R1DIT: Privacy-Preserving Malware Traffic Classification with Attention-Based Neural Networks

Onur Barut[§], *Member, IEEE,* Yan Luo, *Member, IEEE,* Peilong Li, *Member, IEEE,* and Tong Zhang, *Member, IEEE*

***Abstract*—With the advances in deep learning techniques and the increase in the volume of network traffic data, deep neural networks trained directly with the raw traffic data have become more popular and successful for malware traffic classification without explicit feature extraction. However, most of the existing studies raises privacy concerns when using the payload data and ignore the generalization of the model to the newly emerged traffic such as DDoS detection on TLS 1.3. To overcome these limitations, we introduce a malware traffic classification system, Residual 1-D Image Transformer (R1DIT) model. We first leverage network domain knowledge by carefully parsing IP, HTTP, DNS, and unencrypted TLS record headers as sequences of bytes for input without interfering with IP addresses, port numbers and the payload. Then, we apply raw data transform and attention-based modules in our deep model to classify different malware types and benign traffic. Our results on NetML dataset show that the proposed model delivers 0.972 F1 score, nearly 0.3 higher than the feature-based methods and outperforms state-of-the-art models with 0.9999 F1 score for multi-class malware classification task using CICIDS2017 dataset. The generalization of this model has been proven using the TLS 1.3 traffic obtained from CICDDoS2019 dataset with the detection rate 0.9897 using meta-learning.**

***Index Terms*—Flow-based network intrusion detection, malware traffic classification, deep learning, image transformer model, meta learning.**

I. Introduction

THE importance of emerging information and communication technology solutions play an important role in our social and economic life as the number of interconnected devices increases. In this sense, technological developments directly affect our lives with an increased exposure to malicious attacks, such as violating user privacy, stealing sensitive data for ransom, and disabling network services either by flooding or damaging the hardware or the software [1]. Similarly, many companies, governments, and even universities face various types of cyberattacks to steal valuable information for ransom or crash their services with Distributed Denial of Service (DDoS) attacks to damage their reputation. Also, recent developments like TLS 1.3 bring new security concerns as attackers can quickly adapt their malware using this change to disguise their malicious intention. For this reason, it is extremely important that the network security system adapts quicker to this change to keep the servers running and protect valuable information. Hence, an adaptable, scalable, and cost-effective solution for emerging network attacks is of paramount importance. For this purpose, cyber security experts and researchers are focusing on creating a secure Internet in the age of exponentially growing digitalization.

In the early days, port-based and signature-based payload analysis including Deep Packet Inspection (DPI) were the dominating approaches. However, there are two major disadvantages: i) DPI violates of the user's privacy by analyzing the packet payload; ii) new malware and unseen malware families can easily escape from the detection due to outdated databases. Moreover, the increasing adoption of encrypted protocols (TLS) alongside dynamic ports is eliminating these approaches, posing new challenges for accurate network traffic classification. For example, malware traffic classification comes with exacerbated challenges and requirements due to (i) increasing numbers of malware, and (ii) frequent automatic updates of malware and traffic infrastructure resulting in insufficient training samples per malware type and not delivering the desired performance. Therefore, Machine Learning (ML) based classifiers are considered the most suitable solution.

Conventional ML models require feature engineering to train an accurate classifier. However, successful use of standard ML classifiers relies heavily on obtaining handcrafted features by a domain expert that correspond to statistics extracted from the packet sequences in the network flow. Unfortunately, identifying and extracting these features is time consuming and rapidly outdated compared to the evolution of malware traffic, precluding the design of accurate and up-to-date malware traffic classifiers with traditional ML approaches. Additionally, ML-based models trained on a dataset generally do not work well when applied to another dataset with slightly different distribution and require expert intervention, which is often tedious and greatly slows down the cycle of DevOps.

Based on the above limitations, the use of raw data and Deep Learning (DL) methods in network traffic analysis is highly desired in the most recent works. Deep Learning is a sub-domain of Machine Learning which leverages the flexibility and complex design of artificial neural networks. Unlike traditional ML algorithms, DL models can learn their own features once fed directly by the raw data. One drawback of DL models is that they require a large volume of training

Onur Barut and Tong Zhang are with the Intel Corporation, Santa Clara, CA, 95054 USA e-mail: ({onur.barut,tong2.zhang}@intel.com)

Yan Luo is with the Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA, 01854 USA e-mail: Yan_Luo@uml.edu

Peilong Li is with the Department of Computer Science, Elizabethtown College, Elizabethtown, PA, 17022 USA e-mail: lip@etown.edu

[§]The work was conducted when the author was Ph.D. student at the University of Massachusetts Lowell.

data to learn these self-extracted representations for an accurate classification. Luckily, thanks to more readily available network traffic data, deep learning models can get properly trained and unleash the power on malware classification.

Recently, attention-based deep neural networks have shown to outperform the state-of-the-art in object detection and text analysis tasks [2], [3]. Especially for text analysis where a word in the sentence (e.g., subject) may affect another word's form (e.g., verb) in a sentence, attention-based solutions can exploit this relation to learn the representations for each input. Hence, attention-based neural networks offer a great opportunity for accurate malware traffic classification, as the network packets and sequence of packets in a flow have a very similar structure to that used in text analysis.

Securing the user's privacy in the network is one of the most vital goals when designing malware traffic classification systems (MTCSs) for sustainable Internet usage. Companies providing online services must adhere strictly to the GDPR guidelines to prevent litigation over the possible exposure of sensitive data. Since users' data is transferred at the application layer, it would be desirable for the business to perform a network traffic analysis without using the application layer payload to protect users' private information and reduce the risk of being held accountable for the exposure of sensitive data in the event of a breach. Defense mechanisms to classify the network traffic in MTCSs are generally designed using packet-based or flow-based methods. Flow-based methods can be implemented either with machine learning combined with flow feature engineering or deep learning applied directly to the raw flow data [4], [5] while packet-based methods are mostly designed with raw data and deep learning [6]. However, previous works utilize either the payload data or do not perform anonymization, where in both cases the privacy of the user is violated. Besides, prior works ignore the domain specific knowledge and miss a huge potential for a significant improvement in the accuracy.

One of the major issues in malware detection is the constant rise of unseen attacks. Most of the detection systems trained with older dataset cannot perform desirably on the unseen traffic due to the changes of the traffic behavior. For example, with the release of TLS 1.3 in late 2018, typical TLS 1.2 handshake with 5 to 7 exchanged packets between client and server, which places significant overhead on the connection, has been replaced by server certificate encryption making it possible to perform a TLS handshake with 0 to 3 packets. Therefore, the effectiveness of a malware traffic classification system trained with traffic on TLS 1.2 on detecting the unseen encrypted malware traffic using TLS 1.3 is yet to be explored.

In this article, we propose an attention-based privacy-preserving malware traffic classification system which allows to train directly from raw traffic data by automatically learning feature representations for high-performance classification in a dynamic and demanding network traffic environment to generalize to unseen malware traffic. We implement a novel Residual 1-D Image Transformer (R1DIT) model with 2-D positional encoding and multi-head self-attention mechanisms to classify malware traffic. We utilize IP and TCP/UDP header data without IP and port addresses for anonymization and

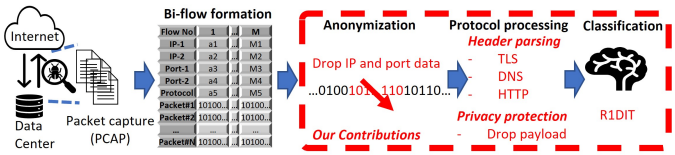


Fig. 1: Flowchart for our malware traffic classification system.

generalization. We also utilize the domain knowledge parsing TLS, DNS and HTTP traffic and take the advantage of unencrypted header information without any payload data. Our model demonstrates superior accuracy by comparing to different classifiers trained with the extracted flow features.

Our contributions in this article are threefold: (1) we propose a novel network flow classification technique leveraging protocol-specific raw header data by preserving users' privacy with self-attention based neural network; (2) compare the accuracy and the inference speed of the proposed model to prior works employing flow features or raw bytes as input utilizing NetML [7] and CICIDS2017 datasets [8] and outperform the state-of-the-art; and (3) propose a generalization approach for our model that uses meta-learning to improve its output class capacity to identify newly emerging TLS 1.3 malware traffic without sacrificing the original accuracy with the limited size of the TLS 1.3 subset of CICDDoS2019 [9] dataset.

The rest of the article is organized as follows. In Section II, we discuss the similarities and differences of our work to the related literature. In Section III, we describe the proposed R1DIT deep neural network and model the malware traffic classification problem with meta learning. In Section IV, we provide detailed explanation about the experiment setup. In Section V, we present and discuss experimental results of the proposed system, and finally in Section VI we conclude our work indicating possible future research directions.

II. Related Work

In Table I, we summarize several state-of-the-art malware traffic classification systems to address their limitations and relevance to position our study in the literature. The first group uses flow features as input type which by default considers privacy of the users as no payload data is utilized. However, their primary drawbacks are the time and complexity of the feature engineering step as well as the different feature preprocessing methods used in each research, which makes it difficult to provide a unified solution. The second group uses raw data with deep learning algorithms for more accurate results to get around this issue; nevertheless, most of them do not take privacy concerns into account by processing the payload data. The third group, unlike others, tackles an important problem of generalization and propose methods to classify unseen and newly emerged traffic. However, there is still a gap in the literature that we want to close with this work by presenting a comprehensive privacy-preserving approach that uses raw data and deep learning to identify new and previously undisclosed malware traffic. By utilizing the raw traffic data and removing user IP addresses and payload data, we aim to resolve this discrepancy and show how it is simple to modify for accurate unseen and newly discovered TLS 1.3 malware traffic classification.

TABLE I: Summary of relevant literature for malware traffic classification. RD: Raw Data. FF: Flow Feature. PA: Proposed Algorithm. ML: Machine Learning. DL: Deep Learning

Ref.	Dataset	Input Data	Privacy Preserving	Method	Unseen Traffic	notes
[8]	CICIDS2017	FF	✓	ML	✗	Introduces CICIDS2017 dataset
[9]	CICDDoS2019	FF	✓	ML	✗	Introduces CICDDoS2019
[10]	ISCX2012,CICIDS2017	FF	✓	DL	✗	Deep learning with CNN and LSTM
[11]	CICIDS2017	FF	✓	DL	✗	DNN outperforms RF
[12]	ISPDSL,Entry09,Entry10	FF	✓	DL	✗	Multi-task flow prediction
[13]	CICIDS2017	FF	✓	ML	✗	RF outperforms all including DNN with their proposed feature set
[14]	NSL-KDD,UNSW-NB15, CICIDS2017	FF	✓	ML	✗	NSGA-II for feature selection
[7]	NetML,CICIDS2017	FF	✓	ML	✗	Introduces NetML dataset and new set of flow features
[15]	NetML,CICIDS2017	FF	✓	ML/DL	✗	Compares ML/DL models on TLS traffic with feature selection by Chi-Square
[16]	CICIDS2017	FF	✓	ML	✗	25 features selected by Fischer Score algorithm
[17]	CICIDS2017	FF	✓	ML	✗	Applies PCA on features to classify DDoS
[18]	NetML,CICIDS2017, VPN-nonVPN2016	FF	✓	DL	✗	Multi-task model with hierarchical predictions using 121 flow metadata features
[19]	CIDDS-001,CICIDS2017, CICDDoS2019	FF	✓	PA	✗	Proposes Energy-based Flow Classifier (EFC) using Inverse Statistics
[20]	USTC-TFC2016	RD	✗	DL	✗	First 784 bytes of a packet (28x28) including TCP payload with CNN
[4]	CTU-13,VPN-nonVPN2016	RD	✗	DL	✗	First 1024 (32x32) bytes of each flow with CNN
[5]	Private	RD	✗	ML/DL	✗	First 784 bytes of L7 payload
[6]	ISCX2012,USTC-TFC2016	RD	✗	DL	✗	First 54 bytes for packet-level malware detection with LSTM
[21]	VPN-nonVPN2016	RD	✓	DL	✗	Proposes Self-Attention Model with first 40 bytes for packet classification
[22]	MTAN,CTU	RD	✗	DL	✗	Specialized for encrypted traffic only
[23]	NSL-KDD	FF	✓	PA	✓	Proposes Clustering-Enhanced Hierarchical Transfer Learning (CeHTL) to detect unseen malware traffic based on transfer learning
[24]	NSL-KDD	FF	✓	DL	✓	Zero-shot learning based sparse autoencoder to detect unseen malware
[25]	ISCX2012,CICIDS2017	RD	✗	DL	✓	First 16 packets and 256 bytes per packet with CNN based FC-Net and applies Few-shot learning to detect unseen malware traffic
[26]	Bot-IoT	FF	✓	DL	✓	Hierarchical anomaly detection with autoencoder to detect unseen malware
[27]	CICIDS2017,CICIDS2018	RD	✗	DL	✓	First 30 packets and 128 bytes per packet and applies Few-shot learning to detect unseen recent malware traffic
Ours	NetML,CICIDS2017, CICDDoS2019	RD	✓	DL	✓	Proposes accurate R1DIT model with raw data that can be easily generalized to unseen TLS 1.3 malware traffic

A. Traditional Machine Learning Methods

Feature engineering is necessary to reduce the input size for training without losing important or relevant information. Although feature extraction implicitly preserves the privacy as the extracted features are not obtained through payload processing, a well-engineered feature processing is required to train an accurate classifier. Hence, many researchers propose different methods for feature engineering [7], [13]–[17]. However, each method heavily depends on the dataset under study, and it is a challenge to apply a method in another dataset to obtain an accurate result. Additionally, the design and execution of the feature extraction and selection step take a significant amount of time and are computationally costly. To circumvent these issues, many studies use the raw data. In our study, unlike these works, we completely depend on raw header data and exclude IP and payload data from the packets to ensure the privacy is preserved in our method it can be generalizable.

B. Deep Learning Based Methods

The most significant advantage of deep learning based classifiers is that they can be trained using directly from the raw data without feature extraction. Although many researchers prefer to leverage this advantage [4]–[6], [20]–[22], [25], [27] because of improved results, there are others who prefer to train deep learning models using pre-extracted features [10]–[12], [15], [18], [24], [26]. However, this brings additional cost of complexity to the classification algorithm and the need for larger amount of training data. Moreover, when the privacy of the users is of utmost importance, extra cautions must be taken while utilizing the raw data. Nevertheless, except Xie *et al.* [21], all other aforementioned studies utilize the application layer payload to train their classifiers. In our research, unlike others, we develop a deep learning-based solution employing unencrypted header data and packet anonymization for privacy.

C. Detection of Newly Emerged Malware Traffic

Previously proposed approaches only focus on the accuracy on the classification using individual datasets, which raises concerns on their performance in the wild where frequently newly released malicious traffic emerges. To evaluate and modify the proposed approaches on the unseen malware traffic, feature-based [23], [24], [26] and raw data-based [25], [27] techniques are studied. However, NSL-KDD dataset used by [23], [24] is relatively outdated and does not reflect the recent advances. On the other hand, [25], [27] uses relatively recent datasets; however, their method utilizes the application layer payload and hence fails to satisfy the privacy issue. To guarantee privacy while adapting to identify emerging TLS 1.3 encrypted threats, our study solely uses the header data and eliminates IP and port data from the packets.

III. Proposed System Design

We propose our deep learning-based malware classification system shown in Figure 1. First, network traffic is captured beforehand and stored as packet capture file format (PCAP). Second, raw data processing is applied to parse the packet

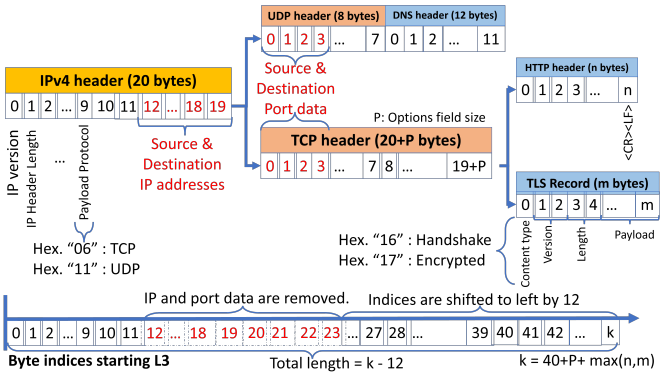


Fig. 2: IP packet parsing in terms of bytes.

data starting from the first bit of IP header and form bi-flow sessions according to 5-tuple, source and destination IP addresses, port numbers and the transport layer protocol. Link layer header, or Ethernet header which contains Media Access Control (MAC) addresses and link type, is excluded because this information reflects the connection setup of the machine to the Internet with its specific MAC addresses that won't generalize. Third, extracted bi-flow sessions are anonymized by removing the IP and port address values to protect the user's privacy. This step also ensures that the proposed system does not overfit, in other words, memorize the static IP and port addresses due to the traffic capture setup. After that, TLS, DNS, and HTTP protocol headers are carefully parsed without interfering with the payload. As options field in IP header is not often used, we do not take it into account when parsing the IP header data.

Byte values in the packet sequence affect each other. For instance, transport layer header size is either eight bytes for UDP or twenty for TCP if the byte value at position index 9 of IP header is hexadecimal "11" or "06", respectively. Similarly, the next packet in the flow may contain "SYN-ACK" information for a TCP flow if the first packet is a TCP handshake initiation "SYN". Because of these spatial and temporal relation in the flow byte sequence, 2-dimensional position encoding is implemented. Finally, extracted raw data is fed to a deep learning model, namely Residual 1-D Image Transformer (R1DIT) model making use of 2-D position encoding and embedding layers. The following subsections provides detailed explanations about the components of our proposed malware classification system.

A. Raw Data Processing and Anonymization

The packet switching on the Internet contains private data of the user such as MAC and IP addresses of the users' devices. In our system, we do not use layer two header; hence, avoid exploiting MAC addresses because layer two header contains information about only the physical connections between the devices which cannot be used to design a generalizing system. Therefore, we start parsing each network packet starting from IP header. Figure 2 demonstrates how we ensure the anonymity and parse the IP packet.

Even though migration from IPv4 to IPv6 continues, most of the network traffic is still handled with IPv4. Under such assumption, we build our system using IPv4 traffic. IPv4 packet is composed of 20-byte header and the payload. Byte index 9 in the IP header gives the information about the encapsulated transport layer protocol. The hexadecimal value "06" and "11" indicates if the IP packet payload is TCP or UDP datagram, respectively. We store this information because UDP and TCP header sizes are not the same as UDP has 8-byte header while TCP has 20 (or up to 60, depending on the options field). According to each protocol, the positions of the following bytes represent different information. Finally, last two 4-byte data of IPv4 header represents the source and destination IP addresses, respectively. Therefore, we drop the byte locations corresponding to IP addresses to comply with the user's privacy. This also guarantees that the designed model does not memorize this information due to the limited number of machines used to collect the traffic data.

The first byte of the datagram is the first byte of the IP packet payload. We continue parsing transport layer datagram after IP header. As in the case of benign traffic, most of the malicious data are transferred using either UDP or TCP as transport layer protocol. In both cases, the first two 2-byte data in the datagram header stand for the source and destination port addresses. Therefore, we also drop those to establish a general model that performs well on real-world scenarios.

B. Protocol Specific Packet Processing

We disregard link layer header, which is specific to network interface cards and do not carry much information about network flows. We make use of IP header in network layer because they reflect non-device specific Internet traffic behavior. For similar reasons, we continue utilizing the raw data for the transport layer protocols such as TCP and UDP. In such a way, we do a first pass to narrow down the range of bits in packet headers to feed to our machine learning models.

TLS protocol is implemented on top of TCP for secure and reliable data transfer. The first byte of TLS header defines the content type of the TLS record. Hexadecimal "16" indicates that the TLS record is a handshake record in which the authentication between the two devices occurs with the certificate key and cipher suite exchange. In this case, we take advantage of all the data available in the TLS record since it is not the encrypted data but the initiation of an encrypted connection. Hexadecimal "17" specifies that the TLS record contains the encrypted data; therefore, we exclude the protocol message in the TLS record but parse the byte data right before the message starts.

Similarly, HTTP is an application layer protocol for data communication for web serving build on top of TCP. The HTTP header of the request or response messages contains metadata to be used for traffic classification. Therefore, we parse the HTTP packets up to the byte location where the message begins by checking the Carriage return and Line Feed (<CR><LF>) which is the last byte before the message.

Unlike HTTP and TLS protocols, DNS protocol which translates domain names to IP addresses uses UDP in transport layer. Every device connected to the Internet has a unique IP address, similar to physical address. With DNS, the users

can reach to other users or services by typing the host name instead of IP in the browser. DNS traffic consists of queries and responses to obtain the IP address so that the data exchange can take place. We take advantage of all the query and response data in DNS for malware traffic classification.

Network traffic other than the specified protocols above are parsed up to the end byte of the transport layer protocol. All packets in the flow are parsed as described and eventually we obtain data with input shape LxN where L and N represent used header length in terms of bytes in each packet and the number of packets, respectively. We build such a parser state machine which can be extended to work with new or customized protocols.

C. Feature Embedding and 2-D Positional Encoding

In to generate rich representations of the raw input data with both spatial and temporal relationships, we use feature embedding with position encoding. Similar to text classification tasks where the relative position for each token is important, we firstly implement a feature embedding layer for our raw packet data. Therefore, we embed the LxNx1 input data to LxNxD using 2-D convolution of size D. The feature embedding layer is implemented and trained within the proposed model. In other words, the model does not count on any other pre-trained embedding layer at this stage.

It is important that we can leverage the temporal and spatial relation among the header bytes since the byte value of each position in the packet sequence affect the other as explained in Section III. The model we opt does not use a recurrent neural network due to its inefficiency in training. Instead, we need to introduce some information to the model regarding the position of the bytes in the raw data sequence. Since our input data is in 2-D form and each instance consists of packets sent and received in an order, we secondly expand the positional encoding introduced by [3] into 2-D and capture the relative positions of the raw byte sequence for each packet by implementing a non-trainable two-dimensional sinusoidal position encoding. LxNx1 input data is sinusoidal encoded to LxNxD, which is the same dimension with feature embedding, allowing us to sum both for further processing in the model.

Using the justification stated above, Algorithm 1 implements feature embedding with 2-D positional encoding. We represent the network flow data formed by the sequence of packets and bytes as $\mathbf{F}$ of size $LxNx1$. Firstly, the learned feature embedding $\mathbf{FE}$ is obtained to enrich the meaning of each byte in tensor $\mathbf{F}$ with $\mathbf{FE} = \mathbf{F} \star W_c$ where $\star$ denotes the 2-dimensional convolution operation, W_c is the learned convolution kernel of size K that is updated during training, and $\mathbf{FE}$ is the $LxNxD$ dimensional embedding tensor. Secondly, the position encoding encodes the position P of a byte in a single packet data, which is sequence of packets in $Lx1x1$ dimension, into a $Lx1xD$ dimensional position tensor $\mathbf{PE}$ via sine and cosine functions with a constant 10000. As discussed by in [3], learned and fixed position encoding do not vary in the output. Thus, we select sinusoidal encoding to reduce the number of learned parameters in our model. Moreover, it creates a geometric progression which allows our model to learn relative positions. Finally, we concatenate the $\mathbf{PE}$ along the dimension of the packets, which refers to axis 1, and obtain the final representation tensor $\mathbf{Y} = \mathbf{FE} + \mathbf{PE}$.

Algorithm 1: 2-D Feature Embedding and Position Encoding

Result: Y /* *Y.shape: [L, N, D]* */
Input: K, N, L, D, F
/* *1) Create feature embedding tensor FE* */
$FE \leftarrow conv2d(embed\_dim = D, kernel\_size = K, stride = 1)(F)$
/* *2) Create position encoding tensor PE* */
$angle\_rads \leftarrow range(L)/10000^{2*(range(D)//2)/D}$
i=0 **while** *i<D* **do**
  if *i%2==0* **then**
    $angle\_rads[:, i] \leftarrow sin(angle\_rads[:, i])$
  else
    $angle\_rads[:, i] \leftarrow cos(angle\_rads[:, i])$
  end
  i+=1
end
/* *Initialize position encodings with angles* */
$PE \leftarrow angle\_rads.reshape(L, 1, D)$
/* *Loop every packets in the flow and expand position encoding tensor* */
j=1 **while** *j<N* **do**
  $PE.concatenate(PE * cos(1/N^{j/N}), axis = 1)$
  j+=1
end
/* *3) Obtain and return output tensor Y* */
$Y = FE + PE$

D. Residual 1-D Image Transformer Model

Deep neural networks accomplish many tasks successfully such as object recognition in computer vision and text analysis in natural language processing. Recently, attention-based models such as transformers achieve supreme performance in both fields [2], [3]. Network traffic and packet formatting can be compared to the relationship between words in a sentence. In either case, the value of one object affects the value of the other object in the array. For example, a word (or byte values at certain locations in the packet) determines the structure of another word in a sentence (the byte value of another index in the packet). This relationship can be used successfully by attention-based models to classify different malware types.

Inspired from those, since the bytes in each position have different meaning and their importance differ, we propose our Residual 1-D Image Transformer (R1DIT) which is shown in Figure 3 (A) utilizing the transformer's attention mechanism. Preprocessed network traffic capture data, also called as raw data in the model, is fed to both feature embedding and position encoding as described in previous sections. Extracted abstract representations are then forwarded to two cascaded residual transformer blocks. Later, a maximum pooling in two dimensions are performed to reduce the complexity in the model which is followed by a flattening operation to obtain a one-dimensional representation vector for each instance.

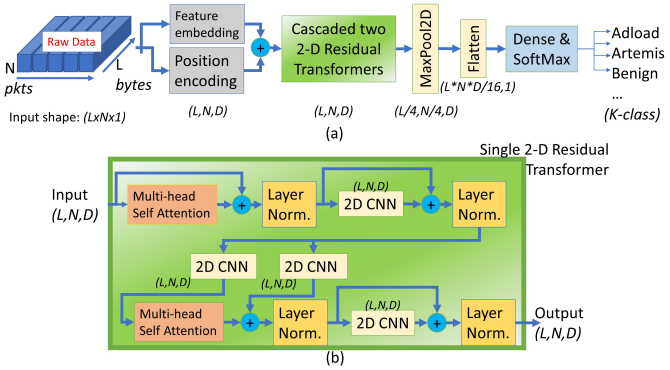


Fig. 3: (A) Residual 1-D Image Transformer (R1DIT) Model. (B) Structure of a single residual transformer block.

Finally, output class probabilities are obtained using a dense layer with Soft-Max activation function.

Residual skip connections are important for efficient training using simple first order algorithms [28] and hence enable the model to back-propagate the gradients without vanishing. In addition, it allows to create deeper neural networks with proper training which empowers the classification performance. Therefore, we employ residual skip connections in our transformer-based model. Figure 3 (B) illustrates that single residual transformer block consists of multi-head self-attention, layer normalization and 2-D convolutional layers with residual connections. Since the input and output shape of the residual transformer block are the same, multiple blocks can be cascaded; however, in our malware classification problem, we find that using two of those residual transformer blocks provide sufficient depth to grasp the essential representations for a successful classification.

E. Meta Learning for Malware Traffic Classification

The goal of meta learning is learning to learn so that the model can easily be adapted to classify new classes with very limited amount of support samples. On the other hand, traditional machine learning techniques rely on large volume of labeled datasets for classification tasks that fail to generalize on unseen data. In other words, we assume that the data used to train the model and the test samples are independent and identically distributed (iid) which may not be always true in a real-world scenario.

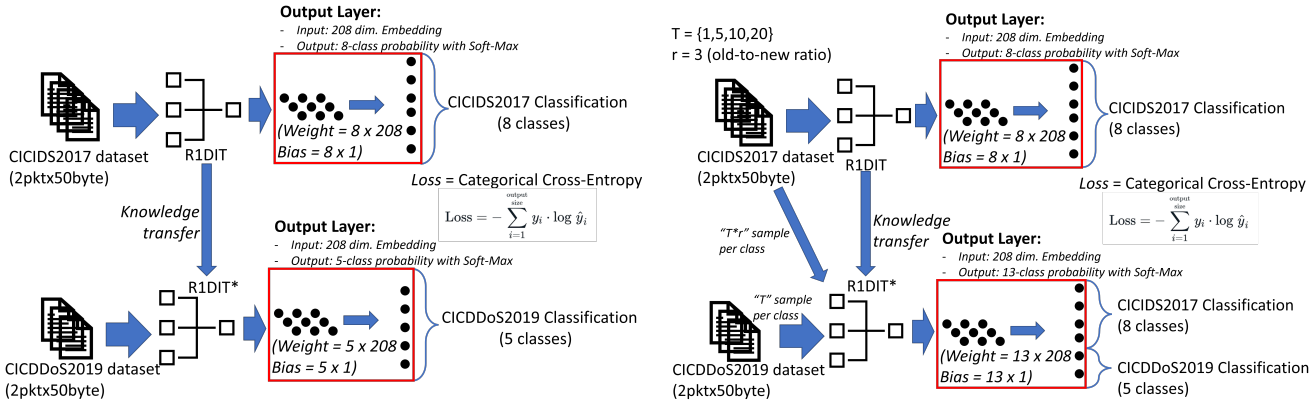
Applications and malicious software are constantly being updated and frequently new versions or unseen traffic behavior are observed in this very dynamic networking environment. As a recent example, packet exchange behaviour of TLS flows has recently been altered significantly as described in Section I. However, due to slow adaptation of the network applications, it is hard to collect a lot of data after TLS 1.3 release to train a deep model. Therefore, it is necessary to use the prior knowledge from a similar problem. To that extent, we claim the hypothesis that although previously collected network traffic data do not have any samples belonging to TLS 1.3 traffic but reflect similar distribution, the model trained on that traffic would be useful to classify TLS 1.3 malware traffic. In this study, we evaluate how previously collected data would generalize to the unseen traffic on TLS 1.3 and propose transfer learning and few-shot learning methods using R1DIT model. To show that with real examples, we have selected CICDDoS2019 dataset as a publicly available and recently collected traffic to extract TLS 1.3 traffic and picked CICIDS2017 trace as the source dataset since its traffic distribution is more similar to CICDDoS2019.

1) Transfer Learning (TL): Transfer learning aims to use the information extracted from the source domain with large volume of labeled data to help build more precise models in the same or a different target domain using only a few labeled data. There are two implementations of transfer learning: (1) Target adaptation and (2) expanded output. Target adaptation learning uses the source data to train the model parameters and use this prior information to hot-start the model for the target domain. In this method, the output layer is redesigned for the target domain and the source data classification capability of the model is not preserved. On the other hand, expanded output learning both preserves the model to perform accurate classification on the source domain while expanding its output layer to classify new classes in the target domain.

Figure 4 demonstrates how we implement transfer learning to detect TLS 1.3 malware traffic. We use CICIDS2017 dataset as source domain and TLS 1.3 traffic of CICDDoS2019 dataset as target domain. Firstly, we train the R1DIT model on CICIDS2017 dataset and chop-off the output layer. Then, we add a new output layer from the target domain with 5 classes for target adaptation learning and 8+5 classes for expanded output learning. For expanded output learning, the first 8 outputs correspond to the source dataset (CICIDS2017) classes and the rest correspond to 5 different DDoS malware types with TLS 1.3 obtained from the target dataset (DDoS2019). We freeze the weights except the new output layer and fit the model to the T-sample per new class training set to overfit the output layer where $T = \{1, 5, 10, 20\}$. If the target class contains fewer samples than T, then we leave at least 1 sample for the validation set and use the rest for the training. Finally, we unfreeze all the weights and fine-tune the whole model with T-sample per new class and Txr sample per old class, where r stands for the ratio of older samples to newer in the training batch and is set to be 3.

2) Few-Shot Learning (FSL): Few-Shot Learning is mainly proposed for computer vision tasks to recognize new classes without collecting a lot of samples for those to train the model. Instead, the proposed method learns how to compare two embedding representations generated by a pre-trained model to classify a query, in our case, raw bytes of the network traffic, based on a specific metric, e.g., cosine similarity. If only one sample is available for the new class, it is called One-Shot Learning (OSL). We want to detect TLS 1.3 malware traffic as the query sample. We know that our pre-trained R1DIT model using CICIDS2017 dataset does not contain any sample belonging to these new classes we want to detect in the support set. The support set contains n-samples for k-new-classes, and it differs from the training set such that training set contains lots of samples in the source domain.

The model which can learn to map the given input into embedding vector is trained using a Siamese network topology



(a) Target Adaptation Learning

(b) Expanded Output Learning

Fig. 4: Transfer Learning for TLS 1.3 Malware Traffic Classification

such that the generated representations are similar for the intra-class samples and different for the inter-class samples. Figure 5 (a) shows how the Siamese network is trained with triplet loss. Triplet loss is determined with an anchor sample, positive sample, and a negative sample. In each training iteration, an anchor sample is randomly selected from the training set. Then, a positive sample is randomly selected from the same class. Similarly, a negative sample is randomly selected from the training set excluding the anchor class. Finally, the model is trained such that the cosine similarity function outputs as close as possible to value 1 for the anchor and positive sample and -1 for the anchor and negative sample. We use CICIDS2017 dataset as the source domain to train our R1DIT model in a Siamese topology.

In the next step, the pre-trained R1DIT model needs to be appended by an output layer whose weights should be initialized with the mean class embeddings for a faster and more accurate convergence [29]. As R1DIT model transforms the input to a vector representation of size 208, we extract mean class embeddings for both source and target datasets in the shape of $8x208$ and $5x208$, respectively, as shown in Figure 5 (b). Then, we use the concatenated weight matrix of size $13x208$ to initialize the weights of the output later before softmax to calculate the class probabilities. The fine-tuning training stage is similar to transfer learning. Finally, we use T-sample per new class and Txr sample per old class to fine-tune the whole model where $T = \{1, 5, 10, 20\}$ and $r = 3$ as shown in Figure 5 (c). We leave at least 1 sample for the validation set and use the rest for the training if the target class contains fewer samples than T.

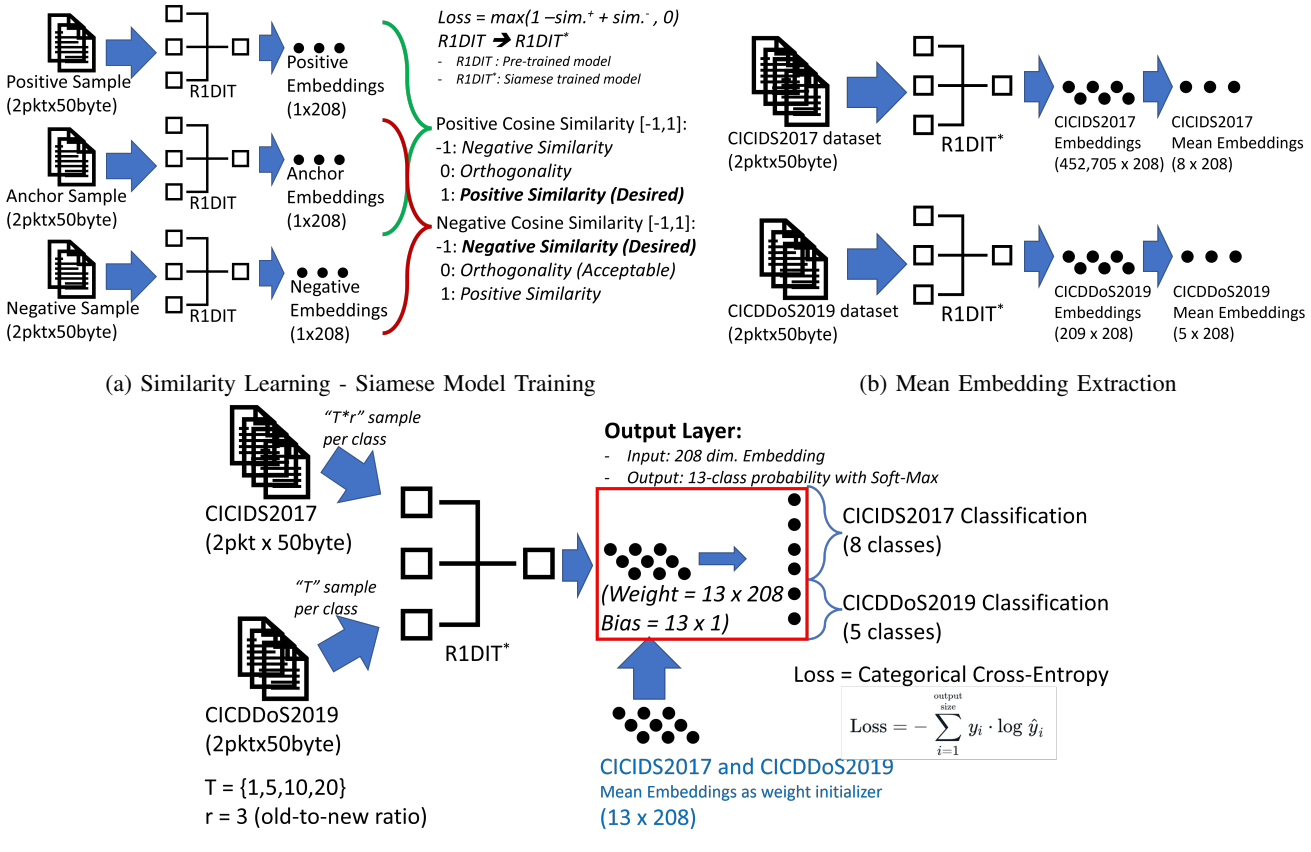
IV. Experiment Setup

A. Dataset

In our experiments, we focus on a multi-class classification task for both encrypted and non-encrypted malware traffic classification. We use two public datasets to evaluate our model. Both Stratosphere IPS [30] and Canadian Institute for Cybersecurity (CIC) provide public network capture data in '.pcap' format for research purpose. We firstly compile a new set of malware traffic classification dataset utilizing a subset of a very large repository of Stratosphere IPS selecting 19 different types of malware classes such as Adload, PUA, TrickBot, Ramnit, and Ransom etc. along with benign traffic data mostly captured in 2017 and name it NetML dataset [15]. We also utilize CICIDS2017 [8] dataset to compare our model with the previous works. In CICIDS2017, raw capture data contains the whole trace record throughout the day. However, during the data collection process, malware attacks are performed only in a specific time slot of the day. Therefore, we filter those network traffic of our interest according to the provided time stamp in the dataset web page, then label the extracted data into 8 classes including "DDoS", "DoS", "portScan", "benign" etc. We extract the bi-flows according to the five-tuples and end up with $288,918$ and $452,705$ flow samples for NetML and CICIDS2017 datasets, respectively.

Flow features extracted from the two datasets for the flow feature-based baseline experiments are well described in [7]. In this study, we focus on utilizing the raw-packet data and compare our results with the flow feature-based approach. For that purpose, we utilize the raw network traffic capture files of the datasets. Unlike the NetML dataset, which consists only of network capture files in PCAP format, the CICIDS2017 dataset is available both in PCAP format as raw data and in CSV format, which provides 80 statistical features extracted using CICFlowMeter and PCAP files. [8]. The CSV file is mostly used by researchers who focus on model optimization. PCAP files, on the other hand, are more useful for those who want to extract their own flow features from the raw data. Additionally, PCAP files are more attractive to deep learning researchers as they can harness the power of deep neural networks over the raw data. Therefore, we use PCAP files instead of CSV files in our experiments with CICIDS2017 dataset.

The network flow data are standardized, in other words, the mean is extracted and from each sample and divided by the standard deviation of the dataset. We train the proposed model with the 90% of the data and obtain results using the remaining 10% utilizing 10-fold cross validation and provide the averaged results. Presenting 10-fold cross validation results supports our claim of an accurate and generalizable model by avoiding presenting a snapshot of a single fold. Both NetML



(a) Similarity Learning - Siamese Model Training

(b) Mean Embedding Extraction

(c) Fine-Tuning

Fig. 5: One(Few)-Shot Learning for TLS 1.3 Malware Traffic Classification

TABLE II: Number of Bi-flow Samples in Different Categories in NetML Dataset

	# of Flows			# of Flows	
Label	Training	Test	**Label**	Training	Test
Adload	34	5	MagicHound	31,004	3,445
Artemis	9,211	1,023	MinerTrojan	1,620	181
Benign	56,385	6,265	PUA	933	106
BitCoinMiner	909	101	Ramnit	410	45
CCleaner	13,982	1,553	Tinba	10,203	1,134
Cobalt	229	26	TrickBot	23,777	2,641
Downware	937	104	Trickster	45,990	5,110
Dridex	8,984	998	TrojanDownloader	915	101
Emotet	17,604	1,957	Ursnif	9,480	1,054
HTBot	19,565	2,170	WebCompanion	7,639	849

TABLE III: Numbers of Bi-flow Samples in Different Categories in CICIDS2017 Dataset

	# of Flows			# of Flows	
Label	Training	Test	**Label**	Training	Test
DDoS	40,338	4,482	ftp-Patator	3,547	394
DoS	12,847	1,428	portScan	136,697	15,188
Heartbleed	1	0	ssh-patator	2,210	246
Benign	210,030	23,336	webAttack	1,764	197

dataset and CICIDS2017 dataset details in terms of number of flow samples used in training and test sets are given in Table II and Table III, respectively.

To evaluate R1DIT model on TLS 1.3 unseen malware traffic, we use CICDDoS2019 dataset. Table IV shows the number of TLS 1.3 flows extracted from the dataset for each malware type. Using the capture files for both days, we extract 209 TLS 1.3 malware traffic flows for 5 DDoS variations.

TABLE IV: Extracted TLS 1.3 bi-flow Samples from CICDDoS2019 Dataset

Label	LDAP	MSSQL	NetBIOS	PortMap	SYN
Day 1	2	3	10	3	186
Day 2	3	1	1	0	0
Total	5	4	11	3	186

B. Environment

We implement the training and inference phases on a workstation whose specifications are detailed in Table V. The inference time to evaluate the efficiency of the R1DIT model for a real-world scenario is measured and the accuracy is denoted with macro-average F1 score to better understand the performance of the model in the multi-class classification task. The models are trained on GPU and their performance is measured on both GPU and CPU for the inference phase.

C. Evaluation Metrics

Classification Accuracy: To compute the performance of the proposed malware traffic classification system, macro average F1 score, detection rate, also known as True Positive Rate (TPR) or sensitivity and False Alarm Rate (FAR) which is

TABLE V: Experiment Platform Specification

Item	Specifications	Item	Specifications
Op. Sys.	Ubuntu 16.04.6 LTS	NVidia Driver	440.64.00
Python	3.8.6	CUDA Driver	10.2
Tensorflow	2.3.1	CuDNN Driver	7.6.5
CPU	2x Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz		
GPU	2x GV100GL NVIDIA Tesla V100 16 GB		
RAM	64 GB DDR4 @ 2666 MHz		

TABLE VI: Hyperparameters used in the experiments

Loss Function	Cat. Cross-Entropy	Optimizer	Adam
Learning Rate (LR)	Exp. decayed	Initial LR	0.0005
LR Decay Rate	0.9	LR Decay Steps	10000
Batch Size	64	Epoch	100
Embedding Dimension	16	Number of Heads	4
Conv. Kernel Size	3	Conv. Stride	1
MaxPool Kernel Size	4	Activation Function	ReLU
MaxPool Stride	4	Dropout ratio	0.25
(2x50) Trainable Parameters:		$40,052^{*}$, $37,544^{\dagger}$	

*: NetML dataset, †: CICIDS2017 dataset

equal to $1 - Specifity$ of the validation split is obtained using the equations given from (1) to (3)

$$Recall = \frac{TP}{TP + FN}, Precision = \frac{TP}{TP + FP} \tag{1}$$

$$F1 = \frac{2 * Recall * Precision}{Recall + Precision} \tag{2}$$

$$TPR = \frac{TP}{TP + FN}, FAR = \frac{FP}{FP + TN} \tag{3}$$

where TP, TN, FP, and FN stand for true positive, true negative, false positive, and false negative for a category, respectively. The macro average F1 score is calculated by taking the arithmetic mean of each category.

We calculate the cosine angle to determine the similarity between the embeddings in FSL experiments. Similarity between anchor (A) to positive (P) and anchor to negative (N) embeddings are calculated as given in equation (4)

$$Similarity = \frac{\sum_{i=1}^{H} A_i.B_i}{\sqrt{\sum_{i=1}^{H} A_i^2}\sqrt{\sum_{i=1}^{H} B_i^2}} \tag{4}$$

where H stands for the size of the embeddings and $B = \{P, N\}$ for positive and negative embeddings, respectively.

Performance Effectiveness: To evaluate the real-time applicability, we measure the time taken by the malware traffic classification algorithm using network flows as input. For flow feature-based models, we measure the time taken by parsing each packet and executing the computation for many different flow features. For our raw data based R1DIT model, we only measure the time to parse and preprocess the raw packets to input to the model. For both methods, we also measure the time taken to classify the test set.

D. Optimization Parameters

We use NetML dataset for model parameter optimization and use the optimum model to evaluate its performance in CICIDS2017 dataset to compare the proposed framework with other's approaches. Our model's weights are randomly initialized. Similarly, hyper-parameters such as batch size, dropout ratio etc. are initialized with default values and then tuned for the best model to achieve highest performance.

Table VI shows the hyper-parameters set to obtain the presented results. Smaller values of kernel size such as 3 are more desirable to make the most of the neighbor bytes and packets relation while the attention mechanism is utilized. Therefore, convolutional kernel size for each operation is set to be 3 with stride value 1. Max-pooling is used to reduce the dimension for memory efficient operations in the output layer and performed using 4x4 window with stride size 4. Moreover, the batch size and dropout ratio are set to 64 and 0.25, respectively. Dropout is applied to avoid over-fitting where the model performs well on training set by memorizing the training samples and fails to generalize on the test set.

After hyper-parameter optimization, the embedding dimension parameter and the number of heads in the attention block are set to 16 and 4, respectively. Number of heads in the attention splits the embedded input data into different heads to avoid losing local information in a large matrix. Rectified Linear Unit (ReLU) is used as the activation function in all layers except the output layer where Soft-Max function is used for multi-class classification. The model is trained for 100 epochs to ensure the convergence.

Classification loss whose explicit function is given in equation (5) below is determined by categorical cross-entropy.

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} \mathbf{1}_{y_i \in C_c} log(p_{model}[y_i \in C_c]) \tag{5}$$

where L is the loss for classification model, N is the batch size, C is the number of classes, $\mathbf{1}$ is the indicator function, and $log(p[y_i]$ is the log-likelihood of the given input y_i belonging to class c after mapped with model p.

As the model loss converges, the smaller values of learning rate allow the model to converge to the minimum. Therefore, learning rate schedule is applied to reduce the initial value as given in equation (6). The learning rate is initialized with 0.0005 and decay rate 0.9 with decay step 10000 are applied for optimum convergence using Adam optimizer $lr(s) = lr_0 * d_r^{(s/d_s)}$ where lr_0, d_r, d_s, and s are the initial learning rate, decay rate, decay step, and step number, respectively.

For few-shot learning, we train the Siamese model with triplet loss as defined in equation (6)

$$TripletLoss = N - \sum_{i=1}^{N} sim^{+} + \sum_{i=1}^{N} max(sim^{-}, 0) \tag{6}$$

where N is the batch size, sim^{+} and sim^{-} are the similarity between anchor-positive and anchor-negative, respectively.

E. Baseline Models

We compare our R1DIT model with several different approaches with and without feature engineering. Flow features used to train these models are metadata features and TLS, DNS, and HTTP protocol features if utilized within the flow. If a flow does not contain any of these protocols, i.e., a proprietary encryption protocol stacked on IP/TCP or any other

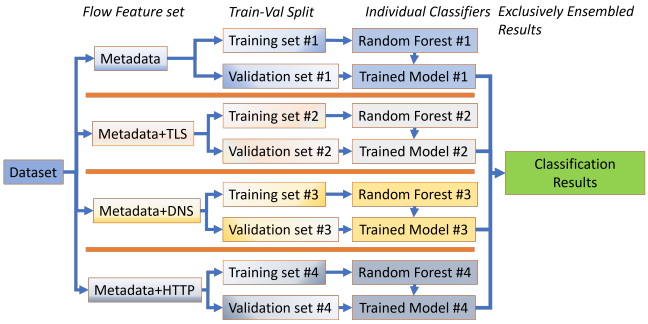


Fig. 6: Multi-model approach to classify the malware traffic.

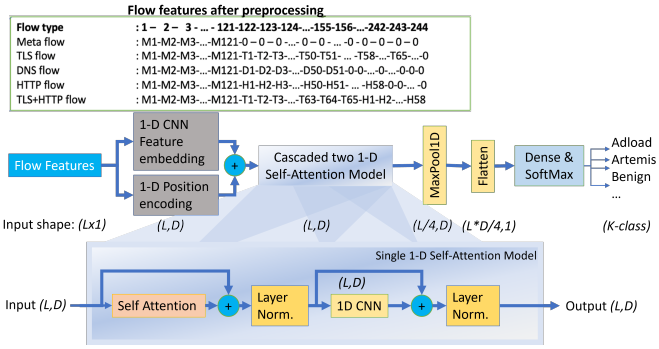


Fig. 7: Self-Attention Mechanism (SAM) model for Malware Traffic Classification.

UDP-based protocol except DNS, only the metadata features are extracted. The features used in this study are calculated using up to first 40 packets in each flow. A comprehensive list of these flow features can be found in [7].

1) Single Random Forest Classifier: A single random forest classifier works as an ensemble model and consists of many individual decision tree classifiers. The main reason we include this classifier as a baseline model is because it is easy and fast to implement. Single random forest (RF) model for malware traffic classification utilizes all the metadata, TLS, DNS, and HTTP features together. To design the random forest classifier, we select 100 estimators and set the maximum depth to 10.

2) Multi-Model Random Forest Classifier: As Random Forest classifier takes a 2-D matrix as input, where columns correspond to features, specifically, Metadata, TLS, DNS, and HTTP features in the given order, and rows correspond to each flow sample, we create the data matrix accordingly. Since the protocol-specific features are exclusive, i.e., a DNS flow will never contain TLS features and vice-versa, our data matrix will contain many entries as zero in TLS, DNS, and HTTP related feature columns. Therefore, multiple random forest classifiers are individually trained for different types of flows such as DNS, TLS, HTTP, and others using the corresponding feature subsets. In other words, all the features are utilized; however, each classifier only uses the features-of-interest for the classification. To say, a TLS encrypted flow is classified by a random classifier which was trained only with metadata and TLS features while an HTTP flow is classified by another model trained only with the metadata and HTTP features. Figure 6 explains the proposed model.

3) Multi-Task Hierarchical Learning (MTHL) Model: MTHL model introduced by Barut et al. [18] consists of an input block, a residual block, and an output block utilizing 1-D convolutional operations, batch normalization, and skip connections for enhanced training performance. The input block takes the preprocessed features as input and produces activation maps which are then forwarded to the residual block for deeper representation learning. Finally, the output block predicts the high-level and low-level labels of the input simultaneously and hierarchically such that the high-level label estimator branch uses the low-level predicted label as auxiliary information. This is also reversely coupled, i.e., the loss due to the error in the high-level label prediction both back propagates to high-level label classification branch and low-level label classification branch through this auxiliary connection. This model is trained using only the flow metadata features of size 121 using non-vpn2016, CICIDS2017, and NetML datasets together and shown in [18] that this model outperforms single-task single-level machine learning classifiers in 22 out of 24 scenarios with different levels of classifications in all three datasets. Due to its success in flow feature-based network traffic classification and malware detection, we use the results reported in [18] to evaluate our raw data-based R1DIT model.

TABLE VII: R1DIT model input shape optimization on NetML dataset

Macro F1-score	28 bytes	50 bytes	75 bytes	100 bytes	125 bytes
2 packets	0.918	0.972	0.967	0.963	0.952
5 packets	0.919	0.971	0.966	0.970	0.969
7 packets	0.912	0.965	0.970	0.960	**0.975**
10 packets	0.914	0.967	0.956	0.955	0.940

4) Self-Attention Based Model: We also adopt the text classification idea from the NLP domain and implement a self-attention-based deep learning model inspired by [21] trained with flow features to evaluate the effectiveness of flow features. Figure 7 shows the SAM model used for malware traffic classification with flow features. Since the flows are grouped as TLS, DNS, HTTP, or combination of TLS and HTTP which actually refers to the flows that terminates when TLS v1.0 handshake fails with fatal alert on protocol version terminating the flow right after HTTP 502 Bad Gateway message is received, we arrange the feature sequence such that the metadata features are followed by the protocol-specific features immediately. The length of the feature sequences L is set to 244 and D to 16, which are the sum of the number of the metadata, TLS, and HTTP features and the embedding dimension size, respectively. The parameter L is determined by the dataset as it includes flows containing TLS and HTTP traffic together. The remaining positions are zero-padded for other types of flows as shown in Figure 7.

V. Results And Discussion

A. Determining L and N

Experimental results of this study are presented and discussed in this section to demonstrate the effectiveness and efficiency of the R1DIT model. Network flows may contain different number of packets depending on the size of the data to be transferred and the Maximum Transmission Unit (MTU)

of the network connection, which is 1500 bytes for an Ethernet connection. Therefore, many packets with different sizes can be found in a single network flow. To find out the optimum configuration, we perform a parameter search for the number of packets and the size of each packet in terms of bytes and present the macro average F1 scores in Table VII. Each row gives the accuracy in macro average F1 score with the given size of bytes used from each packet and each column specifies the size of each packet. Second column which shows the results when 28 bytes are used for each packet stands for the plain analysis. In other words, 28 bytes are the IPv4 header and TCP/UDP headers without IP and port address values and any other protocol specific data such as TLS, DNS, HTTP.

We observe that using only 28 bytes, namely, not leveraging any domain knowledge to process TLS, DNS and HTTP protocols yields the least accurate results with any number of packets used in the flow. We also observe that using only the first two packets can achieve very similar accuracy when the first 5 packets used with 28 bytes per packet. This observation is also valid with other size of packets such as 50, 75 and 100.

Table VII also shows that the input configuration with first 7 packets and 125 bytes for each packet achieves the highest F1 score with 0.975. Similarly, using first 2 packets and 50 bytes per packet achieves 0.972 macro average F1 score. This means that using only the first two packets is sufficient to achieve an accurate multi-class malware classification. Although the latter option is slightly less accurate, in a real-world scenario where the model is deployed for an online malware detection in a data center where the volume of network traffic is huge, it needs only the first 2 packets of a flow to execute the prediction. Furthermore, in a case where the malware flow traffic is assumed to have less than 7 packets, the packets would be stored in the buffer and the malware detection framework would wait for the remaining to fill the first 7 packets until a time threshold is passed. This would become unfeasible and drastically reduce the effectiveness of the malware traffic classification system. Hence, we denote that using first two packets with 50 bytes in our framework is the most suitable configuration for a fast and effective detection system. We also note that our model does not utilize any payload data and replace them with zero values to classify the malware in the network to protect the users' privacy.

B. Comparing R1DIT to baseline models

Table VIII depicts the accuracy of each method in macro-average F1 score. We can understand from results that the malware classification systems tend to achieve much higher accuracy with the raw data as input rather than using flow features. This is because the models with the presence of high volume of training data can learn more abstract but powerful representations from the raw data for a more accurate classification. For example, a random forest classifier trained with extracted flow features can produce 0.636 F1 score. Similarly, Barut et al. [18] proposes a multi-task hierarchical learning model which uses only the metadata features and achieve slightly better accuracy with 0.639 F1 score. When we train multiple random forest (Multi-RF) classifiers using the flow features, we can increase the overall classification accuracy up to 0.677 macro-average F1 score. By implementing a self-attention-based deep learning model proposed in [21] for the flow features, we achieve 0.679 F1 score, which is the best with the flow features. However, it is not much different than the other simpler feature-based classifiers meaning that the extracted set of flow features determines an upper limit to the classification accuracy. When we switch the input to the raw data instead of flow features, we can achieve 0.783 F1 score using a random forest classifier. This can be further improved to 0.972 and 0.975 with our proposed deep learning based R1DIT model using the first two packets of the flow with first 50 bytes per packet and seven packets with 125 bytes per packet as the input configuration, respectively. The complex weight connections of the proposed neural network help the model capture the powerful representations from the raw data which eventually results in a higher accuracy.

To evaluate the efficiency of the R1DIT model for the end-to-end inference speed, we define the metric flow per second which shows how many flow samples can be executed for prediction for each second. Because the buffering time required to extract the network flow features can vary for each flow, we consider offline classification scenario in which the number of required packets are already available in the buffer. Table VIII also provides the end-to-end inference speed both for CPU and GPU as deep learning models perform different on different architectures due to the number of arithmetic logic units and the clock speed. The time measurements for the preprocessing and the classification algorithms are done on the test set which contains 28k flow samples. $t_{process}$ and t_{clf} show the packet processing time to transform the raw network traffic data for the classifier algorithm and the time that the classifier run to classify all the flow samples in the test set, respectively.

We observe that complex deep learning models are more accurate but much slower than the traditional machine learning classifiers in the inference phase, especially when run on CPU. However, the packet processing time for the feature-based methods take much longer time than raw bytes-based methods. Moreover, the waiting time to fill the buffer with the first 40 packets before extracting the flow features adds buffering time to overall pipeline. This disadvantage is significant for online classification in an environment such as a data center where the immediate threat detection is imperative. However, using raw data, we do not need to extract any feature. Instead, we need to parse the packet and preprocess to drop the IP and port address bytes as well as payload itself. Since only first 2 packets are shown to be sufficient for accurate malware traffic detection, the packet processing time is much shorter compared to feature-based methods.

The proposed R1DIT model performs the most accurate but the slowest in terms of flow prediction per second due to its complex structure. We also observe that using GPU for the traffic classification, the inference speed can be improved in the range of 2.2 to 10 times when compared to the inference speed on CPU. The large number of parameters used in the neural network requires a use of GPU for a speedup; however, our proposed model with 2 packets and 50 bytes also performs

TABLE VIII: End-to-end performance comparison of different approaches on NetML dataset

Method		Macro F1-Score	$t_{process}$ (sec) for 28k flows	t_{clf} (sec) for 28k flows		End-to-end Inference Speed (flow/sec)	
Input	**Model**		CPU	CPU	GPU	CPU	GPU
Flow Features (318x1)	Single Random Forest	0.636	0.4386	0.1987	N/A	43,935	N/A
Flow Features (metadata 121x1)	MTHL [18]	0.639	0.4386	N/A	N/A	N/A	N/A
Flow Features (318)	Multi-RF	0.677	0.4386	0.3526	N/A	35,389	N/A
Flow Features (244x1)	Self-attention-based model	0.679	0.4386	42.81	4.2949	647	5,915
Raw bytes (100x1)	Single Random Forest	0.783	0.0202	0.1241	N/A	194,040	N/A
Raw bytes (2x50)	R1DIT (ours)	0.972	0.0202	10.59	4.881	2,639	5,713
Raw bytes (7x125)	R1DIT (ours)	0.975	0.0708	40.638	5.556	688	4,976

TABLE IX: Comparison of R1DIT model to the state-of-the-art on CICIDS2017 dataset. 'TP' stands for Trainable Parameters

Method	# of TP	Macro F1-score
Flow Features - Single ID3 [8]	N/A	0.98
Flow Features - MTHL [18]	131,000	0.955
Flow Features - DNN 3 Layers [11]	91,216	0.935
Flow Features - CNN+LSTM [10]	29,139	0.988
Flow Features - Single RF [13]	N/A	0.999
Raw bytes - FC-Net [25] (16x256)	1,058,945	0.9999
Raw bytes - R1DIT (2x50)	37,544	0.9999

on the CPU as the half speed of GPU. This raises a future research direction to optimize our model to achieve a speedup on CPU to reduce the need for a GPU for inference.

To validate that the proposed R1DIT model generalizes to other datasets, we also perform malware classification using public CICIDS2017 dataset. Table IX presents our F1 score with comparison to other studies. We show that our model not only outperforms state-of-the-art for this dataset in terms of F1 score, but also achieves this using only first two packets of the flow and 50 bytes per packet without payload data and with considerably smaller number of trainable parameters while FC-Net needs 256 bytes per each of 16 packets as input. Although multi-class malware classification with this dataset is relatively an easy task where most of the other studies can achieve more than 0.95 F1 score using different types of flow features, we claim that our R1DIT model relying on raw data with protocol processing outperforms the state-of-the-art with only an average of 5 misclassification out of 45270 test samples and hence it generalizes to other datasets.

C. Detection of TLS 1.3 DDoS traffic with Meta Learning

We also want our model to detect the unseen malware traffic with a recent release of TLS 1.3. However, it is very challenging to find or collect a lot of data to train our model from scratch for this purpose. Therefore, we need to leverage the prior knowledge about the same problem, which is the malware traffic classification. To that extent, we consider utilizing R1DIT model that performs 0.9999 accurate on training and validation sets of CICIDS2017 dataset. Figure 8 visualize the raw data (left) and the embeddings obtained in the final layer of R1DIT model (middle) on 2-D plane using Truncated SVD (Singular Value Decomposition) for CICIDS2017 dataset. We observe that R1DIT model can transform the raw data into a more separable plane as the main reason of its success to achieve 0.9999 accuracy in the CICIDS2017 dataset.

Our hypothesis is that although CICIDS2017 dataset does not have any samples belonging to TLS 1.3, the model pre-trained on CICIDS2017 traffic would perform accurate on detecting the CICDDoS2019 TLS 1.3 traffic as malware. However, our experiment results show that the accuracy on CICDDoS2019 TLS 1.3 dataset is less than 0.3. We visualize the raw data and the embeddings obtained in the final layer of R1DIT model for these two datasets to understand why the accuracy is lower in the newer dataset. The left-most and middle plots in Figure 8 also show the raw data and the embeddings of CICDDoS2019 TLS 1.3 dataset along with the CICIDS2017 data, respectively. In the raw data subplot, we see that newer samples are mostly coming from another distribution where the model does not have seen any previously. This is expected because TLS 1.3 is a newer version and there are differences of those when compared to previous versions. This causes the model to misclassify most of the new classes. To solve this problem, we propose to implement transfer learning (TL) or one/few-shot learning (OSL, FSL). The right-most plot in Figure 8 shows that similarity learning with Siamese architecture can help the model differentiate and cluster the new traffic embeddings.

TABLE X: Expanded Output Learning results of both CICIDS2017 and CICDDoS2019 TLS 1.3 traffic using 2x50 R1DIT model pre-trained on CICIDS2017

		Binary task TPR		Binary task FAR	Multi-class task Accuracy	
	(T)	**IDS**	**T1.3**	**IDS**	**IDS**	**T1.3**
BL	–	0.9999	0.3	0.0001	0.9999	0.06
TL	1	0.9999	0.701	0.0069	0.9963	0.3775
	5	0.9999	0.9897	0.0003	0.9997	0.4124
	10	0.9999	0.9543	0.0075	0.9960	0.6057
	20	0.9999	0.9297	0.0021	0.9988	0.5838
OSL	1	0.9999	0.6471	0.0015	0.9991	0.0098
FSL	5	0.9999	0.9897	0.0030	0.9983	0.0052
	10	0.9999	0.9892	0.0002	0.9998	0.0
	20	0.9999	0.9029	0.0034	0.9981	0.1529

- T: the number of samples for each class in the target dataset used in training
- The ratio of samples for each class in the source dataset used in training: 3

Table X compares the results obtained using different meta-learning methods for expanded output learning. Comparing the results to baseline where there is no transfer learning shows that even with 1 sample per new class with TLS 1.3 improves the detection accuracy of TLS 1.3 malware from 0.3 to 0.7. Since the size of the target dataset is very small (LDAP:5, MSSQL:4, NetBIOS:11, PortMap:3, SYN:186) and highly unbalanced, it is very hard to draw a general conclusion from the experiments. However, within the given scenario of detecting a new version DDoS malware traffic encrypted using TLS 1.3, we can show in Table X that both transfer learning (TL) and one(few)-shot learning (O/FSL) methods provide higher accuracy than using the baseline model which is trained using a dataset that does not include any TLS 1.3 traffic.

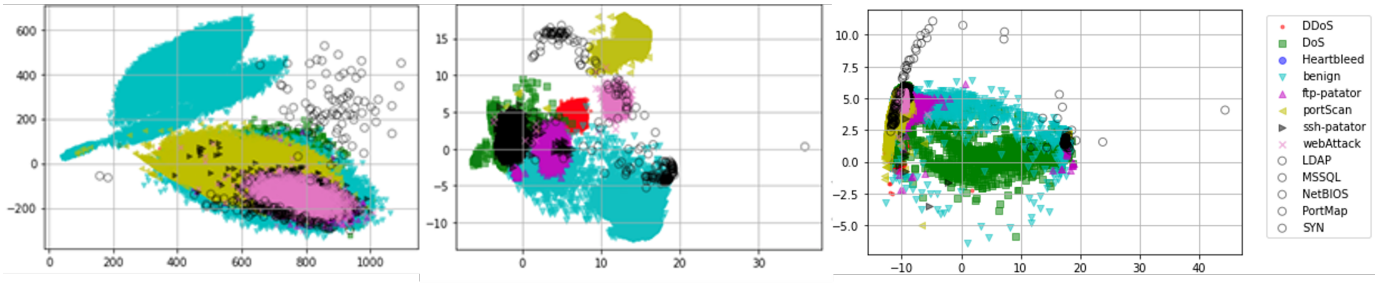


Fig. 8: Raw data (left), H-208 embeddings after R1DIT is pre-trained (middle), and H-208 embeddings using Siamese Model trained with cosine similarity (right)

TABLE XI: Target Adaptation Learning results of CICDDoS2019 TLS 1.3 traffic using 2x50 R1DIT model pre-trained on CICIDS2017

			CICDDoS2019-TLS 1.3		
	T-R	Acc.	TPR	FAR	Multi-Class Accuracy
TL	(1)-(3)	0.9978	0.7500	0.0001	0.4216
	(5)-(3)	0.9997	0.9794	0.0002	0.5103
	(10)-(1)	0.9996	0.9838	0.0003	0.6757
OSL	(1)-(3)	0.9975	0.7892	0.0007	0.4657
FSL	(5)-(3)	0.9994	0.9381	0.0001	0.4742
	(10)-(1)	0.9990	0.9892	0.0009	0.6270

- T: the number of samples for each class in the target dataset used in training
- R: the ratio of samples for benign class obtained from the source dataset for training

Although the results for source CICIDS2017 dataset in both binary and multi-class classification tasks are satisfying, target CICDDoS2019 TLS 1.3 accuracy is not as desired, in both cases. Both TL with one sample per target class and OSL achieve around 0.7 detection rate in TLS 1.3 encrypted DDoS malware. When there are 5 or more number of samples available for the target TLS 1.3 encrypted DDoS samples in the training set, detection of TLS 1.3 DDoS malware reaches up to around 0.99. Other major observation is that TL is more capable in multi-class classification than O/FSL. Even with only one sample per class, multi-class CICDDoS2019 TLS 1.3 malware classification accuracy is 0.3775 while with OSL, multi-class classification is 0.0098 for the target dataset.

We also implemented TL and O/FSL for Target Adaptation Learning scenario. In this scenario, we also introduce the benign traffic from the source dataset to target adaptation learning since the target dataset does not include any traffic for benign class. First, the source dataset CICIDS2017 is used to pre-train the model to obtain a prior knowledge to classify the classes in the target CICDDoS2019 dataset. Then, the model is fine-tuned using the target dataset classes combined with the benign traffic from source dataset with a ratio r determining the number of benign samples used to include in the training as explained in Figure 4 and Figure 5. The results are given in Table XI. We can observe that the proposed model behaves comparable for binary malware detection tasks with either method. However, when only focusing on target class classification, the multi-class classification accuracy is slightly higher than the generalized classification scenario. This is expected because in the latter we do tackle a relatively less complex problem in which source classes are not included for classification. To conclude, we show that the proposed raw data-based Residual 1-D Image Transformer Model (R1DIT) can expand its classification accuracy to unseen malware classes using transfer learning of few-shot learning with only a few numbers of samples per each new class even they are encrypted with an unseen version of TLS 1.3 which we don't find any prior knowledge in the source dataset.

VI. Conclusions and Research Directions

Secure and reliable network connection is important for the continuity of the companies and government services. In this article, we discuss the current methods for malware traffic classification systems and show the potential of deep learning-based models with raw data input by proposing a Residual 1D Image Transformer (R1DIT) model that performs malware classification using network packets. A comprehensive set of experiments on NetML and CICIDS2017 datasets show that our privacy preserved model enhanced with protocol-specific processing outperforms the state-of-the-art work on classifying the malware in the network. We also evaluate its performance on a recently released unseen DDoS malware traffic encrypted with TLS 1.3 and show that the model can expand its ability to detect the new type of traffic with only a few samples per new class using meta-learning methods such as transfer learning or few-shot learning without compromising the accuracy in the source dataset. The slow inference speed brought on by the model complexity, however, is a serious limitation of this method. Our early analysis demonstrates that the multi-head attention method severely hinders performance, especially as the input size increases. Therefore, in the future, we will further optimize the R1DIT model to improve inference speed, enabling its application in the next-generation firewalls where high-speed inference is crucial due to the large traffic volume.

Acknowledgment

This work is supported in part by Intel Corporation and NSF No. 1738965.

References

[1] P. Prasse, L. Machlica, T. Pevný, J. Havelka, and T. Scheffer, "Malware detection by analysing network traffic with neural networks," in *2017 IEEE Security and Privacy Workshops (SPW)*, 2017, pp. 205–210.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[4] H. Huang, H. Deng, J. Chen, L. Han, and W. Wang, "Automatic multi-task learning system for abnormal network traffic detection," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 13, no. 04, pp. 4–20, 2018.

[5] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 445–458, 2019.

[6] R.-H. Hwang, M.-C. Peng, V.-L. Nguyen, and Y.-L. Chang, "An lstm-based deep learning approach for classifying malicious traffic at the packet level," *Applied Sciences*, vol. 9, no. 16, 2019.

[7] O. Barut, Y. Luo, T. Zhang, W. Li, and P. Li, "Netml: A challenge for network traffic analytics," 2020.

[8] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, P. Mori, S. Furnell, and O. Camp, Eds. SciTePress, 2018, pp. 108–116.

[9] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, pp. 1–8.

[10] A. Pektaş and T. Acarman, "A deep learning method to detect network intrusion through flow-based features," *International Journal of Network Management*, vol. 29, no. 3, p. e2050, 2019, e2050 nem.2050.

[11] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, "Deep learning approach for intelligent intrusion detection system," *IEEE Access*, vol. 7, pp. 41 525–41 550, 2019.

[12] H. Sun, Y. Xiao, J. Wang, J. Wang, Q. Qi, J. Liao, and X. Liu, "Common knowledge based and one-shot learning enabled multi-task traffic classification," *IEEE Access*, vol. 7, pp. 39 485–39 495, 2019.

[13] M. Gao, L. Ma, H. Liu, Z. Zhang, Z. Ning, and J. Xu, "Malicious network traffic detection based on deep neural networks and association analysis," *Sensors (Basel)*, vol. 20(5), no. 1452, 2020.

[14] C. Khammassi and S. Krichen, "A nsga2-lr wrapper approach for feature selection in network intrusion detection," *Computer Networks*, vol. 172, p. 107183, 2020.

[15] O. Barut, M. Grohotolski, C. DiLeo, Y. Luo, P. Li, and T. Zhang, "Machine learning based malware detection on encrypted traffic: A comprehensive performance study," in *7th International Conference on Networking, Systems and Security*, ser. 7th NSysS 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 45–55.

[16] D. Aksu, S. Üstebay, M. A. Aydin, and T. Atmaca, "Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm," in *Proceedings of the International Symposium on Computer and Information Science, ISCIS 2018*, T. Czachórski, E. Gelenbe, K. Grochla, and R. Lent, Eds. Springer, 2018, pp. 141–149.

[17] A. Yulianto, P. Sukarno, and N. A. Suwastika, "Improving adaboost-based intrusion detection system (ids) performance on cic ids 2017 dataset," *Journal of Physics: Conference Series*, vol. 1192, 2019.

[18] O. Barut, Y. Luo, T. Zhang, W. Li, and P. Li, "Multi-task hierarchical learning based network traffic analytics," in *ICC 2021 - IEEE International Conference on Communications*, 2021, pp. 1–6.

[19] C. F. T. Pontes, M. M. C. de Souza, J. J. C. Gondim, M. Bishop, and M. A. Marotta, "A new method for flow-based network intrusion detection using the inverse potts model," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1125–1136, 2021.

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

[21] G. Xie, Q. Li, Y. Jiang, T. Dai, G. Shen, R. Li, R. Sinnott, and S. Xia, "Sam: Self-attention based deep learning method for online traffic classification," in *Proceedings of the Workshop on Network Meets AI & ML*, ser. NetAI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 14–20.

[22] J. Yang and H. Lim, "Deep learning approach for detecting malicious activities over encrypted secure channels," *IEEE Access*, vol. 9, pp. 39 229–39 244, 2021.

[23] J. Zhao, S. Shetty, J. W. Pan, C. Kamhoua, and K. Kwiat, "Transfer learning for detecting unknown network attacks," in *EURASIP Journal on Information Security*, vol. 1, 2019.

[24] Z. Zhang, Q. Liu, S. Qiu, S. Zhou, and C. Zhang, "Unknown attack detection based on zero-shot learning," *IEEE Access*, vol. 8, pp. 193 981–193 991, 2020.

[25] C. Xu, J. Shen, and X. Du, "A method of few-shot network intrusion detection based on meta-learning framework," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3540–3552, 2020.

[26] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, "A hierarchical hybrid intrusion detection approach in iot scenarios," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–7.

[27] L. Yu, J. Dong, L. Chen, M. Li, B. Xu, Z. Li, L. Qiao, L. Liu, B. Zhao, and C. Zhang, "Pbcnn: Packet bytes-based convolutional neural network for network intrusion detection," *Computer Networks*, vol. 194, p. 108117, 2021.

[28] T. Liu, M. Chen, M. Zhou, S. S. Du, E. Zhou, and T. Zhao, "Towards understanding the importance of shortcut connections in residual networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[29] G. S. Dhillon, P. Chaudhari, A. Ravichandran, and S. Soatto, "A baseline for few-shot image classification," in *International Conference on Learning Representations*, 2020.

[30] Stratosphere, "Stratosphere laboratory datasets," 2015, [Online; accessed 12-March-2020]. [Online]. Available: https://www.stratosphereips.org/datasets-overview

Dr. Onur Barut is Data Scientist and AI Software Architect in Network and Edge Group (NEX) of Intel Corporation. He has obtained his PhD in Computer Engineering from UMass Lowell in 2022. His interest and research fields are machine learning applications on computer vision, human activity recognition, network traffic analytics and malware detection. Before joining Intel, he was flight test instrumentation engineer at Aselsan, Turkey. He earned B.Sc. and M.Sc. degrees from Middle East Technical University, Turkey, both in Electrical and Electronics Engineering.

Dr. Yan Luo is a professor of the Department of Electrical and Computer Engineering at UMass Lowell. He obtained his PhD in Computer Science from University of California Riverside in 2005. His current research interest centers on machine/deep learning based data analytics of network flows and biomedical data, heterogeneous computing, and embedded sensor systems. He has served on the editorial board of several peer-reviewed journals and as a program chair of several IEEE conferences. Dr. Luo directs the laboratory of Advanced Computer Architecture and Network Systems (ACANETS) at UMass Lowell. He is a member of IEEE and ACM.

Dr. Peilong Li is an Assistant Professor of the Department of Computer Science at Elizabethtown College. He obtained his Ph.D. degree in Computer Engineering from UMass Lowell in 2016. His research interests include heterogeneous and parallel computer architecture, big data analytics with distributed computing framework, and data plane innovation in software defined networking

Dr. Tong Zhang is a Principal Engineer and Lead Architect on AI and analytics in the Network and Edge Group (NEX) of Intel Corporation. She has been leading the architecture efforts for Machine Learning and Deep Learning solutions for network domain use cases. Before joining Intel, Dr. Zhang was a Principal Scientist in Hewlett-Packard Labs where she led over a dozen projects related to media analytics systems and products. She has over 50 granted US patents and more than 70 publications to her credit. She earned B.S. degree from Tsinghua University and Ph.D. degree from University of Southern California, both in Electrical Engineering.