ProNet: Toward Payload-Driven Protocol Fingerprinting via Convolutions and Embeddings

Yafei Sang[1,2], Yongzheng Zhang[1,2](✉), and Chengwei Peng[2,3]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{sangyafei,zhangyongzheng}@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
pengchengwei@iie.ac.cn

[3] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

Abstract. Protocol fingerprinting (PF) focuses on the capability to derive a series of distinguishable features for recognizing which protocol or application generated the network traffic. Unfortunately, deep packet inspection (DPI), a widely adopted method for PF, requires significant expert effort to develop and maintain protocol signatures. Additionally, the new solution paradigm, deep flow inspection (DFI) using machine learning for PF, also relies on hand-designed features. In this paper, we present *ProNet*, a payload based approach to protocol fingerprinting, which overcomes the limitation of artificial feature engineering. The key novelty of *ProNet* is two-fold: *(i)* it takes generic, raw short packet payloads as input, instead of the typical flow-statistical-features (*e.g.*, port, packet size, packet-interval); *(ii)* it learns to simultaneously extract features via convolutional operations on the *byte-level* embeddings and *ngram-level* embeddings. We implement and evaluate *ProNet* on real-world traces, including DNS, QQLive, PPLive, PPStream, SopCast, DHCP, NBNS, HTTP, SMTP and SMB. Our experiment results show that *ProNet* achieves over 99% precision and recall with low false-positives (less than 1%) and nearly no false-negatives.

Keywords: Protocol fingerprinting · Convolutions · Embedding

1 Introduction

Motivation: Last years witnessed a very fast-paced deployment of new Internet applications, ignited by the introduction of the successful Peer-to-Peer (P2P) paradigm and fueled by the growth of Internet access rates. This undermined the reliability of the L7-layer protocol identification mechanisms. Protocol identification (PI for short in this paper), *associating traffic flows/packets with the applications/protocols*, is a fundamental and crucial component for network monitoring, traffic billing, Quality of Service (QoS), security services (such as IDS/IPS), and so on. The existing PI methods can roughly be divided into two categories: DPI and DFI. The core of DPI is to match the content of the traffic payload with the pre-constructed literal fingerprints, also called signatures, typically in form of regular expression. DFI is to define common flow statistical fingerprints (features) such as per-flow duration and volume, mean packet size, inter-packet times of a flow. Note that the key of both DPI and DFI is the protocol fingerprints for PI task. Thus, how to systematically and efficiently automated extract these fingerprints from network payloads remains a challenging issue. We refer to *Protocol Fingerprinting* (PF for short in this paper) as the process that a distinguishable unique pattern is extracted in the absence of the formal protocol specification (including parameters, format, and semantics).

Limitations of Prior Art: The problem PF has received wide research interests in the past decades. However, existing studies including DPI and DFI fingerprints rely heavily on hand-designed features by artificial experience and prior knowledge. The manual feature engineering is a laborious, time-consuming process. Specifically, DPI fingerprints inference falls into some drawbacks. *(i)* A majority of proprietary protocols lack publicly available documentations, although there are standard RFCs for the public-domain protocols. *(ii)* The labor-intensive manual protocol fingerprints extraction process has to be repeated from time to time so as to maintain the latest signature repository. DFI statistical fingerprints also face some limitations. *(i)* They require an in-depth exploration of the feature representation of a given traffic flow type, and an exploration of what machine learning detection approaches yield the best accuracy given those representations. *(ii)* These features were proposed based on intuition and heuristic arguments on why these features are supposed to identify a protocol category. Therefore, The key challenge we address is that protocol fingerprints cannot be automatically generated for traffic identification. Deep learning, a subfield of machine learning, promises to change this by operating on raw input signals and automating the process of feature design and extraction.

Our Research: We present *ProNet*, a deep learning approach to extract protocol fingerprints and perform PI task. Our goal is to directly work on raw traffic payload inputs, which does not require the syntactical understanding of the protocol message, such as field boundaries and delimiters. Specifically, *ProNet* first directly models raw traffic payload data with *byte-level* and *ngram-level* embeddings; then automatically learns fingerprints (signatures/features) by convolution and pool operations on these embeddings; last *ProNet* resorts to a dense softmax neural network to assign a protocol class to every traffic payload.

ProNet has several advantages over classical DPI and DFI fingerprints (signatures/features):

– It considers the process of protocol fingerprinting as part of a holistic optimization objective of protocol identification task; (Typically PF is based on local optimization, *i.e*, protocol feature extraction and identification are two separate parts.)
– It automatically derives protocol fingerprints i.e., no cumbersome and tedious reverse engineering is required;

– The fingerprints learned from traffic payloads are fine-grained comparing with DFI statistical features; (This is helpful for improving classification accuracy.)

Our results show that this approach can achieve the desired performance results, which proves that deep learning is a novel applicable technique to protocol fingerprinting that does not require the explicit definition of the features. Through experiments over a real network trace for protocol identification task, we obtain over 99% precision and recall with low false-positives (less than 1%) and nearly no false-negatives. The rest of this paper is organized as follows: Sect. 3 describes the design and implementation of our proposal. In Sect. 4, we provide experimental results to evaluate the designed approache. Section 2 reviews related work. Finally, conclusions are drawn in Sect. 5.

2 Related Work

Several network security and management tasks require automatic PF technology or fine-grained traffic classification. Here we review selected work related to payload-based PF. Park *et al.* proposed LASER [8] approach which tries to find the longest common subsequence (LCS) among given packet payloads under three constraints: number of packets per flow, minimum substring length and packet size. The LASER is susceptive to the noises and the comparison of the order in the packet payload samples. Ye *et al.* presented a system with a certain tolerance for noises, AutoSig [11], which extracts multiple common substring sequences from sample flows as protocol fingerprints. Some ngram-based proposals [2,9,10,12] try to extract fields by parsing keywords by utilizing a series of stages such as payload segmentation, payload clustering, payload sequence alignment. For example, keywords such as "GET" and "POST" in HTTP have specific delimiters or a similar interval. Notably, keyword parsing methods from the kind of textual protocol analysis, and clearly ineffective for binary protocols since they ignore the fine-grained fingerprints (less than one byte and even multiple bytes). This situation calls for a new paradigm using statistical features to PF problem.

In contrast, the statistical fingerprint methods usually employ machine learning algorithms on the protocol fingerprints (a feature vector with real values). Haffner *et al.* provided the first statistical fingerprint based work called ACAS [4] that explores automatically extracting protocol signatures from packet payload content. And they directly take the values of the first payload content bytes as features for three machine learning classifiers. Different from the ACAS, Finamore *et al.* proposed a system named KISS [3] for UDP traffic, which first extracted the randomness of the first payload bytes, in form of a 24-dimensional feature vector, by using Chi-Square test, then created and evaluated geometric-distance-based and SVM-based traffic classifier in a conventional manner. The KISS leverages the observation that application-layer protocols running over UDP must carry a protocol-specific information header. Afterwards, they extended this idea to support TCP traffic classification [6]. Alas, the limitation of KISS is its high dependency on the number of packet payloads for the online PI task. A more detailed survey of PI and PF studies may be found in [7,9].

3 ProNet

In this section, we describe how we architected our neural network to operate directly on raw payload input, and the intuition behind our decisions. Figure 1 gives an intuitive overview of our approach, showing that our neural network is divided into three notional components: *Byte/Ngram embedding, feature detection, and an identifier*. It is important to note, however, that the entire model is simultaneously optimized, end-to-end, and thus all components are optimized for the singular classification task. The *byte embedding* component embeds the alphabet of single-byte values into a multi-dimensional feature space; Additionally, the *ngram embedding* component embeds the alphabet of ngrams into a multi-dimensional feature space; Thus encoding an input payload's sequence of bytes/ngrams as two two-dimensional tensors from different views. Using the two tensors, the *feature detection* component respectively detects important local sequence patterns within the full byte sequence and ngram sequence, and then aggregates this information into a fixed-length feature vector. Finally, the *identification* component classifies the detected features using a dense neural network. All of these components are optimized jointly using stochastic gradient descent. After we construct a neural network model for the given target protocols from *ProNet*, we can use it to classify traffic.

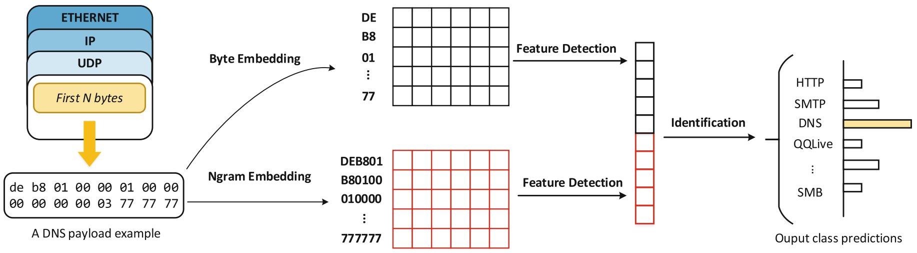


Fig. 1. The architecture of ProNet

3.1 Embedding: A Multi-view Representation of Payload

The input of *ProNet* includes multi-view feature maps of a considered packet payload, each is a matrix initialized by a different embedding version. We use two different payload chunkings and the corresponding embeddings (i.e., views), *Byte Embedding* and *Ngram Embedding*. Let s be payload length, m_s dimension of byte embeddings, m_g dimension of n-gram embeddings. Hence, the whole initialized input of ProNet consists of a two-dimensional array of size $s \times m_s$ and a two-dimensional array of size $(s - n + 1) \times m_g$. In implementation, payloads in a mini-batch will be padded to the same length, and unknown chunks for the corresponding channel are randomly initialized or can acquire good initialization from the mutual-learning phase described in next section. Multi-view initialization brings two advantages: *(i)* a payload (different payload chunking tactics) can have multiple representations in the beginning (instead of only one), which means it has more available information to leverage; *(ii)* different granularity chunk captures are helpful for handling the variant-length message fields, which enhance the robustness of *ProNet* targeting textual and binary protocols.

View 1: Byte Embedding. The first input view of our model is the raw length s sequence of bytes and embeds them into an $s \times m_s$ floating point matrix. This operation is a simple dictionary lookup, where each byte, irrespective of the bytes that came before it or after it, is mapped to its corresponding vector, and then these vectors are concatenated into this matrix. The matrix's rows represent the sequence of bytes in the original payload, and the matrix's columns represent the dimensions of the embedding space.

Formally, we denote $x_i \in \mathbb{R}^{m_s}$ as the m_s-dimensional byte representation for the i-th byte in a payload. A payload with s bytes is denoted as

$$X_{1:s} = x_1 \oplus x_2 \oplus \cdots \oplus x_s \tag{1}$$

where $\oplus$ is the concatenation operator. By this, each input payload (a sequence of bytes) is represented as a $s \times m_s$ matrix. In practice, short payload sequences are padded with zeros to the same length, such that, each matrix shares the same size. In our implementation we set $s = 16$ and $m_s = 50$.

View 2: Ngram Embedding. The another input view of *ProNet* is the length g sequence of ngrams and embeds them into a $g \times m_g$ floating point matrix. Note that different n will produce different ngram sequences and ngram vocabulary. Hence, each special ngram by varying n value, we will obtain an input view. In our implementation we only use 3-grams and $m_g = 50$ (*i.e.*, 3-gram embedding). Notice that for a given n value and s bytes, we have $g = s - n + 1$.

Formally, we denote $z_j \in \mathbb{R}^{m_g}$ as the m_g-dimensional ngram representation for the j-th ngram in a payload with g ngrams. A payload with g ngrams is denoted as

$$Z_{1:g} = z_1 \oplus z_2 \oplus \cdots \oplus z_g \tag{2}$$

where $\oplus$ is the concatenation operator. By this, each input payload (a sequence of ngrams) is represented as a $g \times m_g$ matrix. In practice, short sentences are padded with zeros to same length, such that, each matrix shares the same size.

3.2 Feature Detection

Once we embed our payload input into an $s \times m_s$ matrix and an $g \times m_g$ matrix, the next step is extracting and aggregating locally detected features. This is done in three stages: ① *ByteConv(t, h) and NgramConv(t, h)* - we detect local features by applying multiple kernel convolutions (t) with h-length; ② *ByteMaxPool and NgramMaxPool* - we aggregate the results across the entire sequence by maximizing the kernels' activations; ③ *AggreateView* - we aggregate the extracted features from two different embedding views by means of convolution and pool operations. Notice that the first two steps are done separately for each $h \in \{3, 4, 5\}$. For example, we empirically set $t = 30$, the three results for each h tower are then concatenated together into a 90 length vector. The t filters in our *ProNet* spans the entire length of the byte embedding m_s, and can be thought of as sliding of convolution kernels (or masks) over the sequence of byte embeddings. More details about these feature learning operations as follows.

ByteConv(t, h): A convolution filter $w \in \mathbb{R}^{h \times m_s}$, which is applied to a window of h bytes of m_s-dimensional embeddings, produces a new feature. For instance, given a window of bytes $X_{i:i+h-1}$ and a bias term $b \in \mathbb{R}$, a new feature c_i is generated by

$$c_i = f(w \cdot X_{i:i+h-1} + b), \tag{3}$$

where f is a non-linear function. In our case, we apply the element-wise function Rectified Linear Unit (ReLU) to the input matrices:

$$ReLU(x) = \begin{cases} x, & if \quad x > 0 \\ 0, & otherwise \end{cases} \tag{4}$$

Each filter produces a feature map $c = [c_1, c_2, \cdots, c_{s-h+1}]$ from every possible window $X_{1:h}, X_{2:h+1}, \cdots, X_{s-h+1:s}$ of a sentence of length s.

ByteMaxPool(t): There are several pooling (sub-sampling) methods, such as average pooling, median pooling and max pooling. In this case, we apply max pooling over each feature map produced by the convolution layer and take the maximum element $\hat{c} = \max\{c\}$. Let's denote features generated by this max pooling layer as

$$\hat{c} = \hat{c_1} \oplus \hat{c_2} \oplus \cdots \oplus \hat{c_t} \tag{5}$$

where t is the number of feature maps.

NgramConv($\cdot$) and NgramMaxPool($\cdot$). For ngram embedding, we also execute the similar two operations to extract and aggregate locally detected features. Formally, for a given payload p, we will obtain the convolved and pooled features

$$\hat{d} = \hat{d_1} \oplus \hat{d_2} \oplus \cdots \oplus \hat{d_t}. \tag{6}$$

After convolution and pools operations, we focus on the problem of learning to aggregate multiple views in order to synthesize the information from all views into a single payload features. In our implementation, we simply concatenate the extracted features from two different embedding versions.

AggreateView: In this case, we simply concatenate the extracted features from two different embedding views. Let's denote features generated by this view pooling layer as

$$\hat{e} = \hat{c} \oplus \hat{d}. \tag{7}$$

3.3 Identification Algorithm

Once we extract the features from packet payload, we use a standard dense neural network to predict the protocol class of the packet. The neural network only has one layers with $l = 10$ units (10 classes). Formally, given $\hat{e}$ as the input, the fully connected layer produces

$$P(Y = j|\hat{e}, \theta) = softmax_j(W \cdot (\hat{e} \circ r) + b), \tag{8}$$

where Y is the prediction, θ denotes parameters $\{W, b\}$, W denotes weights, $\circ$ denotes the element-wise multiplication operator and $r \in \mathbb{R}$ is a dropout mask vector of Bernoulli variables with probability ρ of being zero. The fully softmax layer output provides the probability that the input payload belongs to the class label given the convolution based features. We measure our detector's prediction loss using cross entropy,

$$L(\hat{y}, y) = -\frac{1}{N} \sum_{i}^{N} [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] \tag{9}$$

where $\hat{y}$ is our model's prediction probability vector for all the packet samples and y is the vector of true label. We use Adam [5] method to minimize Eq. 9.

4 Experiments

4.1 Data Set

We collect real-world IP traffic traces from a edge router of a local area network (LAN) in mainland China. We evaluate *ProNet* on ten typical application-layer protocols: DNS, SopCast, PPStream, NBNS, DHCP, QQLive, PPLive, HTTP, SMTP and SMB. Notice that the ten target protocols involve both connection oriented protocols (such as TCP) and connection less protocols (such as UDP). Simultaneously, the ten target protocols also contain both textual and binary protocols. Note that DNS, SMTP, HTTP and SMB are representatives of well-known protocols, and they are both textual and connection-oriented protocols. QQLive, PPLive and PPStream are popular peer-to-peer video streaming protocols, and they are representatives of proprietary and new protocols. All of the above protocols offer a great amount of Internet traffic. Our evaluations only consider the first 16 bytes of the payload of an IP packet, which suffice for traffic classification. We do not examine the remaining payload and ensure that the privacy is preserved and for early flow identification. In this work, we obtain ground-truth by combining port-based method and a rather frequently used open tool nDPI [1].

4.2 Performance Metrics

To quantitatively evaluate the effectiveness of our proposal: given a packet trace of one application protocol, we are interested in the following three metrics.

– *Precision:* The Precision is the ratio of the number of true positives to the total number of packets that identified as the target protocol (including true positives and false positives) by *ProNet*.
– *Recall:* The Recall is the ratio of the number of true positives to the total number of packets that actually belong to the target protocol (including true positives and false negatives).
– *F-Measure:* The F-measure is a compromise between recall and precision. It is defined as $2 * \frac{Precision * Recall}{Precison + Recall}$.

4.3 Parameter Configuration

In this section, we'll discuss how to select parameters in our approach. In fact, many different combinations of parameters can give similarly good results. We choose some parameters empirically in our experiment. Before training, we randomize the weights using Gaussian distributions, in which the mean and standard deviation is (0, 0.05). In each convolution layer, the stride is set to 1. In max-pooling layers, the stride is equal to the number of units, as the units are non-overlapping. To speed up model training and prevent overfitting, we use layer-wise batch normalization and dropout (0.5) between layers. The main parameters that we tune in our work are as follows:

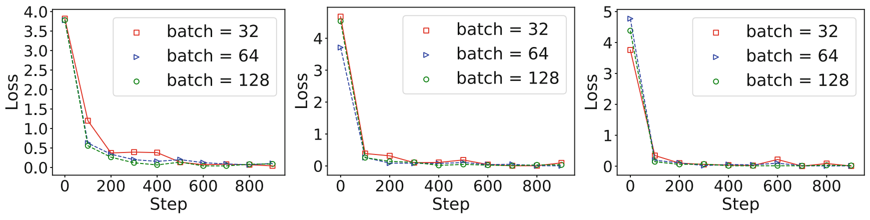
The embedding size (m): Recall that in Sect. 3, we introduced two embedding strategies for descripting payload. For a given payload example, we converted it into a $s \times m_s$ matrix, and a $s - n + 1 \times m_g$ matrix. The byte embedding size (m_s) and the ngram embedding size (m_g). For parameter simplify, we empirically set $m_s = m_g = m = 30$, $s = 16$.

The number of convolution filters (t): We use filter windows (h) of 3, 4, 5 with t feature maps each. The number of filters impact the ability of feature detection of *ProNet*. In the following evaluations, we carry out our experiments for $t \in \{10, 30, 50\}$.

Mini-batch size of SGD (batch): SGD is used in the *ProNet* model to optimize the loss function. We choose $batch \in \{32, 64, 128\}$.

Steps in *ProNet* training phase (step): In one step, the *ProNet* will update the weights once. We need to select the appropriate number of steps, so that the *ProNet* model can converge. The loss function is used to evaluate whether it is convergence or not.

Figure 2 shows the results of selecting above parameters for PI-task. We can observe that the training loss generally converges after 500 steps. The bigger the batch size the faster the loss can converge, and the value of loss is also smaller. Also, when the batch size is set to a larger value, the loss declines more smoothly.



(a) t=10 (b) t=30 (c) t=50

Fig. 2. Parameter tuning with embedding size $m = 50$

4.4 Performance Results

We conducted five separate 5-fold cross-validation experiments, where for each experiment we randomly split our data into five equally sized partitions. For each of the five partitions, we trained against four partitions and tested against the fifth. The results of performance metrics are shown in Table 1. We can observe that our approach can achieve above 99% in precision and recall.

Our experiments were carried out on a single machine with 8 G RAM and a quad-core Intel processor of 3.6 GHz. In the training phase, it takes about 3 min to train 2000 steps for one protocol class with about 200K samples. The larger the data set the longer the time it takes to train. In the evaluation phase, it takes about 5 ms to predict on a packet on average.

Table 1. The results of 5-folds cross validation of *ProNet* on PI-task

	Precision	Recall	F-Measure
DNS	0.9989 ± 0.00084	1.0 ± 0.0	0.9994 ± 0.00042
SopCast	0.9961 ± 0.00209	0.9959 ± 0.00142	0.9961 ± 0.00090
PPStream	0.9976 ± 0.00065	0.9828 ± 0.00116	0.9902 ± 0.00063
NBNS	0.9971 ± 0.00158	0.9995 ± 0.00037	0.9983 ± 0.00083
DHCP	1.0 ± 0.0	0.9979 ± 0.00030	0.9989 ± 0.00015
QQLive	0.9898 ± 0.00185	0.9993 ± 0.00043	0.9945 ± 0.00080
PPLive	1.0 ± 0.0	0.9930 ± 0.00108	0.9964 ± 0.00054
HTTP	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0
SMTP	0.9936 ± 0.00161	0.9979 ± 0.00065	0.9957 ± 0.00065
SMB	0.9998 ± 0.00024	0.9987 ± 0.00127	0.9992 ± 0.00070

5 Conclusion

We present *ProNet*, a robust automatic protocol fingerprint learning approach that can operate on raw packet payloads, for accurately traffic identification.

ProNet leverages a convolutional neural network for extracting features from packet payloads as the key insight, and characterizes traffic payloads via byte embeddings and ngram embeddings. Using embeddings with convolutions as top layers in our neural network coupled with supervised training, allows us to implicitly extract a feature set that is directly optimized for classification. Extensive experimental evaluations show that *ProNet* achieves high accuracy and remains robust in the real trace. *ProNet* can serve as a critical preprocessing tool for many traffic classification applications in quality-of-service control, network security, and resource profiling. We believe that *ProNet* is the first approach that demonstrates how top to bottom deep-learning method can be adapted to the traffic identification problem, where payload data are purposely obfuscated to prevent obvious feature extraction.

Acknowledgment. This work was supported by the National Natural Science Foundation of China (No. 61572496), and National Key Research and Development Program of China (No. 2016YFB0801502).

References

1. Open and extensible lgplv3 deep packet inspection library. http://www.ntop.org/products/deep-packet-inspection/ndpi/
2. Cui, W., Kannan, J., Wang, H.J.: Discoverer: automatic protocol reverse engineering from network traces. In: USENIX Security, vol. 158 (2007)
3. Finamore, A., Mellia, M., Meo, M., Rossi, D.: KISS: stochastic packet inspection classifier for UDP traffic. IEEE/ACM Trans. Netw. **18**(5), 1505–1515 (2010)
4. Haffner, P., Sen, S., Spatscheck, O., Wang, D.: ACAS: automated construction of application signatures. In: Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data, MineNet 2005, pp. 197–202. ACM, New York (2005). http://doi.acm.org/10.1145/1080173.1080183
5. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
6. La Mantia, G., Rossi, D., Finamore, A., Mellia, M., Meo, M.: Stochastic packet inspection for TCP traffic. In: 2010 IEEE International Conference on Communications (ICC), pp. 1–6, May 2010
7. Nguyen, T.T., Armitage, G.: A survey of techniques for internet traffic classification using machine learning. IEEE Commun. Surv. Tutor. **10**(4), 56–76 (2008)
8. Park, B.C., Won, Y., Kim, M.S., Hong, J.: Towards automated application signature generation for traffic identification. In: Network Operations and Management Symposium, NOMS 2008, pp. 160–167. IEEE, April 2008
9. Sang, Y., Li, S., Zhang, Y., Xu, T.: ProDigger: towards robust automatic network protocol fingerprint learning via byte embedding. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 284–291. IEEE (2016)
10. Tongaonkar, A., Keralapura, R., Nucci, A.: SANTaClass: a self adaptive network traffic classification system. In: IFIP Networking Conference, pp. 1–9. IEEE (2013)

11. Ye, M., Xu, K., Wu, J., Po, H.: AutoSig-automatically generating signatures for applications. In: Ninth IEEE International Conference on Computer and Information Technology, CIT 2009, vol. 2, pp. 104–109, October 2009
12. Zhang, Z., Zhang, Z., Lee, P., Liu, Y., Xie, G.: ProWord: an unsupervised approach to protocol feature word extraction. In: 2014 Proceedings IEEE INFOCOM, pp. 1393–1401, April 2014