Received January 10, 2020, accepted February 5, 2020, date of publication February 10, 2020, date of current version February 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2973023

An Unsupervised Deep Learning Model for Early Network Traffic Anomaly Detection

REN-HUNG HWANG[1,2], (Senior Member, IEEE), MIN-CHUN PENG[1], CHIEN-WEI HUANG[1], PO-CHING LIN[1], AND VAN-LINH NGUYEN[1,3], (Member, IEEE)

[1]Department of Computer Science and Information Engineering, National Chung Cheng University (CCU), Chiayi 62102, Taiwan
[2]Advanced Institute of Manufacturing With High-Tech Innovations, National Chung Cheng University (CCU), Chiayi 62102, Taiwan
[3]Department of Information Technology, TNU-University of Information and Communication Technology, Thai Nguyen 24119, Vietnam

Corresponding author: Van-Linh Nguyen (nvlinh@ictu.edu.vn)

This work was supported in part by the Advanced Institute of Manufacturing With High-Tech Innovations (AIM-HI) from the Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan, in part by the Ministry of Science and Technology of Taiwan, China, under Grant MOST 107-2218-E-194-014, Grant 108-2221-E-194-022-MY3, and Grant MOST 108-2221-E-194-019-MY3, in part by the Taiwan Information Security Center, National Sun Yat-sen University, under Grant TWISC@NSYSU, and in part by the TNU-University of Information and Communication Technology, Vietnam.

ABSTRACT Various attacks have emerged as the major threats to the success of a connected world like the Internet of Things (IoT), in which billions of devices interact with each other to facilitate human life. By exploiting the vulnerabilities of cheap and insecure devices such as IP cameras, an attacker can create hundreds of thousands of zombie devices and then launch massive volume attacks to take down any target. For example, in 2016, a record large-scale DDoS attack launched by millions of Mirai-injected IP cameras and smart printers blocked the accessibility of several high-profile websites. To date, the state-of-the-art defense systems against such attacks rely mostly on pre-defined features extracted from the entire flows or signatures. The feature definitions are manual, and it would be too late to block a malicious flow after extracting the flow features. In this work, we present an effective anomaly traffic detection mechanism, namely D-PACK, which consists of a Convolutional Neural Network (CNN) and an unsupervised deep learning model (e.g., Autoencoder) for auto-profiling the traffic patterns and filtering abnormal traffic. Notably, D-PACK inspects only the first few bytes of the first few packets in each flow for early detection. Our experimental results show that, by examining just the first two packets in each flow, D-PACK still performs with nearly 100% accuracy, while features an extremely low false-positive rate, e.g., 0.83%. The design can inspire the emerging efforts towards online anomaly detection systems that feature reducing the volume of processed packets and blocking malicious flows in time.

INDEX TERMS IoT security, anomaly detection, convolutional neural network, autoendcoder, online DL-based anomaly detection.

I. INTRODUCTION

In recent years, with increasingly massive IoT applications and connected devices, distributed denial-of-service (DDoS) attacks have caught the attention of the security community with a series of record-high attack magnitude. Given a small proportion of billions of IoT devices, e.g., cheap and insecure IP cameras, injected to be zombies, an adversary can generate a massive volume of flooding traffic to take down a target such as a critical Internet service. Although this kind of attack is by no means new, it still poses a tremendous threat to most state-of-the-art defense systems [1]–[4]. To stop malicious traffic, including that from DDoS attacks, the first step is to detect traffic anomaly as soon as possible by analyzing network traffic at the gateways, at edge servers, or in a scrubbing center [5].

To date, existing approaches such as signature-based and statistical detection systems still have several flaws, e.g., the rule maintenance cycle cannot keep up with soaring attack variants [3]. When the ecosystem of Internet-connected systems expands and the diversity of IoT devices increases rapidly, it is inevitable that there are more potential vulnerabilities for an attacker to exploit. As a result, a signature-based detection system, which may be able to

The associate editor coordinating the review of this manuscript and approving it for publication was Fan Zhang.

detect well-known attacks with high accuracy, can quickly lose its advantage because unknown attacks may appear nearly per minute [1], [2]. Dealing with the explosion of the attack variants, the anomaly detection approaches, as opposed to the signature-based ones, can significantly help. Unlike signature-based approaches, anomaly detection systems can monitor network flows and classify them as either normal or anomalous ones; thus, new attack variants are less likely to bypass the detection. Nonetheless, anomaly detection approaches often face high false alarm rates, since the systems must be taught to recognize normal activities [6]. So far, such systems are often designed with strict mathematical models and a set of predefined features [7]. Fortunately, deep learning (DL) promises to be the game-changer to help to solve the learning problem, i.e., automatically building the traffic profile. The most benefit of deep learning is to build a thorough pattern that can precisely characterize specific objects through automatically learning a large volume of data and species.

DL-based approaches have been well investigated in many fields over the years, including anomaly detection. However, many challenges remain, e.g., speeding up the detection and auto-profiling the traffic patterns effectively, which are also the target of this work. From the design perspective, the detection systems should characterize normal network flows and define well-represented traffic profiling. Based on this profiling, the systems can identify and isolate anomalous network activities. In the literature, the common profiling method is building a pre-defined list of features [6], [8] from flow statistics, e.g., sending rate, packet count or flow size, and then using the DL models such as a convolutional neural network (CNN) for learning [9]. However, defining a list of well-represented features manually for effective learning poses tremendous challenges, e.g., labor time, particularly if the network has a diversity of application traffic. Recently, a promising approach is to use CNN to automatically extract such features directly from raw traffic, instead of from the summarized data, e.g., [10]. In this work, we go further in building the traffic patterns (e.g., of benign applications) *by examining only the first few bytes of the first few packets of the flows*. This approach promises to have many advantages, particularly for online anomaly detection systems. For example, the detection does not need to waste remarkable computation and time for checking redundant data and storage in a whole long session, while a few first packets of the flows are sufficient for the detection. As a result, our system has a significant advantage of speeding up the detection. Note that summarizing the traffic in a flow-based approach may demand much memory space for flow tracking in a large network, particularly if many long flows exist.

The proposed system, namely D-PACK, consists of two main parts: (1) A CNN module is designed for auto-learning the features from the raw data; (2) An unsupervised DL model (autoencoder) trained with the output data of (1) targets at building the profile of benign traffic and then precisely judge whether the traffic in the examined flows is abnormal. The experimental results show that D-PACK is competitive and prominently outperforms prior studies in terms of accuracy, precision, recall, and F1-measure. Specifically, it can detect malicious traffic with nearly 100% accuracy and less than 1% FNR and FPR, even if it examines only two packets from each flow and 80 bytes from each packet. To train the system with the normal traffic characteristics and activities, the training is set to run at the time of deploying the devices to ensure the devices are in the clean state before any possible compromising. The detection is also deployed close to the devices (i.e., the traffic sources) to identify traffic anomaly.

In summary, the main contributions of this work are as follows:

- We propose a CNN-based deep learning approach for auto-learning the traffic features and profiling traffic directly from the raw traffic with only a few first packets per flow. Following this, the auto-learning approach can significantly save the efforts to build traffic patterns for a complex network where the diversity of application traffic is the major challenge to conventional methods.
- We implement the proposal and evaluate it with both the credible datasets and self-collected realistic ones. The evaluation with the datasets from multiple sources shows that D-PACK can achieve nearly 100% accuracy and precision in detecting malicious packets.
- Our design on packet-based deep learning classification and detection promise to provide valuable information and inspire the research community to overcome the remaining challenges, particularly for speeding up online DL-based anomaly detection.

The remainder of this paper is organized as follows. Section II surveys the state-of-the-art work on traffic classification and deep learning detection approaches. Section III presents our learning strategy and the detailed structure of the auto-building traffic profile module, the key features of this work. Our selected datasets and the detail of the D-PACK framework are presented in Section IV. We show the experimental design and results in Section V. Finally, the conclusion and future work are summarized in Section VI.

II. RELATED WORK

The issue of detecting malicious network traffic has been studied extensively. The mainstream approaches fall into four primary classes [8]: port-based/rule-based, deep/stochastic packet inspection, statistical, and behavioral techniques. While the first three approaches are common and gain high performance in intrusion detection over decades, it is expected to see a silver lining with the rising trend of deep learning to overcome the known flaws of existing studies, particularly in the last approach. From the evaluation perspective, the behavioral approach may have poor performance at several metrics, e.g., high false alarm, due to the complexity of benign traffic profiling and the definition of 'normal' network activities. With the potential of accumulating knowledge from large-scale raw data without manual interference, deep learning is an effective approach to improve anomaly

TABLE 1. Summary of related anomaly detection strategies.

Author	Category	DL method	Features	Dataset	Performance	Year
E. Min [12]	Intrusion detection	Autoencoder	Statistical	NSL-KDD CICID 2017	0.99 DR	2018
E. Min [9]	Intrusion detection	Word embedding, CNN RF	Statistical+payload	ISCX 2012	0.99 DR	2018
Y. Meidan [13]	Intrusion detection	Autoencoder	Statistical	Self-collected	100% TPR 0.007±0.01 FPR	2018
W. Wang [14]	Intrusion detection	CNN+LSTM	Header+payload	DARPA 1998 ISCX 2012 FPR	0.99 DR	2018
G. Aceto [15]	Traffic classification	SAE,CNN,LSTM MLP,RF	Header+payload	Microsoft	0.93 DR	2019
M. Al-Qatf [16]	Intrusion detection	Autoencoder+SVM	Statistical	KDD99	0.95 DR	2018
L. Vu [17]	Traffic identification	AC-GAN	Statistical	Self-collected	0.99 DR	2017
W. Wang [18]	Traffic identification	CNN	Header+payload	ISCX VPN-nonVPN	0.86 DR	2017
Z. Chen [19]	APP/protocol identification	RKHS+CNN	Statistical	Self-collected	0.88 DR	2017
M. Lotfollahi [20]	APP/traffic identification	CNN/SAE	Header+payload	ISCX VPN/non-VPN	0.98 Precision	2019
M. L.Martin [21]	Mixed-type classification	CNN+LSTM	Header+time	RedIRIS	0.99 DR	2019
J. Hochst [22]	Traffic identification	Autoencoder	Statistical+header	Self-collected	0.8 Precision	2017
I. Arnaldo [23]	Intrusion detection	CNN, LSTM RNN, RF, PCA	System log	Self-collected	0.94 AUROC	2017
Y. Yu [24]	Intrusion detection	Dilated convolution, autoencoder	Header+payload	CTU-UNB, ISCX-IDS 2012	0.98 DR	2017
Y. Li [25]	Malicious code detection	Autoencoder+DBN	Statistical	KDD CUP'99	0.92 DR	2015

- DR(Detection Accuracy), TPR (True-Positive Rate), FPR(False-Positive Rate), AUROC (Area Under the Receiver Operating Characteristics)
- CNN(Convolutional Neural Network), LSTM (Long-Short Term Memory), SAE (Stacked auto-encoders), DBN(Deep Belief Networks)
- MLP(Multilayer Perceptron), RKHS(Reproducing Kernel Hilbert Space), RNN(Recurrent Neural Networks)
- AC-GAN(Auxiliary Classifier Generative Adversarial Networks), PCA(Principal Component Analysis), RF(Random Forest)

detection in the future [11]. In Table 1, we summarize several well-known state-of-the-art DL-based traffic classification methods, their performance on common datasets, and then address our research position.

As shown in Table 1, most works are in favor of using CNN/autoencoder to build the traffic classifier for anomaly detection. Notably, these approaches rely on a set of predefined features, e.g., statistical features [9], [16], [25], for data preprocessing. However, the DL techniques in these systems are supposed to vary due to the different interests in addressing specific applications and defense strategies [15], [21]. For example, the authors in [18], [20] select CNN for extracting the features from the packet headers and payloads. Autoencoder, on the other hand, is proper for training on the extracted features [12], [16], [22]. Also, the bias and imbalance of the traffic classes in the training datasets are a key factor to motivate the authors to select a proper DL model. Finally, while Table 1 reflects no quantitative measurement of which method is better due to *their different usages in the datasets and further different metrics for evaluation*, our summary here aims to give an overview of the dominant trends of applying DL for anomaly detection.

Unlike prior research, we pursue a novel approach to detect traffic anomaly. Instead of collecting the entire traffic flows for finding the evidence of abnormal behavior, we pack *the first several packets per flow into segments of fixed lengths*, e.g., 80 bytes. The goal is to build a subset of the data and train only on this trimmed version. This approach significantly reduces the computation and memory space to process the session flows, particularly the long ones. A closely related work to ours is [13], in which the authors first collected both normal and malicious traffic from an IoT environment with devices infected by Bashlite and Mirai. The traffic is trimmed by various time scales for experimental purposes, including 100ms, 500ms, 1.5sec, 10sec, and 1min time windows. For each time window, a set of pre-defined 23 flow features are extracted as the input to the autoencoder. However, the method still consumes quite much time since the sampling is performed on the offline data and extracting the features is an independent pre-processing step. In contrast, in this work, we build a CNN model for auto-learning the features and the sampling targets at just several packets per flow (instead of sampling the whole flows at several check time points). Compared with prior studies, this work features two major differences: (1) D-PACK can build the profile of the traffic by examining the first packets per flow, instead of checking the total packets in the flows; (2) D-PACK can work directly with raw packets (after data sanitization), i.e., building the patterns, reading the input data and making the detection decision. Finally, we believe that a detection approach by trimming the data for inspection like ours is competitive with the high detection accuracy achieved by prior studies for

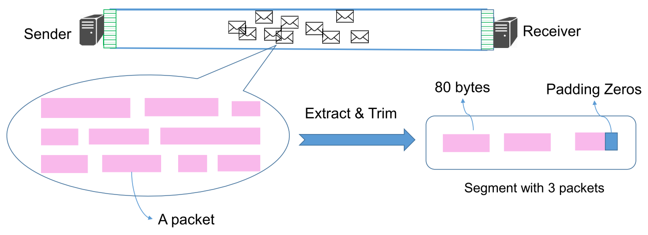

FIGURE 1. Illustration of the sampling to extract the segment of the first 3 packets from a flow of 10 packets.

the same attack types. This approach can motivate further research on optimizing and accelerating the processing in early anomaly detection systems.

III. LEARNING STRATEGY FOR EARLY CLASSIFICATION AND DETECTION

This section gives an overview of the learning strategy that targets at building a thorough traffic profile while dramatically reduces the data volume for processing. This mechanism also makes early anomaly detection possible, which is the key feature of this work.

A. SAMPLING NETWORK FLOWS

The traffic volume can be enormous in a high-speed network; thus, it is important to reduce a load of packet capturing and analysis on the detection function for high efficiency. According to the heavy-tailed nature of Internet traffic [26], it was reported that keeping only a small portion of each flow is sufficient for protocol identification or retrospective analysis while reducing the total traffic volume significantly [27], [28]. Therefore, we consider sampling the flows *by extracting the first n packets of each*, and each packet is trimmed into a fixed length of l bytes, starting with the header fields (with zero-padding if necessary; see the next subsection for the detail). Note that the raw packets of a flow is classified based on their 5-tuple information (source IP, source port, destination IP, destination port, and transport layer protocol). n and l may be flexibly adjusted according to the network traffic characteristics at the time of deployment. This sampling can significantly reduce the total amount of traffic in the analysis, while the characteristics of the packets are still representative to reveal whether the associated flows are malicious or not for early anomaly detection. Our sampling workflow is illustrated as Fig. 1.

As an example in Fig. 1, a flow consists of ten packets, and we inspect only the first three packets and trim their lengths. Since only several packets per flow are examined, the system can inspect significantly fewer data in total, like the results from in [27], [28]. This saving is even much more if the flow is part of a long session. Notably, this saving is essential for a system that requires early detection.

After the data extraction (from pcap) with trimming, the next step is to read the data and form an understandable format for the learning task. Since a DL system cannot be fed with data of different lengths, short packets are padded with zeros. We view the trimmed bytes in each flow as a one-dimensional vector with $n \times l$ elements, and turn them into the input of the CNN model for training. Like major DL studies, a step of trace sanitization is also used to eliminate the errors and repeated traffic.

B. AUTO-BUILDING THE TRAFFIC PROFILE

Building the pattern of benign traffic is the next important task. Compared with traditional methods that extract traffic features manually, an auto-learning approach does not contain independent modules such as feature extraction and feature selection. The features are automatically learned, and the traffic is directly passed to the classifier. Therefore, the non-linear relationship between the raw input and the expected output is determined, partially achieving the goal of end-to-end learning.

To date, CNN has been mainly applied in the domain of computer vision, e.g., image classification [29]. Recently, there are also successful applications in the field of natural language processing (NLP) [7], [11]. CNN is most suitable for the kinds of data in the form of multiple arrays or the ones with strong local correlations or whose features can appear anywhere, even in which objects are invariant to translations and distortions [19]. Specifically, 1D-CNN is good for data like sequential data or language [7], [11]. 2D-CNN is good for data like images. 3D-CNN is good for data like video or volumetric images. 1D-CNN is also widely used for network traffic analysis, e.g., [19], including this work.

It is common to visualize the extracted bytes from each flow as the pixels of two-dimensional (grayscale) images like those in Fig. 2, and then to apply 2D-CNN to traffic classification. The authors [24] use this approach, although they prefer to use a stacked auto-encoder (SAE) other than 2D-CNN. However, since a flow may consist of more than one packet, a 2D-CNN filter may cover a region of irrelevant bytes from two packets or more (e.g., the bytes in adjacent rows of pixels are semantically irrelevant). Inspired by this fact, our auto-profiling module is built on a 1D-CNN model (i.e., the input to CNN is a one-dimensional image). A similar approach was taken in [18], but this work differs from theirs: we set a fixed number of packets and bytes taken from a flow to form the one-dimensional image, regardless of whether the bytes are from one or multiple packets.

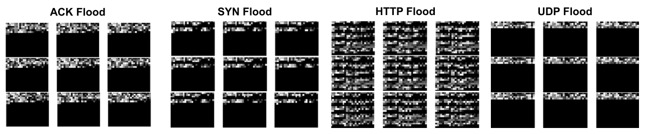

FIGURE 2. Visualization of the traffic types in our dataset (Mirai-CCU). The detail will be described in section IV.

The architecture of the 1D-CNN model is illustrated as in Fig. 3. We assume that the first n packets per flow are sampled, and each sampled packet is trimmed into the first l bytes (primarily in the packet header). The data are padded

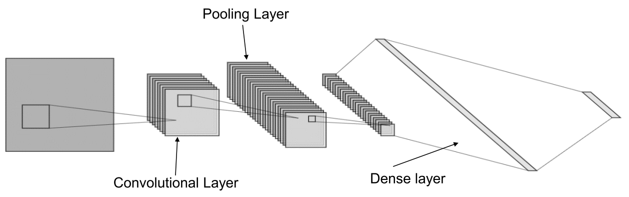

FIGURE 3. CNN model for auto-building the traffic profile.

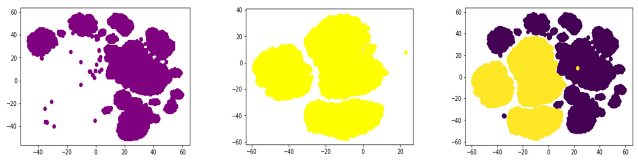

FIGURE 4. Packet distribution after binary classification, from left to right: the normal traffic distribution, malicious traffic distribution and common distribution.

with zeros if the number of packets is less than n or the packet size is less than l bytes. Let p_i be the ith packet after the above trimming. The one-dimensional image is composed of the bytes in $p_1||p_2||\dots||p_n$ as the input, where $||$ is the concatenation operator.

Let x_i be a window of s bytes beginning from the ith byte of the input, where $i = 1, \dots, n \times l - s + 1$. In the convolution operations, let m be the number of filters to be applied to the input x_i to produce a new feature. As a result, a feature h_i^k from the kth filter to the window x_i is generated by

$$h_i^k = f(W^k x_i + b_k), \tag{1}$$

where f is a ReLU function [18], W^k is the weight vector of the kth filter, and b_k is a bias term or the offset of the kth filter. The above parameters in this work will be summarized in Table 5. Note that each filter is applied to the possible windows of s bytes in the input to produce a feature map (assuming the stride is 1 in Eq. 2):

$$h = [h_1, h_2, \dots, h_{nl-s+1}]. \tag{2}$$

We then apply a max-over-time pooling operation over the feature map and take the maximum value max(h) as the feature in the next layer. In this work, we use multiple convolution layers and pooling layers to extract high-level features. These features form the layer and are passed to a fully connected dense layer whose output is the probability distribution of the type of benign flows (e.g., Gmail, Skype).

It is noted that increasing the parameter n may offer more statistical characteristics about the flow, but pours more data into the system. The evaluation in Section V reveals clearly this relationship. In this work, we attempt to use as few packets as possible while still guarantee detection accuracy. Therefore, the ideal configuration of n and l is the threshold in which the system performance is balanced among several measurements, e.g., high accuracy and low traffic volume for processing. We found in the empirical experiments that, with $n = 2$ and $l = 80$, the system can attain this target on most of the selected datasets (see Section V).

In this work, we run the model on benign traffic to build the traffic profile for the binary classifier, i.e., to classify whether the flows are from benign traffic or not. In this case, autoencoder is used as the binary classifier. An example of the distribution of benign/malicious packets after binary classification from our autoencoder is illustrated in Fig. 4. Notably, the hidden layers of the learning module are designed to connect directly with the input layers of the autoencoder network (see Section IV-B). This connection significantly helps to achieve the goal of end-to-end learning, since the learning and classification are under a connected and unified DL network.

In summary, besides the auto-learning ability of the CNN model on the raw packets, the learning strategy, along with the sampling procedure to shorten the data for processing, plays a decisive role in computation reduction and early detection. Moreover, with the adjustable parameters, n and l, the administrator can adjust the proper thresholds to balance learning from as many features of the traffic as possible and satisfying the limited system resources or the detection speed. The effectiveness of this approach is revealed in Section V.

IV. AN UNSUPERVISED DEEP LEARNING MODEL FOR EARLY ANOMALY DETECTION

In this section, we first cover the detail of our selected datasets and then the architecture of the D-PACK detection framework. The strategy for tuning the parameters to improve the performance is also described.

A. DATASET

Lack of a variety of shareable traces dataset has hindered the progress in traffic classification by deep learning. Researchers may create synthetic datasets through their testbed, but the generated traffic may not reflect the Internet traffic faithfully. Thus, the evaluation based on such traces may not be credible. Over the decades, researchers have been tempted to use famous public traffic datasets such as KDD CUP 99 and NSL-KDD. The datasets provide useful statistics on labeled features and both benign and malicious flows as well, but they do not provide information at the raw packet level, which is required in our approach. Also, while the credible dataset from Microsoft Malware Classification Challenge [30] can provide a metadata manifest and hexadecimal representation of the binary content of malware, missing packet information in the data, unfortunately, makes it unusable in this work.

USTC-TFC2016 [18] is a prominent dataset that matches our requirements. Table 2 summarizes the statistics of benign and malicious traffic in the dataset. Per their statement, totally ten types of malware traffic were collected from public websites in a real network environment from 2011 to 2015. Along with such malicious traffic, the benign part contains ten types of normal traffic collected using IXIA BPS, a professional

TABLE 2. Summary of benign and malware traffic in USTC-TFC2016 dataset.

Traffic type	Total # of flows	Flow type	# of flows
Benign	**309,887**	Facetime	6,000
		Skype	6,321
		Bittorrent	7,517
		Gmail	8,629
		Outlook	7,524
		WarCraf	7,883
		MySQL	86,089
		FTP	101,037
		SMB	38,937
		Weibo	39,950
Malware	**624,414**	Tinba	8,504
		Zeus	10,970
		Shifu	9,634
		Neris	33,791
		Cridex	461,548
		Nsisay	6,069
		Geodo	40,947
		Miuref	13,481
		Virut	33,103
		Htbot	6,367

network traffic simulation equipment. The size of this dataset is 3.71GB in the pcap format.

For the Mirai-based DDoS traffic, we use the dataset from Robert Gordon University, denoted by Mirai-RGU, and its statistics are shown in Table 3. The readers are referred to [31] for the testbed used to collect the dataset. This dataset contains Mirai botnet traffic such as scan, infect, control, attack traffic, and normal IP camera traffic. The Mirai botnet traffic consists of four main attack types: UDP flood, ACK flood, DNS flood, and SYN flood attacks. The dataset includes features such as time, source, destination, protocol, length, and overall payloads.

TABLE 3. Summary of benign and malicious traffic by packet count in the Mirai-RGU dataset [31].

Traffic type	Total # of flows	Flow type	# of flows
Benign	**76,725**	Mixed traffic	76,725
Malware	**2,991,832**	Ack flood	7,425
		HTTP flood	143
		UDP flood	32,418
		DNS flood	4,852
		Mirai	2,795,422
		VSE flood	4,990
		GREIP flood	27,804
		SYN flood	118,754
		UDPPLAIN flood	19
		GREETH flood	5

In order to enrich the traffic activities and identify the attack traffic, we have built a Mirai botnet in the campus of National Chung Cheng University (CCU) for testing (denoted by Mirai-CCU), as shown in Fig. 5. The C&C server controls seven IoT devices (i.e., WiFi access points) and collects all the DDoS attacks from Mirai. Despite the small testbed, we argue that the detection performance in the experiments does not deviate from that in a large testbed because the detection refers only to the first few bytes in the first few packets of the flows as the features, instead of the number of flows, packets

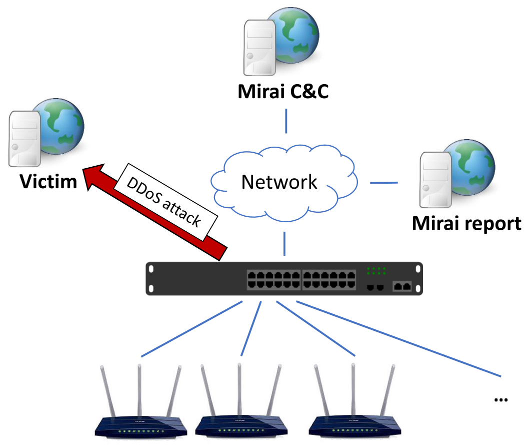

FIGURE 5. The testbed of our Mirai-based DDoS dataset in the campus of National Chung Cheng University (CCU), in which the Mirai botnet is at the WiFi APs in the bottom.

or bytes over a given period of time. It is important to note that reproducing a large-scale Mirai-based botnet would be costly and probably illegal; thus, we do not resort to that alternative.

The collected dataset consists of four DDoS attack types: SYN flood, UDP flood, ACK flood, and HTTP flood. The dataset covers over 304K malicious flows and its size is 277.1GB. Table 4 summarizes the collection of these four types. Like [18], we also visualize these four attack types in Fig. 2. For each type of attack, we randomly select nine flows, and for each flow, a 2D grayscale image of 20*25 is produced by selecting the first 100 bytes of the first five packets (i.e., totally 500 bytes in a flow). Fig. 2 shows that the same type of flows are similar in the visualization, but are quite different between different flows.

TABLE 4. Mirai DDoS types collected in our testbed, namely Mirai-CCU.

Total # of flows	Flow type	# of flows
1,021,145	ACK flood	150,001
	HTTP flood	7,722
	UDP flood	99,986
	SYN flood	763,436

B. UNSUPERVISED DL-BASED ANOMALY DETECTION ARCHITECTURE

Fig. 6 shows the schematic diagram of the D-PACK framework. After the pre-processing (left side), CNN is used to automatically extract the flow features by classifying flows into different types, e.g., 10 types in the USTC-TFC2016 dataset. The network traffic is composed of raw packets. The step of flow generation implies the packets are grouped by 5-tuple. One may question that IP can be either benign or malicious alternatively and then the classification on such data is meaningless. In this case, D-PACK is a flow-based classifier, an IP source with both malicious and benign flows does not affect D-PACK's classification. D-PACK will classify benign flows as benign and malicious flows as malicious no matter what is the source IP.

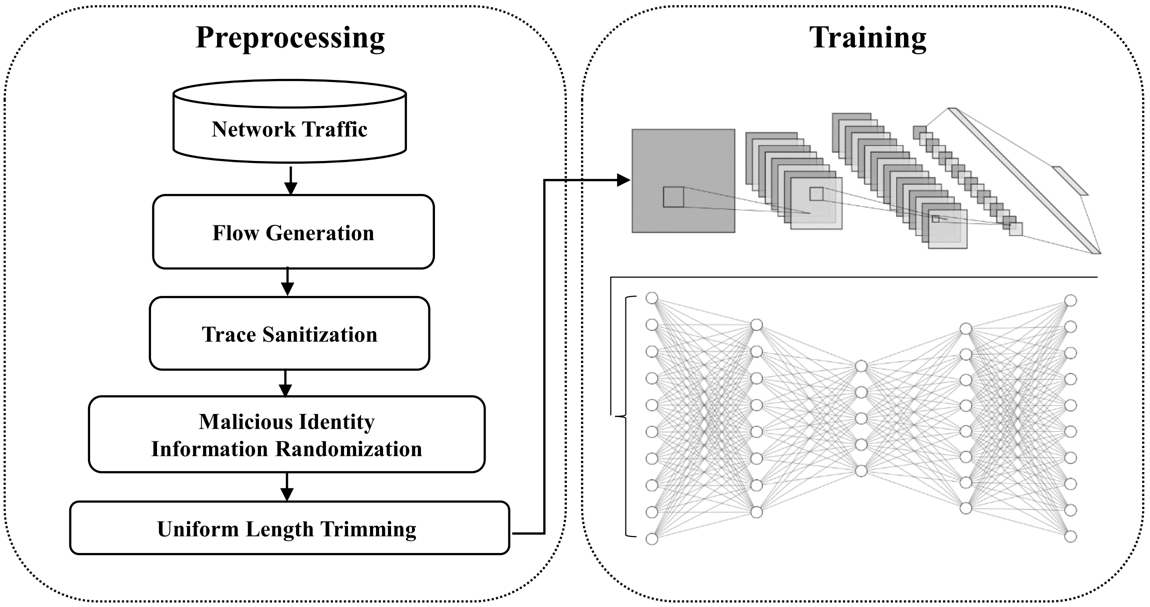

FIGURE 6. The architecture and workflow of D-PACK, including the preprocessing (left), training and auto-learning module (right).

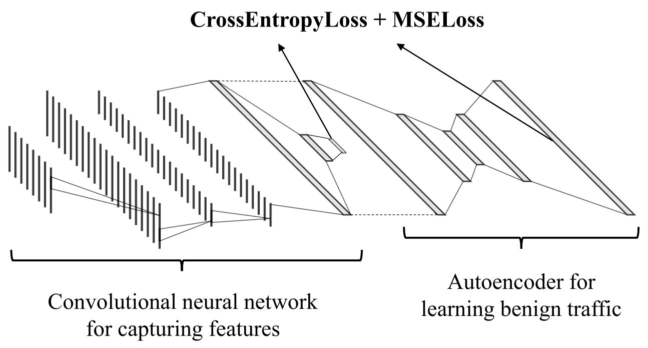
CrossEntropyLoss + MSELoss
Convolutional neural network for capturing features
Autoencoder for learning benign traffic

FIGURE 7. The direct connection between one-dimensional CNN and autoencoder through the hidden layers of the CNN model.

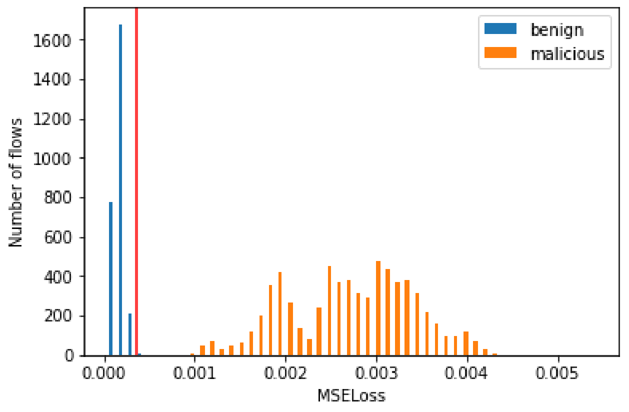

FIGURE 8. The statistical results of benign and malicious classification from CNN learning, including the MSELoss distribution of the testing set (the red line is the threshold set by the classification).

Also, n and l are referred to in Uniform Length Trimming. To anonymize the source of the attack traffic, particularly in the synthetic dataset, the identity information such as IP and MAC addresses is randomized, while the same identity of all the packets in the same flow is preserved. The next is the detection model on the right-hand side, which consists of two parts. The first part is the CNN for flow features extraction, and the second is the autoencoder for classifying flows into benign or malicious. The detailed design of the detection model is illustrated in Fig. 7.

For classification, a threshold is set to distinguish benign and malicious traffic based on the MSELoss distribution of benign traffic. Note that, when detecting an attack, the MSELoss distribution generated by the autoencoder training set (i.e., benign flows) is used for determining the detection threshold. We hope that this design can reduce the error caused by data dispersion and improve accuracy. Also, MSELoss calculates the flow based on the blurred features, and these features are treated as different vectors and put into MSE to calculate loss attributes. To avoid the impact of the extreme maximum value of the MSELoss, the maximum is compared with the 99th percentile of the MSELoss. For threshold, if the difference between these two values exceeds the triple standard deviation of the MSELoss distribution, the 99th percentile is set as the detection threshold; otherwise, the maximal MSELoss is set as the threshold. To improve the classification performance, we also adopt the approach to jointly optimize the loss functions of CNN and autoencoder, as shown in Fig. 7. Fig. 8 illustrates an example of the statistical results of MSELoss of benign and malicious flows from autoencoder. The red line is the threshold set by the above mechanism.

For the layer structure of the detection module, Table 5 shows the structural parameters of the design in detail.

TABLE 5. Structural parameters.

Layer	Type	Filters/neurons	Stride	Padding
1	1D-ConV+Relu+ Batch Normalization	32 (kernel size=6)	1	5
2	Maxpooling	Kernel size=2	2	-
3	1D-ConV+Relu+ Batch Normalization	64 (kernel size=6)	1	5
4	Maxpooling	kernel size=2	2	-
5	Dense + Batch Normalization	1024	-	-
*6	Dense + Batch Normalization	25	-	-
*7	Dense	10	-	-
8	Dense	512	-	-
9	Dense	256	-	-
10	Dense	512	-	-
11	Dense	1024	-	-

Note that the number of filters/neurons in the last layer of CNN depends on the number of types of benign traffic (e.g., 10 in the USTC-TFC2016 dataset), while the other parameters are fixed for different traffic types. First, the one-dimensional CNN filter gets the first l bytes of the first n packets as the input (i.e., the input size is $n \times l$); the kernel size of the filter of the convolution layer is set to 6 (in the header, the largest field is the MAC address). The hidden layer (layer 5) in the CNN, connecting to the autoencoder module, is used to learn the flow features. Specifically, layers from 1 to 7 in Table 5 are the architecture of the CNN, and layer 5 plus layers from 8 to 11 are the architecture of autoencoder. Summary, the structural parameters in Table 5 are the parameters that we refer to the architecture mentioned in [18] and improve them. Moreover, such parameters are what we have found after many cross-validation and experiments.

Hyperparameters for tuning

The performance of the DL-based approaches is susceptible to the changes of the hyperparameters such as the learning rate. In this work, tuning the following parameters may significantly help to improve the performance of the system.

1) n and l: As mentioned in Section III-A, these are the key parameters for early detection while maintaining high accuracy.
2) Batch normalization: Due to the depth of the deep learning architecture, the batch normalization between each layer can keep the parameter distribution stable, accelerate the learning efficiency, ease the gradient disappearance and avoid over-fitting.
3) Add additional dense layers for CNN: In general, adding more hidden layers could improve the performance of CNN. In our CNN architecture, it consists of 3 hidden layers (layer 5 to 7). What we have learned from our experiments is that adding the second layer with size of 25 improves the performance significantly. Our conjecture is that, besides the benefit of an additional hidden layer, the size of 25 actually corresponds to the number of header fields in a TCP/IP packet. Although we may also encounter UDP/IP packets with fewer header fields, from our observation, setting the size to 25 seems to be able to accommodate both TCP and UDP packets.
4) All dense layers adopt layer-wise greedy pre-training for initialization: The layer-wise greedy pre-training design targets at mitigating the impact of vanishing gradient problem and overfitting in the deep architecture. Fortunately, Unsupervised Greedy Layer-Wise Pretraining model [32] genuinely works well with the classification based on the autoencoder.

V. EVALUATION RESULTS

Similar to most existing deep learning research, our proposed classification model has been implemented using TensorFlow/Keras and Pytorch. The evaluations have been all performed on the GPU-enabled TensorFlow (Pytorch) running on a 64-bit Ubuntu 16.04 LTS server with an Intel Xeon Silver 4116 CPU@2.10GHz, 256GB RAM, and NVIDIA Tesla V100. We also use the Python package dpkt, a pcap parser and creator, to convert the raw packets for training.

To perform the evaluations, we have sequentially tested the system on USTC-TFC2016, Mirai-RGU, and Mirai-CCU datasets (mentioned in Section IV-A). Training the model is performed only on the benign traffic; however, to verify the system on various conditions, the testing scenario will involve three cases: the input data of only benign, benign and malicious, and pure malicious traffic. Also, since our approach aims to gain a significant reduction in the processing time, we prefer to classify an incoming flow, whether it is malicious or not, instead of considering its attack type in detail. In practice, if the proposed system detects a malicious flow, it will raise an alarm and redirect the packets to some off-line computationally intensive traffic classification systems for further analysis, while blocking the malicious flow simultaneously. Therefore, in the following subsections, several common metrics are used for performance evaluation in terms of a binary classifier.

- True Positive (TP) – number of attack flows that are correctly classified as an attack.
- False Positive (FP) – number of benign flows that are incorrectly classified as an attack.
- True Negative (TN) – number of benign flows that are correctly classified as normal.
- False Negative (FN) – number of attack flows that are incorrectly classified as an normal.

The accuracy in Eq. 3 measures the proportion of the total number of correct classifications. The precision, recall, and F1-mesaure are defined in Eq. 4, Eq. 5 and Eq. 6, respectively. The first two reflect the rate of correct classifications influenced by incorrect ones, and the last is the overall measure between the precision and the recall.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F1 = \frac{2TP}{2TP + FP + FN} = \frac{2 * Precision * Recall}{Precision + Recall} \quad (6)$$

The FAR in Eq. 7, also known as FPR, measures the rate of benign flows incorrectly classified as malicious. The FNR in Eq. 8 measures the rate of malicious flows incorrectly classified as benign.

$$False\ Alarm\ Rate = \frac{FP}{FP + TN} \quad (7)$$

$$False\ Negative\ Rate = \frac{FN}{TP + FN} \quad (8)$$

In the following subsections, we present the detection performance of D-PACK for three selected datasets in three scenarios: the input data for training and testing can consist of only benign, benign and malicious, and pure malicious traffic.

In the first scenario, the classes and number of flows of the training and testing sets are shown in Table 6. The training data in the CNN and autoencoder module in this scenario include only benign traffic. The testing data include benign traffic from USTC-TFC 2016 and malicious traffic from Mirai-CCU.

TABLE 6. Scenario 1: USTC-TFC 2016 (benign) and Mirai-CCU (malicious) datasets.

(a) Training set for CNN and autoencoder (benign from USTC-TFC 2016)

Traffic type	# of flows
BitTorrent	6000
Facetime	6000
FTP	6000
Gmail	6000
MySQL	6000
Outlook	6000
Skype	6000
SMB	6000
Weibo	6000
World of Warcraft	6000

(b) Testing set A (benign from USTC-TFC 2016 and malicious from Mirai-CCU)

Traffic type	# of flows
BitTorrent	2398
Facetime	2398
FTP	2399
Gmail	2399
MySQL	2399
Outlook	2399
Skype	2399
SMB	2399
Weibo	2399
World of Warcraft	2399
ACK Flood	5997
SYN Flood	5997
UDP Flood	5997
HTTP Flood	5997

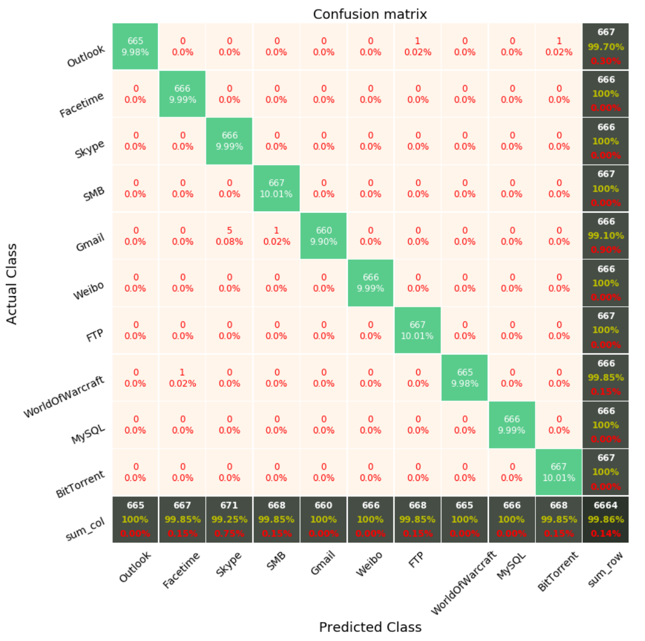

FIGURE 9. The learning performance of CNN for benign traffic.

TABLE 7. The accuracy of D-PACK running with various values of *n* and *l* (scenario 1: testing set A).

Packet count	Packet size (bytes)				
	40	50	60	70	80
2	99.96%	100%	100.00%	100%	100%
3	99.99%	99.99%	100.00%	100%	100%
4	99.97%	99.95%	100.00%	99.99%	100%
5	99.98%	99.39%	99.99%	99.99%	100%

TABLE 8. The performance of the framework in Scenario 1: testing set A with $n = 2$ and $l = 50$.

Measurements	Result
Accuracy	100%
Precision	100%
Recall	100%
F1-Measure	100%
FNR and FPR	0%

In the case of auto-learning traffic features, the learning performance of the CNN model on benign traffic is summarized in the confusion matrix table in Fig. 9. The top number in each cell of the confusion matrix is the number of flows for testing in the benign traffic set. The learning module can classify the benign traffic type with nearly 100% accuracy.

Table 7 shows the accuracy of D-PACK in the first testing scenario, i.e., on the testing set A, which contains benign flows from USTC-TFC 2016 and malicious flows from Mirai-CCU. As we observe from Table 7, with only 40 bytes per flow, the accuracy is nearly 100%. When the number of packets per-flow and bytes per packet are increased for inspection, e.g., 2 packets per-flow and 50 bytes per packet or 3 packets per flow and 60 bytes per packet, the system can achieve 100% accuracy, a very promising result. However, we also notice that when we increase the number of packets per flow does not yield higher accuracy. This result indicates inspecting more bytes per packet is better than more packets per flow. This is promising for reducing the flow processing time as well as the training and detecting time. The best result for the testing set A is with the configuration of $n = 2$ and $l = 50$, which yields ideal performance, i.e., 100% accuracy and no FPR/FNR. The performance with the five-metric measurements in this ideal case is presented in Table 8.

In the second scenario, the training data and testing data are all from the USTC-TFC 2016 dataset. The ratio of benign traffic in the training set and the testing set is 7:1, as shown in Table 9. The malicious traffic only appears in the testing set, making the ratio of the total amount of flows in the training set and that of the testing set becomes 4:1. After training, we have tested the system with the testing set B.

TABLE 9. Scenario 2: USTC-TFC 2016 dataset (both benign and malicious).

(a) Training Set for CNN and autoencoder (benign from USTC-TFC 2016)

Traffic type	# of flows
BitTorrent	5333
Facetime	5333
FTP	5333
Gmail	5333
MySQL	5333
Outlook	5333
Skype	5333
SMB	5333
Weibo	5333
World of Warcraft	5333

(b) Testing set B (both benign and malicious are from USTC-TFC 2016)

Traffic type	# of flows
BitTorrent	666
Facetime	666
FTP	666
Gmail	666
MySQL	666
Outlook	666
Skype	666
SMB	666
Weibo	666
World of Warcraft	666
Cridex	667
Geodo	666
Htbot	667
Miuref	666
Neris	666
Nsis-ay	666
Shifa	666
Tinba	667
Virut	667
Zeus	666

TABLE 10. The detection accuracy of running with variances of n and l (testing set B).

Packet count	Packet size				
	40	50	60	70	80
2	99.95%	100%	100.00%	100%	100%
3	99.99%	100%	100.00%	100%	100%
4	99.99%	99.96%	100.00%	100%	93.07%
5	99.96%	99.91%	99.95%	100%	100%

Table 10 shows the system performance on the testing set with variances of n and l. Compared with the first scenario, the system gains better performance with the low values of n. The performance with five metrics in Table 11 reinforces this indication. However, similar to the previous case, increasing the value of n and l does not mean that the system gets better performance.

In the third scenario, the training and testing data are from the Mirai-RGU dataset. Since this dataset classifies all benign traffic into one type, namely, the normal type; thus, we use both benign and malicious traffic to train the CNN, but only the benign traffic to train the autoencoder, as shown in Table 12. The testing data also include both benign and malicious traffic from this dataset, as shown in the third column of Table 12. The Mirai-RGU dataset contains nine types of attack traffic with quite a variant number of flows.

TABLE 11. The performance of the framework on the testing set B with $n = 2$ and $l = 50$.

Measurements	Result
Accuracy	100%
Precision	100%
Recall	100%
F1-Measure	100%
FNR and FPR	0%

TABLE 12. Scenario 3: training and testing sets from Mirai-RGU.

CNN Training set		Autoencoder testing and training		
Traffic type	# of training flows	# of testing flows	Traffic type	# of training flows
ACK Flood	6600	825	Normal	68200
HTTP Flood	120	15		
UDP Flood	28816	3062		
DNS Flood	4312	539		
Mirai C&C	68200	539		
VSE Flood	4432	554		
GREIP Flood	24712	3089		
SYN Flood	68200	8525		
Normal	68200	8525		

TABLE 13. The detection accuracy when running with the Mirai dataset for various values of n and l.

Packet count	Packet size				
	40	50	60	70	80
2	99.01%	99.11%	99.71%	99.76%	99.77%
3	97.88%	98.40%	99.67%	99.77%	99.77%
4	96.39%	97.60%	99.51%	99.71%	99.75%
5	95.54%	96.66%	99.38%	99.69%	99.73%

TABLE 14. Performance of the framework on the whole malicious traffic with $n = 2$ and $l = 80$.

Measurements	Result
Accuracy	99.77%
Precision	99.93%
Recall	99.17%
F1-Measure	99.55%
FNR and FPR	0.02% and 0.83%

The detection accuracy of running the framework on the testing data with variances of n and l is shown in Table 13. Compared with the first two scenarios, the performance is slightly worse with the same configuration of n and l, but the accuracy is still higher than 99.7% when two packets per-flow and more than 60 bytes per packet are examined. Moreover, in the applications where recursive communications probably generate extremely long flows, the configuration (i.e., $n = 2$, $l = 60$) still gives the detection system significant advantages in skipping tremendous traffic volume for processing.

Similar to the accuracy, the performance of the rest measurements in this scenario also deteriorates slightly. In this case, we set $n = 2$ and $l = 80$ for the sampling configuration. The system gains nearly 99% for the first four measurements,

TABLE 15. Pre-processing capacity of our framework on various datasets.

Datasets	Benign	Processing capacity (flows/s)	Malicious	Processing capacity (flows/s)
USTC	Weibo	11300	Geodo	18992
	MySQL	46716	Shifu	36901
	Skype	81402	Cridex	36330
	Gmail	77597	Tinba	83772
	Outlook	134833	Zeus	69217
	Facetime	315628	Miuref	99057
	SMB	87156	Nsis-ay	103894
	World of Warcarft	87410	Htbot	107344
	BitTorrent	233757	Neris	105687
	FTP	158360	Virut	115988
Mirai-CCU			HTTP Flood	37047
			SYN Flood	80604
			ACK Flood	127525
			UDP Flood	333088
Mirai-RGU	Normal	2555180	Ack Flood	521845
			HTTP Flood	205510
			UDP Flood	232113
			DNS Flood	130353
			Mirai C&C and spread	1743501
			VSE Flood	65618
			GREIP Flood	1651175
			SYN Flood	223064

while the FNR rises marginally but is still less than 1%, as summarized in Table 14. The results reinforce our original judgment that a well-designed learning strategy combined with a slight adjustment of n, l may not significantly impact the overall performance of the detection system while it can significantly help to save traffic volume for processing and speed up the detection. Admittedly, in the case of other applications and datasets, e.g., malware binary, the values of n and l can vary since their data structure for sampling may need a tweak.

In summary, our experiments in all the scenarios demonstrate that the performance of the proposed mechanism either outperforms or is competitive to that of state-of-the-art research works listed in Table 1.

A. TIME EFFICIENCY

The time efficiency in terms of the training time and the detection time is also an important metric in our experiment. The training time is the total time to complete the training on the selected dataset. This time depends on the data pre-processing, the training model (number of layers and dimensions) and the server capacity. Therefore, it would be unfair to compare the execution with different hardware and software. The pre-processing capacity of our system is summarized in Table 15. The system can capture the packets, classify them into different flows, extract n packets per flow and discard the rest with hundreds of thousands of the flows per second. Since the pre-processing time includes reading all the packets in the dataset, which demands much I/O time, and training the deep learning model, which requires much CPU/GPU processing time; thus, in some cases, it can only process tens of thousands of flows. However, the pre-processing task shall be done offline; thus, it is acceptable to have slightly longer pre-processing time.

TABLE 16. Detection speed with different datasets.

Training Set	Testing Set	Detection Speed
USTC-TFC2016	USTC-TFC2016	576,058 flows/s
USTC-TFC2016	Mirai-CCU	675,832 flows/s

Unlike the pre-processing, the detection on the same traffic type is supposed to be significantly faster because the classification can work on the raw traffic directly after training, instead of spending time in the learning tasks. In our empirical experiment, the detection can serve hundreds of thousands of flows per second (as summarized in Table 16), no matter which dataset it runs on. The processing capacity here is defined as the total number of flows that can be classified per second on a given testing dataset. With the processing of hundreds of thousands of flows per second, our approach can work for on-line monitoring since this speed can satisfy most on-demand applications and in medium networks. For the core networks, we may need more efforts to integrate this system for potential deployment, since the network speed at such nodes can be up to hundreds of gigabits per second. Note that the system at the core networks can be equipped with much more powerful hardware.

B. DISCUSSION

There is a trade-off between the number of trimmed packets, trimmed length, and the classification performance, i.e., accuracy and detection time. Reducing the trimmed length, e.g., from 80 bytes to 40 bytes, can potentially help to reduce the detection time and the training time, but may dramatically impact the accuracy. The accuracy also depends on the considered attack type. In the evaluation, we found that the ideal configuration is at $n = 2$ and $l = 80$. However, promising research is to propose an algorithm to calculate a proper configuration automatically in a deployed environment.

Moreover, the approach to consider the classification at the packet level can open the door towards accelerating the detection since we can schedule the packets for parallel processing. However, the explicit shortcoming is to increase the training time and resource usage (e.g., memory) due to a large number of parameters and data size put into the training model. Balancing the factor of acceptable training time but still gaining high classification performance is a non-trivial task and a potential research direction.

Finally, the DL-based classification approach is highly susceptible to the data poisoning attack due to its dependence on the training data. So far, we have found few attack models targeting at evading the deep learning-based malicious classification systems, including ours. However, this can soon be changed when the popularity of deep learning will attract more attackers to exploit its vulnerabilities for hacking or monetization. Generating/preventing adversary models against deep learning thus is a very interesting and promising security research topic.

VI. CONCLUSION

In this work, we present a novel early malicious traffic detection framework, namely D-PACK, based on traffic sampling, traffic auto-profiling (CNN), and an unsupervised DL model (autoencoder). By targeting at examining as few packets and number of bytes from each packet as possible, our system can significantly reduce the traffic volume for processing. The evaluation results show that D-PACK can detect malicious traffic with nearly 100% accuracy and less than 1% FNR and FPR, even if it examines only two packets from each flow and 80 bytes from each packet. Moreover, it is supposed to consume much less flow pre-processing time and detection time than prior works because much fewer packets and bytes are inspected. Thus, the important advantage of this framework is to speed up the detection. We believe that this first attempt can inspire the research community to consider further optimization methods, particularly by exploiting the advantages of deep learning to build effective online anomaly detection systems without suffering significant detection delay.

REFERENCES

[1] M. Bacon. *New Mirai Variant Attacks Apache Struts Vulnerability*. Accessed: Jun. 12, 2018. [Online]. Available: https://searchsecurity.techtarget.com/news/252448779/New-Mirai-variant-attacks-Apache-Struts-vulnerability

[2] S. Gatlan. *Mirai Botnet Variants Targeting New Processors and Architectures*. Accessed: Apr. 9, 2019. [Online]. Available: https://www.bleepingcomputer.com/news/security/mirai-botnet-variants-targeting-new-processors-and-architectures

[3] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[4] R. Hallman, J. Bryan, G. Palavicini, J. Divita, and J. Romero-Mariona, "IoDDoS the Internet of distributed denial of service attacks—A case study of the mirai malware and IoT-based botnets," in *Proc. 2nd Int. Conf. Internet Things, Big Data Secur.*, vol. 1, 2017, pp. 47–58.

[5] P. Zilberman, R. Puzis, and Y. Elovici, "On network footprint of traffic inspection and filtering at global scrubbing centers," *IEEE Trans. Depend. Sec. Comput.*, vol. 14, no. 5, pp. 521–534, Sep. 2017.

[6] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 303–336, 2014.

[7] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1153–1176, 2016.

[8] S. Valenti, D. Rossi, A. Dainotti, A. Pescap, A. Finamore, and M. Mellia, "Reviewing traffic classification," in *DataTraffic Monitoring and Analysis*. Cham, Switzerland: Springer, 2013, pp. 123–147.

[9] E. Min, J. Long, Q. Liu, J. Cui, and W. Chen, "TR-IDS: Anomaly-based intrusion detection through text-convolutional neural network and random forest," *Secur. Commun. Netw.*, vol. 2018, pp. 1–9, Jul. 2018.

[10] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *Proc. IEEE Int. Conf. Intell. Secur. Informat. (ISI)*, Jul. 2017, pp. 43–48.

[11] D. Berman, A. Buczak, J. Chavis, and C. Corbett, "A survey of deep learning methods for cyber security," *Information*, vol. 10, no. 4, p. 122, Apr. 2019.

[12] E. Min, J. Long, Q. Liu, J. Cui, Z. Cai, and J. Ma, "SU-IDS: A semi-supervised and unsupervised framework for network intrusion detection," in *Proc. 4th Int. Conf. Cloud Comput. Secur.*, 2018, pp. 322–334.

[13] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici, "N-BaIoT: Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervas. Comput.*, vol. 17, no. 3, pp. 11–22, Jul./Sep. 2018.

[14] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, "HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection," *IEEE Access*, vol. 6, pp. 1792–1806, 2018.

[15] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescape, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Trans. Netw. Serv. Manage.*, vol. 16, no. 2, pp. 445–458, Jun. 2019.

[16] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi, "Deep learning approach combining sparse autoencoder with SVM for network intrusion detection," *IEEE Access*, vol. 6, pp. 52843–52856, 2018.

[17] L. Vu, C. T. Bui, and Q. U. Nguyen, "A deep learning based method for handling imbalanced problem in network traffic classification," in *Proc. 8th Int. Symp. Inf. Commun. Technol. (SoICT)*, 2017, pp. 333–339.

[18] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, 2017, pp. 712–717.

[19] Z. Chen, K. He, J. Li, and Y. Geng, "Seq2Img: A sequence-to-image based approach towards IP traffic classification using convolutional neural networks," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 1271–1276.

[20] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Comput.*, vol. 24, no. 3, pp. 1999–2012, Feb. 2020.

[21] 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. 18042–18050, 2017.

[22] J. Hochst, L. Baumgartner, M. Hollick, and B. Freisleben, "Unsupervised traffic flow classification using a neural autoencoder," in *Proc. IEEE 42nd Conf. Local Comput. Netw. (LCN)*, Oct. 2017, pp. 523–526.

[23] I. Arnaldo, A. Cuesta-Infante, A. Arun, M. Lam, C. Bassias, and K. Veeramachaneni, "Learning representations for log data in cybersecurity," in *Proc. Int. Conf. Cyber Secur. Cryptogr. Mach. Learn.*, 2017, pp. 250–268.

[24] Y. Yu, J. Long, and Z. Cai, "Network intrusion detection through stacking dilated convolutional autoencoders," *Secur. Commun. Netw.*, vol. 2017, pp. 1–10, Nov. 2017.

[25] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning," *Int. J. Secur. Appl.*, vol. 9, no. 5, pp. 205–216, May 2015.

[26] V. Paxson and S. Floyd, "Wide area traffic: The failure of Poisson modeling," *IEEE/ACM Trans. Netw.*, vol. 3, no. 3, pp. 226–244, Jun. 1995.

[27] B. Hullar, S. Laki, and A. Gyorgy, "Efficient methods for early protocol identification," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 10, pp. 1907–1918, Sep. 2014.

[28] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. S Hneider, "Enriching network security analysis with time travel," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, p. 183, Oct. 2008.

[29] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proc. Int. Conf. Eng. Technol. (ICET)*, Aug. 2017, pp. 1–6.

[30] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," 2018, *arXiv:1802.10135*. [Online]. Available: http://arxiv.org/abs/1802.10135

[31] C. D. Mcdermott, F. Majdani, and A. V. Petrovski, "Botnet detection in the Internet of Things using deep learning approaches," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8.

[32] P. L. Y. Bengio, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. 19th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2006, pp. 153–160.

REN-HUNG HWANG (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Massachusetts, Amherst. He joined the Department of Computer Science and Information Engineering, National Chung Cheng University (CCU), in 1993, where he is currently a Distinguished Professor. He has published more than 200 international journals and conference papers. He has served as the Dean for the College of Engineering, from 2014 to 2017. He has received the IEEE Outstanding Paper Award from the IEEE UIC 2012, and the IEEE Best Paper Award from the IEEE IUCC 2014, the IEEE SC2 2017, and the IEEE Ubi-Media 2018. His current research interests include the Internet of Things, network security, cloud/edge/fog computing, and software-defined networks.

MIN-CHUN PENG is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan. His research interests include information security and deep learning, deep learning, cloud computing, the Internet of Things, and information security.

CHIEN-WEI HUANG is currently pursuing the master's degree with the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan. He is working on L2, L3 network protocol. His research interests include information security and deep learning.

PO-CHING LIN received the Ph.D. degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2008. He joined the Department of Computer Science and Information Engineering, National Chung Cheng University (CCU), in August 2009, as a Faculty Member, where he is currently an Associate Professor. His research interests include network security, network traffic analysis, and performance evaluation of network systems.

VAN-LINH NGUYEN (Member, IEEE) received the Ph.D. degree in computer science and information engineering from National Chung Cheng University (CCU), Taiwan, in 2019. He is currently an Assistant Professor with the Department of Information Technology, TNU-University of Information and Communication Technology, Vietnam. His research interests include network security, vehicular security, the Internet of Things, deep learning, and edge computing.