSNN and CLE-TFE are less affected by random noise than DeepPacket, but note that these methods have achieved an accuracy of nearly 99% without any noise, after inserting random noise, the highest accuracy in website fingerprinting is only 65%, and the lowest is below 40%. BLJAN neither separately processes the header and payload, nor considers model robustness. Its accuracy rates are all below 80%, indicating that RBLJAN greatly improves the robustness of BLJAN.

Such results demonstrate that RBLJAN has the advantages of high accuracy and strong robustness. In fact, during the training process of RBLJAN, adversarial examples of different lengths and distributions are inserted, so that the classifier can learn the difference between the adversarial example sequence and the original packet byte sequence. For the inserted random bytes, RBLJAN will give them a smaller weight, so the adversarial packet and the original packet will finally have similar high-level representations. Therefore, RBLJAN can maintain high accuracy even after inserting noise of different lengths and distributions.

4.4.2 Categories Comparison on USTC-TFC

The previous section introduces the average accuracy of each method in four noisy datasets. In order to gain a deeper understanding of the classification behavior of each model on each category, we take the USTC-TFC dataset as an example and show the results of some classes in Figure 6.

From the results, random noise will make the classification model misclassify more or less for each class, among which the Nsis-ay category has the least impact, and the other classes have a greater impact. Although the previous analysis argues that inserting random noise has little effect on Securitas, take C4.5 as an example, the accuracy drops more significantly on the Shifu and Tinba. In addition, the

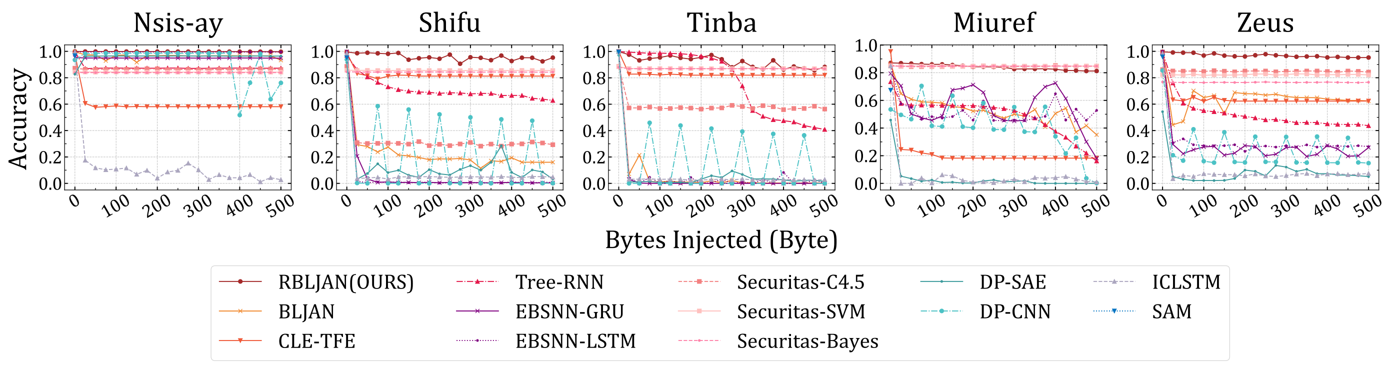

Fig. 6. Robustness comparison of some classes on USTC-TFC.

classification of different classes under random noise can correspond to the structural characteristics of the classification behavior of the model. For example, when DP-CNN is classifying Miuref, Shifu, Tinba and Zeus, the accuracy jumps to a higher value for every 75 bytes added, and the accuracy decreases sharply under other length noise. This can be interpreted that the model is a two-stage convolution process, the stride of the first layer convolution is 3 and the second layer is 5, so that the model learns 15 consecutive bytes of features when moving as a whole (the step size is 25 bytes in the figure and the least common multiple of 15 and 25 is 75). For EBSNN, on the Miuref and Zeus, the accuracy does not decrease continuously as the number of random noise increases but increases to higher accuracy for some specific number of bytes, which is also due to its two-stage network designation. Both the accuracy of CLE-TFE and BLJAN varies greatly across different categories, indicating that they heavily rely on the raw data and may have bias towards certain categories.

Overall, the accuracy of the comparison methods in noisy environments drops to unsatisfactory results, while RBLJAN maintains high accuracy in each class. Such results verify its practical value to cope with various network TC tasks in noisy environments.

4.4.3 Investigation of the inner generator structure

To further obtain better performances in our generator, we implemented three inner generators based on different neural network architectures (i.e., MLP, CNN, and RNN) to generate different adversarial examples and analyze their performances. Similar to the robustness testing method, the evaluation results on different neural network generators on ISCX-VPN dataset are shown in Figure 7(left).

In order to ensure fairness through experiments, the parameter quantity of each inner generator is controlled at the order of 10^6. We also add batch normalization operation and ReLU between each layer to improve the training effect. Specifically, for MLP, we employ three fully connected layers on the input sequence with a dropout of 0.05 after each linear layer. And for CNN, the generator is mainly based on deconvolution, which has already been introduced in section 3.4.2. And for RNN, we implement a sequence generation method based on LSTMCell [65], which generates a specific length (e.g., 25 bytes) of bytes in every timestamp and finally forms the whole adversarial byte sequence.

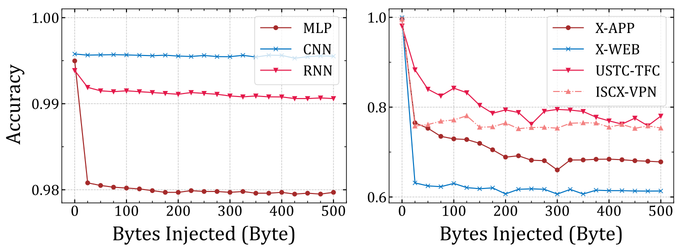

Fig. 7. (left) Robustness evaluation of RBLJAN on different neural network generators (Section 4.4.3); (right) robustness evaluation of RBLJAN-GAN on four datasets (Section 4.6.3).

From the results, when evaluating the original dataset, i.e., the inserted byte length is 0, CNN performs best, followed by MLP, and RNN performs worst. Notice that the accuracy of all three generators exceeds 99.4%, which indicates that our classification model itself has a strong classification ability. However, when inserting random bytes, the accuracy of MLP sharply decreases to around 98.0%, while the accuracy of RNN slightly decreases but remains above 99.0%. The robustness of CNN is the best, with almost no decrease in accuracy as the number of inserted adversarial bytes increases. In addition, there is no significant change in the accuracy of these three generators although different lengths of adversarial bytes were inserted. In general, the deconvolution-based generator has significantly improved the robustness of the classification model proposed in this paper, which may also be due to the fact that the classifier is based on CNN. Although generators with different structures perform differently, they can still perform well when inserting adversarial bytes, outperforming other baselines in Section 4.4, which also proves the effectiveness of the proposed GAN-based training framework in this paper.

4.5 Comparison of Testing Time

The classification speed is of great significance to the online classification of network traffic. Hence, we compare the classification speed of RBLJAN with those baselines. Since training is conducted only once and can be done offline, running time is measured by testing time. We set the batch size $B = 128$ and the comparison results are shown in Figure 8. On average, RBLJAN only needs 0.051ms to detect the class of one packet, which achieves around 4x, 10x, 50x and 100x speedups compared with EBSNN, Tree-RNN, Securitas

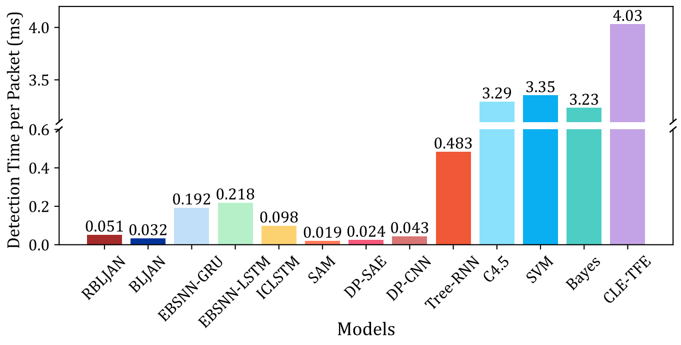

Fig. 8. Comparison of testing time per packet among 13 models.

TABLE 8
Numbers of operations and parameters and detection time under limited resources.

Model	FLOPS	Params	Server (ms/pkt)		Laptop (ms/pkt)	
			CPU	GPU	CPU	GPU
BLJAN	16.16 M	40.77 K	1.122	0.042	4.533	0.444
EBSNN-LSTM	502.4 M	433.1 K	9.324	0.218	16.98	5.914
DP-CNN	199.2 M	254.2 K	1.838	0.043	6.368	0.860
CLE-TFE	253.2 G	44.88 M	9.871	4.030	-	-
RBLJAN	16.04 M	78.92 K	1.490	0.051	4.839	0.473

and CLE-TFE, demonstrating the efficiency superiority of RBLJAN. SAM spends the shortest time to detect a single packet, reaching 0.019ms. But note that SAM only uses the packet header, while other methods use both the header and payload for classification. Regarding DeepPacket, they are slightly faster than RBLJAN due to the use of simpler neural networks, such as MLP in DP-SAE and two CNN layers in DP-CNN. Additionally, they do not utilize the embedding layer, which reduces many computations but loses effectiveness. The reasons why RBLJAN has a high testing speed are as follows. Firstly, The internal network of RBLJAN can be calculated in parallel and save much time cost. The four attention encoders in Formula (5) and Formula (6) are independent and can be calculated in parallel. Most of the other methods do not support parallel computation, for example, the two-stage RNN in EBSNN can only be calculated one by one, and the prediction of child nodes in Tree-RNN can only be executed after the output of the parent node. Secondly, compared with SAM and DeepPacket which need to stack several CNN or MLP layers, RBLJAN has an efficient network structure composed of one embedding layer, parallel joint-attention layers, and one MLP layer for classification. In the joint attention layer, the byte part has only one CNN layer and the label part has only two fully connected layers. Thirdly, compared with Securitas and CLE-TFE, RBLJAN is an end-to-end method and does not require feature extraction. Securitas needs to extract packet features through the LDA in advance and CLE-TFE spends much time in generating graphs, and then use classifiers to make predictions, so their testing time is much longer than other models.

In consideration of real-world scenarios, we also conduct experiments under limited resources. In addition to the device described in Section 4.1, we also choose a laptop with Intel i5-10210U CPU and NVIDIA MX250 GPU as the constrained testing platform. We choose four DL-based baselines and test RBLJAN and them in CPU and GPU environments on these two devices. The testing results and the analysis of the number of operations and parameters of these models are shown in Table 8. CLE-TFE has the highest Params (number of parameters) and FLOPS (Floating Point Operations Per Second) due to its complex GNN and multiple layers of neural networks. Its testing time on the server far exceeds that of other models and even cannot be deployed on laptops due to excessive memory usage. EBSNN-LSTM and DP-CNN contain stacked RNNs or CNNs, resulting in a larger number of parameters and operations than our method. Interestingly, DP-CNN has a bit less detection time on the server than RBLJAN but more on the laptop. This indicates that RBLJAN has good deployability and fast detection speed on restricted devices. Compared with our previous work BLJAN, RBLJAN doubles the Params but its FLOPS has not increased. RBLJAN greatly improves robustness and effectiveness while ensuring faster detection speed, which is a significant improvement. Due to the GPU's accelerated processing of neural network model operations, the running speed of each model has been improved to some degrees. RBLJAN is lightweight and efficient, which can be deployed on regular laptops and detect packets at a relatively fast speed (0.473 ms/pkt). Compared with other methods, RBLJAN can not only leverage GPU for accelerated processing, but its internal byte-label joint attention module can also be calculated in parallel, which greatly improves its detection speed.

In summary, the structure of RBLJAN is reasonably and meticulously designed, which makes it effective and fast and can handle multi-scenario TC tasks in the current network environment. Such results show the possibility that RBLJAN can be deployed in real networks to perform online classification of encrypted traffic.

4.6 Ablation Study

In this section, we perform ablation studies to verify the effectiveness of important modules, i.e., the label attention mechanism, the header-payload parallel processing mechanism, and the improved GAN mechanism in RBLJAN.

4.6.1 *Label Attention*

We discard the label parts $h_l^{(h)}$ and $h_l^{(p)}$ and only use the byte parts $h_b^{(h)}$ and $h_b^{(p)}$ as the packet representation. Certainly, $\mathcal{L}_{\text{reg}}$ is also removed. We name this simplified model RBLJAN-LA and test its average F1-score on four datasets. Results are shown in Table 9. It is clear that after removing the label part, the average F1-score of RBLJAN obviously drops (more than 4.5%) on all datasets, demonstrating that the label attention mechanism is an indispensable part of RBLJAN. In fact, the label attention part is the key for RBLJAN to handle encrypted traffic packets. Even if the packet payload is encrypted to be a pseudo-random-like format [13], its relationship with each label embedding may still exist, which can be learned by the label attention mechanism to enhance the final packet representation and help identify the packet class.

4.6.2 *Header-Payload Parallel Processing*

Most of the existing classification models take the entire sequence of the packet as the input of the neural network,

TABLE 9
Ablation study results on four datasets w.r.t. average F1-score.

Model	X-APP	X-WEB	USTC-TFC	ISCX-VPN
RBLJAN-LA	0.9537	0.9368	0.9524	0.9386
RBLJAN-HP	0.9822	0.9745	0.9867	0.9773
RBLJAN	**0.9993**	**0.9990**	**0.9980**	**0.9958**

ignoring the structural composition of the packet. In fact, the information contained in the header and the payload is completely different and the processing methods should also be different. To validate the effectiveness of our header-payload parallel processing, we take the entire packet as the input of RBLJAN and adjust Formula (7) as $h = [h_b||h_p]$.

We set the length of byte units $n = 50$ according to [8], name the new model as RBLJAN-HP, and show results in Table 9. When RBLJAN does not separate the header and payload parts, the average F1-score is only about 97%-98%, which verifies the advantage of header-payload parallel processing. In fact, the header part is generally the feature information in plaintext, and the payload part is generally user data, which can be encrypted or unencrypted. For encrypted traffic, the similarities between the bytes of the header and payload and the label are totally different. In this way, using different encoders to learn knowledge from the header and payload parts is more helpful and reasonable. Furthermore, by separating these two parts, the attention encoders in RBLJAN can be parallelly computed and save much time cost, which enhances its efficiency.

4.6.3 improved GAN

The reason for the high robustness of RBLJAN is that it incorporates an improved GAN mechanism. We remove the adversarial traffic generator, only use the classification model for training, and obtain the new model named RBLJAN-GAN. It is tested by inserting random noise of different lengths into the original data. The test results are shown in Figure 7(right). After the improved GAN mechanism is removed, the robustness of the RBLJAN-GAN is greatly reduced, and the accuracy of website recognition is reduced from 99% to about 40%. In fact, RBLJAN can learn the difference between the noise sequence and the original packet byte sequence and it can cope with random noise of different lengths, so the classification effect after inserting noise of different lengths remains basically unchanged. Therefore, our improved GAN mechanism can effectively improve the robustness of the classification model, which is a very important part of RBLJAN.

4.7 Visualization for Interpretability

In this part, take the packet-level classification as an example, we provide an intuitional illustration of why RBLJAN performs well for TC tasks by visualizing the header attention scores, the similarities between bytes and labels, and label attention scores learned by the joint attention module.

4.7.1 Header Attention Scores

For the header part of the packet, RBLJAN classifies according to the information of the specific field of the packet header. Therefore, the attention score of the header byte, i.e., the α, reflects the importance of every header field. After

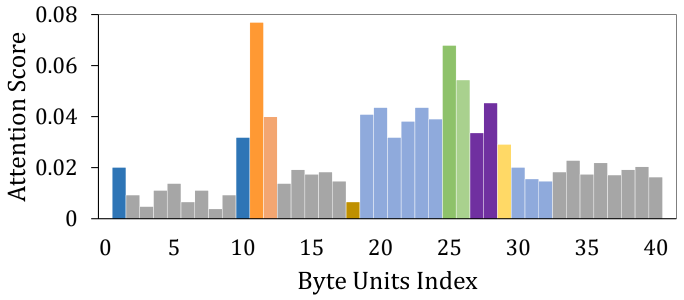

Fig. 9. The average byte attention scores of all packet headers at their corresponding positions.

preprocessing, we consider the following fields kept in the header section: *IP Header Length* (byte 1), *Total Length* (bytes 3~4), *TTL* (byte 9), *Protocol* (byte 10), *Ports* (bytes 21~24), *TCP Header Length* (byte 33), *TCP Flags* (byte 34), *Window Size* (bytes 35~36), and *Urgent Pointer* (bytes 39~40).

In order to get a better understanding of the classification behavior of RBLJAN, we set the length of the header byte unit as 17, and add padding with 8 zeros to both the beginning and end of the header byte sequence, ensuring that the length of the byte unit is equivalent to the size of the packet header. Then we average the byte attention scores of all packet headers at their corresponding positions and show the results in Figure 9. Therefore, the first column in the figure represents the weight of the first-byte unit, which includes 8-byte padding at the beginning and the first 9 bytes of the packet. Without considering padding, this byte unit contains the *IP Header Length*, *Total Length*, and *TTL*. The *Protocol* field is then covered into the following 2~9-byte units. They share the same byte unit vector but the weights vary due to the movement of the convolutional kernels. Note that their average weight is lower than most other byte units, which indicates their average embedding is not very important for classification. As the kernels move, the weight increases (columns 10~13) and decreases to a stationary stage (columns 13~17). This is because the *IP Header Length* and the *Total Length* are no longer covered by the kernels, and the trend indicates that the *Total Length* is given a large weight by the classifier. Besides, The 18th-byte unit discards *TTL* and only includes the *Protocol*. The weight reduction indicates that *TTL* is more important than the *Protocol* for the classification task. The weights of other byte units can also be analyzed similarly, and we can draw the following conclusions: *TCP Header Length* plays a significant role in classification (column 25), some *Well-known Ports* are still discriminate features (columns 19~24), and the low byte of some field (e.g., *Window Size* and *Ports*) is more important than the high byte (columns 27~32).

In total, results show that the weights of byte units vary depending on their position, and the larger weight of a certain byte unit is directly correlated to some more significant fields in the packet header. This demonstrates that RBLJAN can learn the most suitable weights and features from the packet header.

4.7.2 Similarities between Packets and Labels

RBLJAN embeds each byte and label into a joint space and endeavors to ensure that the byte representation, i.e., the h

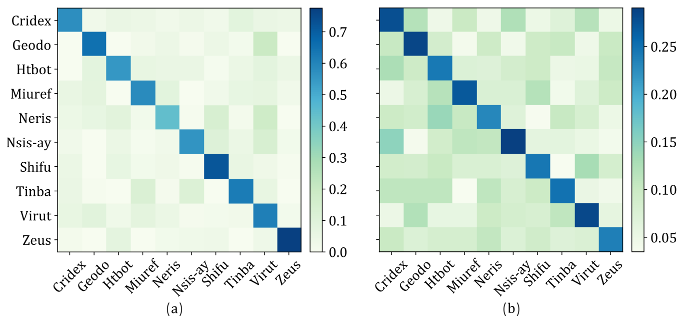

Fig. 10. (a) Visualization of similarities between *packet class representation* and *label representation*; (b) visualization of the average *label attention vectors*, each row represents an average *label attention vector* of that class.

in Formula (7), is close to its true label representation, i.e., the $\widehat{L}_t$ in Formula (10). In this section, we start by averaging all the packet representations that belong to the same class to obtain *packet class representation* for each class. Then we calculate the similarities between each *packet class representation* and label representation and present the corresponding confusion matrix in Figure 10(a). Consequently, the value in the i-th row and j-th column shows the cosine similarity between the *packet class representation* of the i-th class and the label representation of the j-th class. Therefore, the diagonal elements measure the compatibility of these two representations for the same class, which should be as large as possible. While the elements not on diagonal reflect the ability of the two representations to distinguish from other classes, which should be small.

It can be observed that the on-diagonal elements have high values, whereas the off-diagonal elements have low values. This indicates that RBLJAN successfully learns the inherent dependencies and concealed associations between packets and labels. Hence, it is evident why RBLJAN is efficient in detecting the class of the traffic packet.

4.7.3 Label Attention

In RBLJAN, as shown in Formula (6), the label part of the packet representation is the weighted sum of all label vectors. The design of RBLJAN expects to enhance the packet representation by incorporating the true label embedding, which makes it easier to classify the packet into the right class. Considering the label attention vector, i.e., the β, the i-th element of β represents the weight of the i-th label. We hope that the position of the largest element in β corresponds exactly to the true label of the packet. Therefore, to verify that RBLJAN indeed achieves this goal, we average all label attention vectors of the packets (including the header part and the payload part) that belong to the same class, to get the average *label attention vector* of each class, and visualize them in a confusion matrix in Figure 10(b). Specifically, the i-th row represents the average *label attention vector* of that class, and its j-th value represents RBLJAN's attention to the j-th class when classifying the i-th class. It is clear that the largest elements are positioned on the diagonal, which demonstrates that RBLJAN is capable of focusing on the correct label and performing effective classification.

In summary, RBLJAN achieves excellent results on all classification scenarios, outperforming the other state-of-the-art methods in terms of accuracy, detection speed, and robustness. The core mechanism of RBLJAN is also highly interpretable. Such results demonstrate that our header-payload parallel processing-based, joint attention-based, GAN-based RBLJAN is an effective TC model, which is superior to the traditional ML-based Securitas, (only) header-based SAM, image processing-based ICLSTM and DeepPacket, (only) RNN-based EBSNN and Tree-RNN and graph features-based CLE-TFE.

5 Conclusions and Future Work

In this paper, we propose a novel deep learning network, the Robust Byte-Label Joint Attention Network (RBLJAN), for both the packet-level and the flow-level traffic classification. RBLJAN could learn the implicit correlations between bytes and labels through well-designed attention encoders. At the same time, it generates diversified adversarial examples to improve its robustness. Through experiments on four large real-world datasets, RBLJAN achieves superior performance in accuracy, detection speed, and robustness, which proves the practical value of RBLJAN in the real network environment. We have released the source code of RBLJAN at GitHub[6]. In terms of the limitation of this work, RBLJAN only considers the adversarial attack of inserting malicious bytes. In the future, we will further investigate the technology of adversarial examples to improve the robustness of RBLJAN [59]. In addition, We will extend RBLJAN to other traffic identification scenarios (e.g., mobile terminal traffic [40]) and abnormal traffic detection [60]. At last, we will continue to optimize the detection speed and test RBLJAN in a real traffic classification system.

References

[1] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[2] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 IEEE international conference on intelligence and security informatics (ISI)*. IEEE, 2017, pp. 43–48.

[3] S. Alcock and R. Nelson, "Libprotoident: traffic classification using lightweight packet inspection," Technical report, University of Waikato, Tech. Rep., 2012.

[4] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE communications surveys & tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[5] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE communications magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[6] H. Zhou, Y. Wang, X. Lei, and Y. Liu, "A method of improved cnn traffic classification," in *2017 13th international conference on computational intelligence and security (CIS)*. IEEE, 2017, pp. 177–181.

[7] R. Li, X. Xiao, S. Ni, H. Zheng, and S. Xia, "Byte segment neural network for network traffic classification," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–10.

[8] K. Mao, X. Xiao, G. Hu, X. Luo, B. Zhang, and S. Xia, "Byte-label joint attention learning for packet-grained network traffic classification," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 2021, pp. 1–10.

6. https://github.com/ws0407/RBLJAN

[9] Y. Li, B. Liang, and A. Tizghadam, "Robust online learning against malicious manipulation and feedback delay with application to network flow classification," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2648–2663, 2021.

[10] M. S. Sheikh and Y. Peng, "Procedures, criteria, and machine learning techniques for network traffic classification: A survey," *IEEE Access*, vol. 10, pp. 61 135–61 158, 2022.

[11] Z. Liu, S. Li, Y. Zhang, X. Yun, and Z. Cheng, "Efficient malware originated traffic classification by using generative adversarial networks," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–7.

[12] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, "A semantics-aware approach to the automated network protocol identification," *IEEE/ACM transactions on networking*, vol. 24, no. 1, pp. 583–595, 2015.

[13] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[14] G. Xie, Q. Li, and Y. Jiang, "Self-attentive deep learning method for online traffic classification and its interpretability," *Computer Networks*, vol. 196, p. 108267, 2021.

[15] X. Ren, H. Gu, and W. Wei, "Tree-rnn: Tree structural recurrent neural network for network traffic classification," *Expert Systems with Applications*, vol. 167, p. 114363, 2021.

[16] X. Xiao, W. Xiao, R. Li, X. Luo, H. Zheng, and S. Xia, "Ebsnn: extended byte segment neural network for network traffic classification," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3521–3538, 2021.

[17] B. Lu, N. Luktarhan, C. Ding, and W. Zhang, "Iclstm: encrypted traffic service identification based on inception-lstm neural network," *Symmetry*, vol. 13, no. 6, p. 1080, 2021.

[18] H. Zhang, X. Xiao, L. Yu, Q. Li, Z. Ling, and Y. Zhang, "One train for two tasks: An encrypted traffic classification framework using supervised contrastive learning," *arXiv preprint arXiv:2402.07501*, 2024.

[19] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "ndpi: Open-source high-speed deep packet inspection," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2014, pp. 617–622.

[20] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," in *Proceedings of the 2015 ACM conference on special interest group on data communication*, 2015, pp. 213–226.

[21] Y. Wang, X. Yun, and Y. Zhang, "Rethinking robust and accurate application protocol identification: a nonparametric approach," in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*. IEEE, 2015, pp. 134–144.

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

[23] P. Tang, Y. Dong, and S. Mao, "Online traffic classification using granules," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 1135–1140.

[24] W. Li, X.-Y. Zhang, H. Bao, Q. Wang, and Z. Li, "Robust network traffic identification with graph matching," *Computer Networks*, vol. 218, p. 109368, 2022.

[25] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "Kiss: Stochastic packet inspection classifier for udp traffic," *IEEE/ACM Transactions on Networking*, vol. 18, no. 5, pp. 1505–1515, 2010.

[26] H. Liu, Z. Wang, and Y. Wang, "Semi-supervised encrypted traffic classification using composite features set," *Journal of Networks*, vol. 7, no. 8, p. 1195, 2012.

[27] F. Zaki, F. Afifi, S. Abd Razak, A. Gani, and N. B. Anuar, "Grain: Granular multi-label encrypted traffic classification using classifier chain," *Computer Networks*, vol. 213, p. 109084, 2022.

[28] H. Mohanty, A. H. Roudsari, and A. H. Lashkari, "Robust stacking ensemble model for darknet traffic classification under adversarial settings," *Computers & Security*, vol. 120, p. 102830, 2022.

[29] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *Proceedings of the 2006 ACM CoNEXT conference*, 2006, pp. 1–12.

[30] M. Shen, Y. Liu, L. Zhu, X. Du, and J. Hu, "Fine-grained webpage fingerprinting using only packet length information of encrypted traffic," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2046–2059, 2020.

[31] C. Liu, Z. Cao, G. Xiong, G. Gou, S.-M. Yiu, and L. He, "Mampf: Encrypted traffic classification based on multi-attribute markov probability fingerprints," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–10.

[32] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International conference on information networking (ICOIN)*. IEEE, 2017, pp. 712–717.

[33] G. Bendiab, S. Shiaeles, A. Alruban, and N. Kolokotronis, "Iot malware network traffic classification using visual representation and deep learning," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 444–449.

[34] A. Tekerek and M. M. Yapici, "A novel malware classification and augmentation model based on convolutional neural network," *Computers & Security*, vol. 112, p. 102515, 2022.

[35] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*. IEEE, 2019, pp. 1171–1179.

[36] S. Rezaei, B. Kroencke, and X. Liu, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2019.

[37] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "Mimetic: Mobile encrypted traffic classification using multimodal deep learning," *Computer networks*, vol. 165, p. 106944, 2019.

[38] X. Wang, S. Chen, and J. Su, "Real network traffic collection and deep learning for mobile app identification," *Wireless Communications and Mobile Computing*, vol. 2020, pp. 1–14, 2020.

[39] Z. Bu, B. Zhou, P. Cheng, K. Zhang, and Z.-H. Ling, "Encrypted network traffic classification using deep and parallel network-in-network models," *Ieee Access*, vol. 8, pp. 132 950–132 959, 2020.

[40] X. Wang, S. Chen, and J. Su, "App-net: A hybrid neural network for encrypted mobile traffic classification," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 424–429.

[41] A. Nascita, A. Montieri, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "Xai meets mobile traffic classification: Understanding and improving multimodal deep learning architectures," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4225–4246, 2021.

[42] K. Lin, X. Xu, and H. Gao, "Tscrnn: A novel classification scheme of encrypted traffic based on flow spatiotemporal features for efficient management of iiot," *Computer Networks*, vol. 190, p. 107974, 2021.

[43] J. Lan, X. Liu, B. Li, Y. Li, and T. Geng, "Darknetsec: A novel self-attentive deep learning method for darknet traffic classification and application identification," *Computers & Security*, vol. 116, p. 102663, 2022.

[44] X. Yun, Y. Wang, Y. Zhang, C. Zhao, and Z. Zhao, "Encrypted tls traffic classification on cloud platforms," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2022.

[45] X. Meng, Y. Wang, R. Ma, H. Luo, X. Li, and Y. Zhang, "Packet representation learning for traffic classification," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 3546–3554.

[46] G. Bovenzi, A. Nascita, L. Yang, A. Finamore, G. Aceto, D. Ciuonzo, A. Pescapé, and D. Rossi, "Benchmarking class incremental learning in deep learning traffic classification," *IEEE Transactions on Network and Service Management*, 2023.

[47] A. Nascita, A. Montieri, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "Improving performance, reliability, and feasibility in multimodal multitask traffic classification with xai," *IEEE Transactions on Network and Service Management*, 2023.

[48] Z. Song, Z. Zhao, F. Zhang, G. Xiong, G. Cheng, X. Zhao, S. Guo, and B. Chen, "I^{2}rnn: An incremental and interpretable recurrent neural network for encrypted traffic classification," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[49] G. Wang, L. Tang, Z. Yang, L. Yan, P. Liu, and H. Qu, "Deep cnn-rnn with self-attention model for electric iot traffic classification," in *2023 4th International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*. IEEE, 2023, pp. 363–368.

[50] Y. Li, Y. Huang, S. Seneviratne, K. Thilakarathna, A. Cheng, G. Jourjon, D. Webb, D. B. Smith, and R. Y. Da Xu, "From traffic classes to content: A hierarchical approach for encrypted traffic classification," *Computer Networks*, vol. 212, p. 109017, 2022.

[51] K. Li, W. Ma, H. Duan, H. Xie, J. Zhu, and R. Liu, "Unbalanced

network attack traffic detection based on feature extraction and gfda-wgan," *Computer Networks*, vol. 216, p. 109283, 2022.

[52] Y. Liang, Y. Xie, S. Tang, S. Yu, X. Liu, and J. Hu, "Network traffic content identification based on time-scale signal modeling," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[53] H. Zhang, L. Yu, X. Xiao, Q. Li, F. Mercaldo, X. Luo, and Q. Liu, "Tfe-gnn: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2066–2075.

[54] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE access*, vol. 5, pp. 18 042–18 050, 2017.

[55] A. L'heureux, K. Grolinger, H. F. Elyamany, and M. A. Capretz, "Machine learning with big data: Challenges and approaches," *Ieee Access*, vol. 5, pp. 7776–7797, 2017.

[56] G. Xie, Q. Li, Y. Jiang, T. Dai, G. Shen, R. Li, R. Sinnott, and S. Xia, "Sam: self-attention based deep learning method for online traffic classification," in *Proceedings of the Workshop on Network Meets AI & ML*, 2020, pp. 14–20.

[57] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[58] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.

[59] A. M. Sadeghzadeh, S. Shiravi, and R. Jalili, "Adversarial network traffic: Towards evaluating the robustness of deep-learning-based network traffic classification," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1962–1976, 2021.

[60] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescape, "Ai-powered internet traffic classification: Past, present, and future," *IEEE Communications Magazine*, pp. 1–7, 2023.

[61] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[62] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[63] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414.

[64] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1249, 2018.

[65] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

Xi Xiao is an associate professor with the Shenzhen International Graduate School, Tsinghua University. He got his Ph.D. degree in 2011 in the State Key Laboratory of Information Security, Graduate University of the Chinese Academy of Sciences. His research interests focus on information security and the computer network.

Shuo Wang is a student with the Shenzhen International Graduate School, Tsinghua University. His research interests focus on machine learning, computer network, and information security.

Guangwu Hu is the Vice Dean of School of Computer Science, Shenzhen Institute of Information Technology. He received the Ph.D. degree in the Department of Computer Science and Technology and held a post-doctoral position from Tsinghua University. His research interests include Next-Generation Internet, Cyber Security, and Blockchain Technology.

Qing Li is an Associate Researcher with the Peng Cheng Laboratory, Shenzhen, China. He received the B.S. degree in computer science and technology from the Dalian University of Technology in 2008, and the Ph.D. degree in computer science and technology from Tsinghua University in 2013. His research interests include network function virtualization, in-network caching/computing, and intelligent self-running networks.

Kelong Mao is a student with the Gaoling School of Artificial Intelligence, Renmin University. He received the M.S degrees in Computer Science from Tsinghua University in 2021. His research interests include Conversational Information Seeking and Large Language Models.

Xiapu Luo He is an associate professor with the Department of Computing, The Hong Kong Polytechnic University. His research focuses on Blockchain/smart contracts, mobile/IoT security and privacy, network/web security and privacy, software engineering, and internet measurement.

Bin Zhang is currently a Researcher with the Cyberspace Security Research Center, Peng Cheng Laboratory, Shenzhen, China. He received the Ph.D. degree in computer science and technology from Tsinghua University in 2012. He was a Postdoctoral Researcher with Nanjing Telecommunication Technology Institute from 2014 to 2017. His research interests include network anomaly detection, network traffic measurement, and information privacy security.

Shutao Xia is a full professor with the Shenzhen International Graduate School, Tsinghua University. He received the B.S. degree in mathematics and the Ph.D. degree in applied mathematics from Nankai University in 1992 and 1997, respectively. His research interests include coding and information theory, machine learning, and deep learning.