SAM: Self-Attention based Deep Learning Method for Online Traffic Classification

Conference Paper · August 2020

DOI: 10.1145/3405671.3405811

8 authors, including:

Qing Li
Graduate School at Shenzhen, Tsinghua Unviersity
86 PUBLICATIONS **449** CITATIONS
SEE PROFILE

Tao Dai
Tsinghua University
54 PUBLICATIONS **221** CITATIONS
SEE PROFILE

SAM: Self-Attention based Deep Learning Method for Online Traffic Classification*

Guorui Xie
Tsinghua University,
Peng Cheng Laboratory

Qing Li†
Southern University of
Science and Technology,
Peng Cheng Laboratory

Yong Jiang
Tsinghua University

Tao Dai
Tsinghua University

Gengbiao Shen
Tsinghua University

Rui Li
Tsinghua University

Richard Sinnott
University of Melbourne

Shutao Xia
Tsinghua University

ABSTRACT

Network traffic classification categorizes traffic classes based on protocols (e.g., HTTP or DNS) or applications (e.g., Facebook or Gmail). Its accuracy is a key foundation of some network management tasks like Quality-of-Service (QoS) control, anomaly detection, etc. To further improve the accuracy of traffic classification, recent researches have introduced deep learning based methods. However, most of them utilize the privacy-concerned payload (user data). Besides, they generally do not consider the dependency of bytes in a packet, which we believe can be exploited for the more accurate classification. In this work, we treat the initial bytes of a network packet as a language and propose a novel Self-Attention based Method (SAM) for traffic classification. The average F1-scores of SAM on protocol and application classification are 98.62% and 98.93%. With the higher accuracy of SAM, better QoS and anomaly detection can be met.

CCS CONCEPTS

• **Networks** → **Network protocols**; • **Computing methodologies** → **Classification and regression trees**.

*This work is supported by the National Key Research and Development Program of China under Grant 2018YFB1800204 and the project "PCL Future Greater-Bay Area Network Facilities for Large-scale Experiments and Applications (LZC0019)".
†Corresponding author, liq8@sustech.edu.cn

NetAI'20, August 14, 2020, Virtual Event, NY, USA

ACM ISBN 978-1-4503-8043-0/20/08…$15.00
https://doi.org/10.1145/3405671.3405811

ACM Reference Format:
Guorui Xie, Qing Li, Yong Jiang, Tao Dai, Gengbiao Shen, Rui Li, Richard Sinnott, and Shutao Xia. 2020. SAM: Self-Attention based Deep Learning Method for Online Traffic Classification. In *Workshop on Network Meets AI & ML (NetAI'20), August 14, 2020, Virtual Event, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3405671.3405811

1 INTRODUCTION

Network traffic classification categorizes traffic classes based on protocols (e.g., HTTP or DNS) or applications (e.g., Facebook or Gmail) [18]. It is vital in network management tasks such as Quality-of-Service (QoS) control and anomaly detection. [20]. To date, various types of methods have been proposed to solve this problem. They can be divided into three categories: payload inspection based methods, traditional machine learning (ML) based methods and deep learning (DL) based methods. Especially, the DL based methods are proposed to generate features (signatures) adaptively and achieve a much higher improvement. [13, 15, 19]. But these DL based methods use the payload as the input, which conflicts with user privacy protection. Besides, they ignore the dependency of input bytes, which we believe can be exploited for the more accurate classification.

Generally, a network packet consists of three parts: the IP header, the transport layer (TCP/UDP) header, and the payload. The initial bytes in different positions are relevant and indicate meaningful information. Therefore, we treat the first l bytes of a packet as a language during classification. This means:

- User privacy can be guaranteed by limiting the number of bytes used in the classification.
- We can exploit self-attention relating different positions of a single input sequence to compute a representation of the sequence for further performance improvement.

Accordingly, we propose a novel Self-Attention based Method (SAM) in this paper. Self-attention, which has been used successfully in language tasks, is first introduced into traffic classification. Based on the experimental results, we find that:

- SAM outperforms the other state-of-the-art methods. The average F1-scores of SAM on protocol and application classification are 98.62% and 98.93%.
- SAM shows a classification speed of 0.18 ms/packet with a GTX 1080 Ti graphics card, which proves the feasibility of SAM in the scenario of online differentiated traffic scheduling.

2 RELATED WORK

2.1 Payload Inspection based Methods

Payload inspection checks the packet payload against a set of known protocol signatures, e.g., '\GET' signature in HTTP traffic [11]. Some famous DPI libraries are libprotoident [1], OpenDPI [22], and nDPI [8]. nDPI contains specialized protocol decoders of many well-known protocols (e.g., HTTP and FTP). Also, it uses the port number as the priority of decoders while parsing packets. For instance, a packet with port 80 is likely parsed by HTTP decoder at first. However, parsing payload involves privacy issues and raises some legal issues [7]. Meanwhile, generating protocol decoders costs much effort of domain experts and is useless on unpublished protocols.

2.2 Traditional Machine Learning based Methods

Traditional ML based methods assume that traffic can be distinguished from each other via statistical features, e.g., per-flow duration or mean packet size. Auld et al. proposed a Bayesian Neural Networks (BNN) with a series of packet features for P2P traffic classification [2]. Gil et al. use K-Nearest Neighbor and C4.5 decision tree algorithms with time-related features, e.g., the duration of the flow, flow bytes per second, and forward/backward arrival intervals [9]. Yun et al. present Securitas, which uses the word bag model and Latent Dirichlet Allocation to get the feature of protocols. Then, a classifier, such as SVM, C4.5 decision tree and Bayes Network, is constructed [23]. Most of these ML methods depend on hand-crafted features, which is time-consuming and error-prone especially with the rapid growth of network traffic types.

2.3 Deep Learning based Methods

DL based methods are unlike traditional ML based methods or packet inspection, because it does not rely on experts to extract features (signatures). Moreover, it has a stronger capacity of learning in comparison to traditional ML methods and thus can achieve a higher performance [19]. Thus, DL attracts a lot of attention in traffic classification. Lotfollahi et al. propose Deep Packet for traffic classification [15]. In Deep Packet, an entire packet is passed to a DL based framework for identification. The framework itself can be implemented by either stacked Auto-encoders (SAE) or one-dimensional CNN. Li et al. put forward an RNN based method called BSNN for traffic classification [13]. In BSNN, the payload of a packet is split into byte segments. These segments are then fed into corresponding RNN encoders. Finally, a softmax function is applied for classification. The RNN component of BSNN is based on either Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) [6].

Although these methods show a better performance, they take payload for granted without considering the privacy issues. Furthermore, none of them take the dependency of bytes in a packet into account.

2.4 Self-Attention

In a human sentence, the words in different positions are related. For instance, the subject can affect the forms of the predicate (singular or plural), and the time adverbial determines whether the predicate adds "ed". Hence, self-attention is proposed to consider different positions of a single sequence when it is employed to compute a representation of the sequence [21]. It enables an extra performance improvement and thus is widely used in some NLP tasks [5, 14, 17].

3 METHODOLOGY

3.1 Packet Format

A general packet contains the IP header, the transport layer header, and the payload. Besides, the bytes in different positions affect each other. For instance, the "Version" in IP header decides the length of "Source IP address" is either four bytes (IPv4) or sixteen bytes (IPv6). Hence, the well-formatted headers can be treated as a special language when applications exchange information through it. Hence, self-attention is unitized in SAM for the reason that self-attention is presented to make use of positions in a language sequence and compute a better representation of that sequence.

3.2 The Structure of SAM

The structure of SAM is provided in Figure 1. We now discuss all the main components in detail.

Preprocessing. Since headers lie at the beginning of a packet, we choose the initial l bytes of a packet as the input of SAM. For connection-oriented traffic, the common header length is 40 bytes, which is larger than that of connectionless traffic. And connection-oriented protocols are more commonly used. Thus, a 40-byte vector ($l = 40$) $\boldsymbol{p}$ is selected. Note

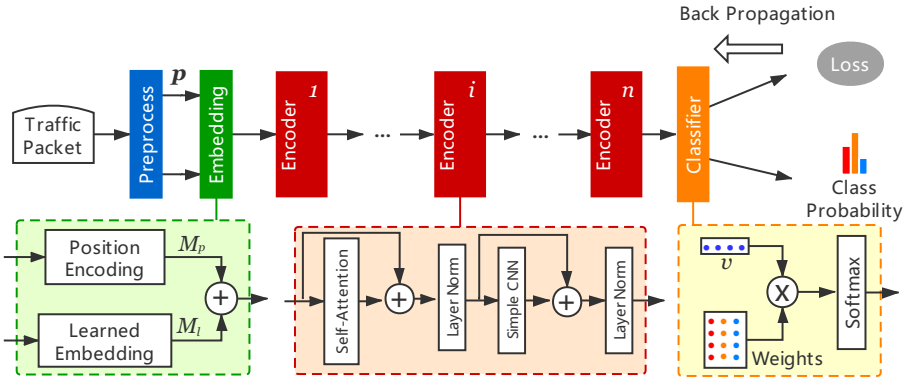

Figure 1: The structure of SAM. There are four main components in SAM, including the preprocessing, the embedding, the encoder, and the classifier.

that limiting the number of bytes used in a packet is helpful on user privacy protection. There are some techniques that confuse the classifier including random port assignment and network address translation (NAT). Hence, we mask the IP address and the port number with zeros to get rid of this confusion.

Embedding. Embedding contains two sub-components: position encoding and learned embedding. As is described in Section 3.1, the bytes in different positions affect each other, so we need to factor in the position information in SAM. Thus, the position encoding is introduced [21]. The position encoding encodes the position pos of a byte in a given input sequence to a d-dimensional position vector PE_{pos} through the following equations:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d}) \tag{1}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d}) \tag{2}$$

where $2i, 2i+1 \in [0, d-1]$ is the channel of the vector to be generated. We select sine and cosine functions with a constant 10000. Thus, each dimension of the output corresponds to a sinusoid. And the wavelengths form a geometric progression in $[2\pi, 10000 \times 2\pi]$. These operations allow the model to learn the relative positions since for any fixed offset k, PE_{pos+k} can be described as a linear function of PE_{pos}.

The learned embedding is used to enrich the meaning of each byte in vector $\boldsymbol{p}$ [4]. Specifically, let $p_i \in [0, 255]$ be an element of $\boldsymbol{p}$, we have:

$$\boldsymbol{y_i} = W_l \text{onehot}(p_i) \tag{3}$$

where onehot denotes the one hot encoding for p_i; W_l is the learned transformation matrix that is updated adaptively during training, and $\boldsymbol{y_i}$ is the (converted) d-dimensional embedding vector.

Therefore, for each $p_i \in \boldsymbol{p}$, we have a PE_{pos} and a $\boldsymbol{y_i}$ respectively. For $\boldsymbol{p}$, we subsequently have $M_p = \mathrm{F}_{\text{pos}}(\boldsymbol{p})$ and $M_l = \mathrm{F}_{\text{emb}}(\boldsymbol{p})$ where F_{pos} and F_{emb} denote the position encoding and learned embedding respectively. M_p and M_l are the corresponding results in the form of matrix.

Encoder. As is shown in Figure 1, each encoder includes the self-attention, the residual connection, the layer normalization, and a simple one-dimensional CNN.

Self-attention can be described as a function that maps three matrices: Query (Q), Key (K) and Value (V) to a weighted result W_aV. "Self" indicates that $Q = K = V$ [21]. W_a is given as:

$$W_a = \text{softmax}(\frac{QK^T}{\sqrt{d'}}) \tag{4}$$

where d' is the dimension of K. The softmax function here is defined as:

$$\text{softmax}(\boldsymbol{y})_j = \frac{e^{y_j}}{\sum_i e^{y_i}} \tag{5}$$

where we apply the exponential function to each element y_j of the input vector $\boldsymbol{y}$ and normalize these values by dividing the sum of all the associated indices. Moreover, the input of the encoder is a matrix in $\mathbb{R}^{l \times d}$ which is so large that applying the self-attention on it may result in the loss of local information. Therefore, we divide the d channels into h groups (so $d' = d/h$). The specific procedures of our self-attention are shown in Figure 2.

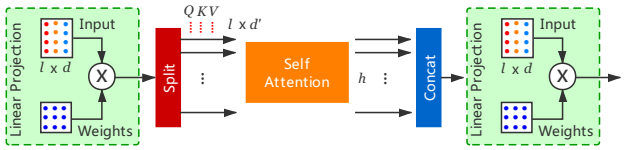

Figure 2: The self-attention. We divide the d channels into h groups (so $d' = d/h$) and apply the self-attention mechanism respectively.

Practice shows that deeper networks sometimes work better, so we cascade n encoders to deepen SAM. But deep networks have the characteristics of slow training. Thus, we use the residual connection [10] and the layer normalization [3] to speed up the training. The residual connection can be described as $y = \mathrm{F}(x)+x$ where the function F denotes some operations (self-attention or CNN) performed on input x. In Figure 1, the two "+" symbols in the *encoder component* show the residual connection, the left arrow of "+" indicates $\mathrm{F}(x)$, well the top arrow means x. It is worth mentioning that the residual connection requires us to ensure that the input dimension d_x and the output dimension d_y are always the same. Hence we set $d_x = d_y = d$, where d is the dimension of the embedding vector mentioned before. Besides, the layer normalization maps the elements of output y to floats ranging from 0 to 1 to help with the convergence.

The structure of CNN in the encoder is simple. There is one convolution layer followed by ReLU and Maxpool respectively. The settings of convolution and pooling layers in the CNN module are given as: $kernel = 3$, $stride = 1$, $pad = 1$.

Classifier. The result of the last encoder is a feature matrix $M' \in \mathbb{R}^{l \times d}$. We sum it up by row to get a feature vector $\boldsymbol{v} \in \mathbb{R}^d$. Then we apply a linear projection on $\boldsymbol{v}$ to get the vector $\boldsymbol{v_N} \in \mathbb{R}^N$, where N is the number of categories. $\boldsymbol{v_N}$ is the input of the softmax function:

$$\boldsymbol{\sigma} = \mathrm{softmax}(\boldsymbol{v_N}) \tag{6}$$

where softmax is defined in Equation (5). Each element value $\sigma_j \in \boldsymbol{\sigma}$ represents the probability that the packet belongs to the corresponding category j. If σ_j has the maximum value, the label j is selected as the final prediction.

Meanwhile, in the training phase, the cross-entropy loss is generated based on $Loss = \frac{-\sum_{\boldsymbol{\sigma}} \log \sigma_j}{T}$ where $\boldsymbol{\sigma}$ is the class probability vector in Equation (6), and j represents the true class index. T is the number of packets in the current training batch. All variables in SAM will be updated by back propagation with $Loss$.

4 EXPERIMENTAL EVALUATION

4.1 Experiment Setup

Dataset. The details of these datasets are shown in Table 1. We utilize two datasets in our evaluation: **i)** The protocol dataset that contains traffic of nine protocols. These protocols are famous, and we capture their traffic via their registered port number in our lab. **ii)** The application dataset is the UNB ISCX VPN-nonVPN dataset which contains traffic of 17 applications including Facebook, YouTube, etc. [9]. Originally, these traffics are labeled by the actions that generates them (browsing, email, etc.), we relabel them by their generating applications.

We identify the packets of different protocols (DNS, HTTP, etc.) on dataset i, different applications (Gmail, Youtube, etc.) on dataset ii. These datasets are randomly split as follows: 60% for training, 20% for validation, and 20% for testing. In all the experiments, the model which has the best performance on validation is chosen for testing.

Table 1: Details of datasets.

Dataset i		Dataset ii			
Class	Packet amount	Class	Packet amount	Class	Packet amount
Bittorrent	11833	AIM chat	1785	SFTP	416813
DNS	2483	Email	17578	Skype	23710
Finger	158	Facebook	11233	Spotify	40592
HTTP	10067	FTPS	3784620	Tor	326251
HTTPS	27341	Gmail	11014	Torrent	108227
SMTP	110	Hangouts	984241	Vimeo	145947
SSH	3959	ICQ	1106	Voipbuster	2480
eDonkey	82828	Netflix	299057	YouTube	209785
Whois	1615	SCP	447792		

Baselines. For protocol classification, a payload inspection based method (nDPI), a traditional machine learning based method (Securitas), and two deep learning based methods (BSNN, Deep Packet) are compared with SAM. These methods are introduced in Section 2. As is mentioned, these methods have different implementations. For Securitas, the classifier can be SVM (Securitas-SVM), C4.5 decision tree (Securitas-C4.5) or BN (Securitas-Bayes). For BSNN, the structure can be based on LSTM (BSNN-LSTM) or GRU (BSNN-GRU). For Deep Packet, SAE (DeepPacket-SAE) or CNN (DeepPacket-CNN) can be the classifier.

Metrics. We calculate Recall (R.), Precision (P.), and F1-score (F1.) for each class as follows: $Recall = \frac{TP}{TP+FN}$, $Precision = \frac{TP}{TP+FP}$, $F1\text{-}score = \frac{2 \times Recall \times Precision}{Recall + Precision}$ where TP, FP and FN stand for true positive, false positive and false negative, respectively. For each class, TP represents the number of packets sorted into the (true) correct class. FP refers to the number of instances that are incorrectly categorized into a particular class. FN represents the number of instances that are classified into other classes but actually belong to a particular class. In detail, when calculating F1 of class A, instances of class A are regarded as the positive class, while instances of other classes (B, C, etc.) are all regarded as the negative class. This procedure is the same as that in [13].

Implementation of SAM. For SAM, we use the Adam optimizer, $batch\_size = 64$, and dropout ratio is 0.1 [12]. Other significant hyper-parameters are chosen by grid search: the embedding vector dimension $d = 256$, the self-attention group $h = 4$, and the number of encoders $n = 2$. We utilize PyTorch, the high-performance DL library, to implement SAM [16].

Table 2: Protocol classification results on dataset i.

Protocols	Bittorrent			DNS			Finger			HTTP			HTTPS		
Metrics	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%
nDPI	70.73	40.94	51.86	74.04	52.33	61.32	-	-	-	86.83	86.34	86.58	-	-	-
Securitas-SVM	-	-	-	41.91	82.11	55.49	-	-	-	**99.61**	67.20	80.25	**100.00**	62.67	77.05
Securitas-C4.5	97.43	**100.00**	98.70	99.57	**100.00**	**99.78**	81.25	**100.00**	89.66	56.13	**99.82**	71.86	99.88	63.77	77.85
Securitas-Bayes	96.15	92.49	94.28	**100.00**	97.57	98.77	92.31	**100.00**	96.00	53.21	92.93	67.67	28.92	99.41	44.80
BSNN-GRU	97.25	99.90	98.56	98.76	98.56	98.66	**100.00**	96.43	**98.18**	69.41	97.71	81.16	29.17	99.52	45.12
BSNN-LSTM	97.30	98.51	97.90	99.59	93.77	96.59	**100.00**	81.82	90.00	63.76	95.76	76.55	39.90	67.25	50.09
DeepPacket-CNN	96.39	99.37	97.86	98.99	98.40	98.69	-	-	-	94.24	94.71	94.47	97.78	96.98	97.38
DeepPacket-SAE	94.78	98.18	96.45	99.18	97.37	98.27	-	-	-	88.80	90.63	89.70	95.73	94.40	95.06
SAM	**99.67**	99.83	**99.75**	98.96	**100.00**	99.48	**100.00**	93.75	96.77	98.94	99.09	**99.02**	99.65	**99.60**	**99.62**

Protocols	SMTP			SSH			eDonkey			Whois			Average		
Metrics	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%
nDPI	94.55	16.05	27.44	94.25	89.15	91.63	-	-	-	94.98	74.43	83.46	57.26	39.92	44.70
Securitas-SVM	-	-	-	-	-	-	97.91	65.62	78.57	-	-	-	37.71	30.84	32.37
Securitas-C4.5	**100.00**	**100.00**	**100.00**	-	-	-	64.62	98.91	78.17	**100.00**	**100.00**	**100.00**	77.65	84.72	79.56
Securitas-Bayes	**100.00**	**100.00**	**100.00**	29.24	**100.00**	45.25	55.65	96.37	70.56	95.65	97.78	96.70	72.35	97.39	79.34
BSNN-GRU	70.00	93.33	80.00	2.99	62.16	5.70	99.73	76.12	86.34	96.77	97.83	97.30	73.79	91.28	76.78
BSNN-LSTM	75.00	83.33	78.95	2.86	45.83	5.38	95.98	78.26	86.22	**100.00**	95.88	97.89	74.93	82.27	75.51
DeepPacket-CNN	-	-	-	99.08	95.30	97.15	99.85	99.50	99.67	82.34	82.34	82.34	74.30	74.07	74.17
DeepPacket-SAE	-	-	-	97.75	95.24	96.48	99.66	98.97	99.31	60.42	69.69	64.72	70.70	71.61	71.11
SAM	91.67	95.65	93.62	**99.88**	**100.00**	**99.94**	**99.98**	**99.91**	**99.94**	99.14	99.71	99.43	**98.65**	**98.62**	**98.62**

4.2 Metric Results of Different Methods

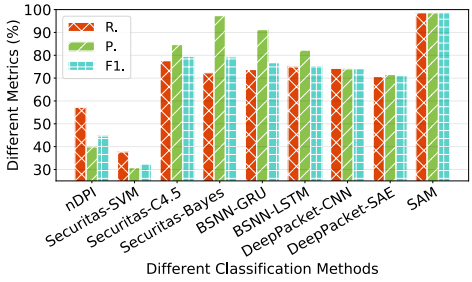

Figure 3: The average recall, precision, and F1-score of different methods on protocol classification.

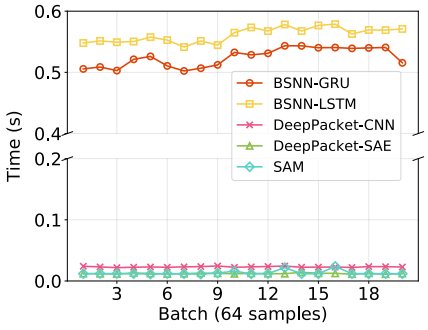

Figure 4: The average training time.

Protocol classification. Experiment results on dataset i are shown in Table 2. For nDPI, there are no results of Finger, HTTPS, and eDonkey because no expert makes an effort to add the corresponding decoders. As for Securitas, although Securitas-C4.5 and Securitas-Bayes have an F1-score of 100% on DNS and SMTP, these are binary classifiers and easier than SAM. Furthermore, on protocols like Bittorrent, Finger, SSH, etc., Securitas predicts all their packets as negative or positive, which makes no sense. As for BSNNs (BSNN-GRU and BSNN-LSTM), their performance is unstable. The F1-score ranges from 5% (SSH) to 98% (Finger). Especially, the F1-score is lower on encrypted protocols (HTTPS 45%, SSH 5%). Although DeepPacket-CNN and DeepPacket-SAE have a high performance (over 94%) on many protocols including Bittorent, DNS, HTTPS, etc., they can not handle the rare traffic. For instance, the traffic of Finger and SMTP is rare, which causes a failure of Deep Packet. Generally, SAM has superiority over other methods: it has the highest average Recall (98.65%), Precision (98.62%), and F1-score (98.62%). Furthermore, Figure 3 gives an intuitive feeling about average metrics of different methods.

Application classification. The results on dataset ii are illustrated in Table 3. As nDPI and Securitas are protocol-oriented classifiers, they are not listed in this table. According to Table 3, BSNNs have the poorest average F1-score (below 45%) even though the author claims the support of application classification. We argue that their experiments are not sufficient, because they only compared five applications [13].

Table 3: Application classification results on dataset ii.

Applications	AIM chat			Email			Facebook			FTPS			Gmail			Hangouts		
Metrics	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%
BSNN-GRU	19.23	76.27	30.72	69.96	78.33	73.91	52.17	93.90	67.07	60.80	82.86	70.14	99.58	99.04	99.31	70.25	86.73	77.63
BSNN-LSTM	22.22	55.32	31.71	80.14	83.04	81.57	72.66	73.23	72.94	63.05	85.99	72.75	99.46	99.65	99.55	64.46	97.50	77.61
DeepPacket-SAE	88.08	89.53	88.80	77.96	78.36	78.16	99.73	99.68	99.70	83.50	87.24	85.33	99.80	99.74	99.77	91.55	95.51	93.49
DeepPacket-CNN	88.62	85.49	87.03	59.70	82.67	69.33	99.72	**99.87**	**99.80**	88.67	88.87	88.77	99.82	99.79	99.81	83.41	95.09	88.87
SAM	**97.67**	**96.61**	**97.14**	**99.02**	**93.66**	**96.26**	**99.98**	99.60	99.79	**97.29**	**92.62**	**94.90**	**99.88**	**99.96**	**99.92**	**94.25**	**98.76**	**96.45**

Applications	ICQ			Netflix			SCP			SFTP			Skype			Spotify		
Metrics	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%
BSNN-GRU	2.31	80.11	4.49	10.72	99.80	19.36	10.72	99.80	19.36	11.16	98.02	20.04	75.10	85.14	79.81	2.45	22.83	4.43
BSNN-LSTM	4.95	88.42	9.37	10.73	99.77	19.37	10.73	99.77	19.37	11.20	98.40	20.11	78.06	93.25	84.98	5.15	69.85	9.59
DeepPacket-SAE	99.72	99.73	99.72	99.99	99.98	99.99	99.99	99.98	99.99	99.81	99.86	99.83	98.51	98.55	98.53	97.41	97.84	97.62
DeepPacket-CNN	99.63	99.82	99.72	99.99	99.97	99.98	99.99	99.97	99.98	99.88	99.94	99.91	99.21	97.66	98.43	97.83	96.41	97.11
SAM	**99.95**	**100.00**	**99.98**	**100.00**	**99.99**	**99.99**	**100.00**	**99.99**	**99.99**	**99.98**	**99.96**	**99.97**	**99.51**	**99.95**	**99.73**	**99.69**	**98.14**	**98.91**

Applications	Tor			Torrent			Vimeo			Voipbuster			YouTube			Average		
Metrics	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%	R.%	P.%	F1.%
BSNN-GRU	17.01	98.99	29.03	4.04	73.72	7.65	7.59	83.68	13.92	77.39	98.22	86.57	4.34	73.65	8.21	34.99	84.18	41.86
BSNN-LSTM	17.30	98.59	29.44	11.44	94.33	20.40	9.04	73.81	16.11	85.08	89.46	87.22	5.26	66.92	9.75	38.29	86.31	44.81
DeepPacket-SAE	99.90	99.89	99.90	98.78	98.69	98.73	98.87	99.43	99.15	99.88	99.92	99.90	99.47	98.82	99.14	96.06	96.63	96.34
DeepPacket-CNN	99.88	**99.96**	99.92	98.75	98.68	98.72	99.13	99.44	99.28	99.87	99.90	99.88	**99.57**	98.86	99.21	94.92	96.61	95.63
SAM	**99.95**	99.89	**99.92**	**99.32**	**99.36**	**99.34**	**99.89**	**99.95**	**99.92**	**99.98**	**99.97**	**99.97**	99.22	**100.00**	**99.61**	**99.15**	**98.73**	**98.93**

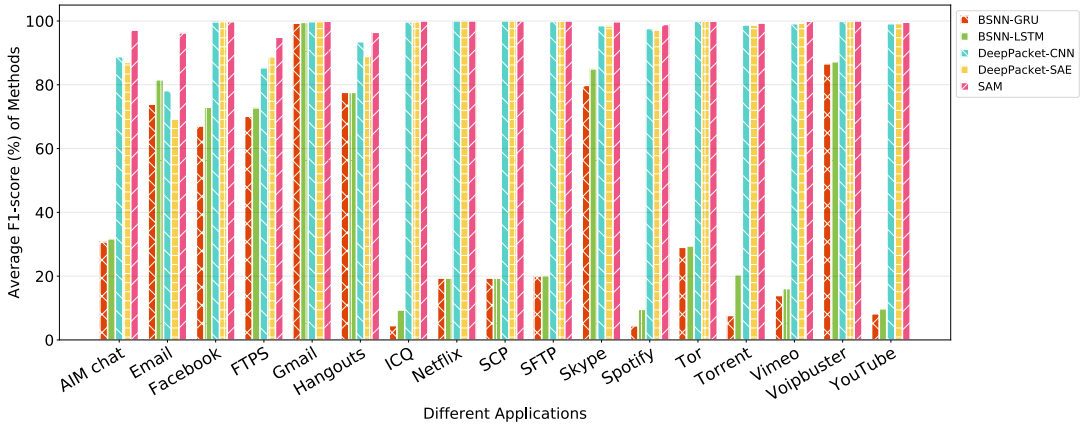

Figure 5: The average F1-score of different DL methods on application classification. Generally, SAM and DeepPacket are better on ICQ, Spotify, etc.

As for Deep Packet and SAM, their average F1-score is over 95%. Especially, SAM has the highest average F1-score of 98.93%. Figure 5 gives an intuitive feeling about average F1-score of different methods too. Besides, in this experiment, we use a GTX 1080 Ti graphics card and the CPU is Intel (R) Xeon (R) E5-2603 v2 @ 1.80 GHz. Figure 4 describes the time spent per batch of different models. As seen, SAM is faster (50x) than BSNNs. As there are 64 packets per batch, the average speed of SAM is 0.18 ms/packet, which is fast enough for online traffic classification.

5 CONCLUSION

In this paper, we present a novel Self-Attention based Method (SAM) to explore the correlation among input bytes. To the best of our knowledge, this is the first attempt to introduce self-attention into traffic classification. Besides, user privacy is guaranteed by limiting the number of packet bytes. SAM gains the highest F1-score on the protocol (98.62%) and application (98.93%) classification than other methods. Moreover, the speed of SAM is 0.18 ms/packet, which is faster (50x) than that of BSNN.

REFERENCES

[1] Shane Alcock and Richard Nelson. 2012. *Libprotoident: Traffic Classification using lightweight packet inspection.* Technical Report. WAND Network Research Group.

[2] T. Auld, A. W. Moore, and S. F. Gull. 2007. Bayesian Neural Networks for Internet Traffic Classification. *Transactions on Neural Networks* 18 (2007), 223–239.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research* 3 (2003), 1137–1155.

[5] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733* (2016).

[6] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555* (2014).

[7] A. Dainotti, A. Pescape, and K. C. Claffy. 2012. Issues and Future Directions in Traffic Classification. *Network* 26, 1 (2012), 35–40.

[8] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. 2014. ndpi: Open-source high-speed deep packet inspection. In *Proceedings of the 2014 International Wireless Communications and Mobile Computing Conference*. IEEE, Washington, DC., 617–622.

[9] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2016. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy*. SciTePress, Setúbal, Portugal, 407–414.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the 2016 Conference on Computer Vision and Pattern Recognition*. IEEE, Washington, DC., 770–778.

[11] Jawad Khalife, Amjad Hajjar, and Jesus Diaz-Verdejo. 2014. A Multilevel Taxonomy and Requirements for an Optimal Traffic-classification Model. *International Journal of Network Management* 24, 2 (2014), 101–120.

[12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[13] Rui Li, Xi Xiao, Shiguang Ni, Haitao Zheng, and Shutao Xia. 2018. Byte Segment Neural Network for Network Traffic Classification. In *Proceedings of the 26th International Symposium on Quality of Service*. ACM, New York, 1–10.

[14] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* (2017).

[15] Mohammad Lotfollahi, Ramin Shirali Hossein Zade, Mahdi Jafari Siavoshani, and Mohammdsadegh Saberian. 2017. Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning. *arXiv preprint arXiv:1709.02656* (2017).

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the 2019 Advances in Neural Information Processing Systems*. Curran Associates, Inc., New York, 8024–8035.

[17] Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304* (2017).

[18] Shahbaz Rezaei and Xin Liu. 2019. Deep learning for encrypted traffic classification: An overview. *Communications Magazine* 57, 5 (2019), 76–81.

[19] Jürgen Schmidhuber. 2015. Deep Learning in Neural Networks: An Overview. *Neural Networks* 61 (2015), 85–117.

[20] Bagui Sikha, Fang Xingang, Kalaimannan Ezhil, Bagui Subhash C, and Sheehan Joseph. 2017. Comparison of Machine-Learning Algorithms for Classification of VPN Network Traffic Flow Using Time-Related Features. *Journal of Cyber Security Technology* 1, 2 (2017), 108–126.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems*. Curran Associates, Inc., New York.

[22] Yong Wei, Z Yun-Feng, and Li-chao Guo. 2011. Analysis of message identification for OpenDPI. *Computer Engineering* (2011), S1.

[23] Xiaochun Yun, Yipeng Wang, Yongzheng Zhang, and Yu Zhou. 2016. A semantics-aware approach to the automated network protocol identification. *Transactions on Networking* 24, 1 (2016), 583–595.