Adversarial Network Traffic: Towards Evaluating the Robustness of Deep-Learning-Based Network Traffic Classification

Amir Mahdi Sadeghzadeh, Saeed Shiravi, and Rasool Jalili

Abstract—Network traffic classification is used in various applications such as network traffic management, policy enforcement, and intrusion detection systems. Although most applications encrypt their network traffic and some of them dynamically change their port numbers, Machine Learning (ML) and especially Deep Learning (DL)-based classifiers have shown impressive performance in network traffic classification. In this article, we evaluate the robustness of DL-based network traffic classifiers against Adversarial Network Traffic (ANT). ANT causes DL-based network traffic classifiers to predict incorrectly using Universal Adversarial Perturbation (UAP) generating methods. Since there is no need to buffer network traffic before sending ANT, it is generated live. We partition the input space of the DL-based network traffic classification into three categories: packet classification, flow content classification, and flow time series classification. To generate ANT, we propose three new attacks injecting UAP into network traffic. AdvPad attack injects a UAP into the content of packets to evaluate the robustness of packet classifiers. AdvPay attack injects a UAP into the payload of a dummy packet to evaluate the robustness of flow content classifiers. AdvBurst attack injects a specific number of dummy packets with crafted statistical features based on a UAP into a selected burst of a flow to evaluate the robustness of flow time series classifiers. The results indicate injecting a little UAP into network traffic, highly decreases the performance of DL-based network traffic classifiers in all categories.

Index Terms—Network traffic classification, adversarial network traffic, deep learning, adversarial example, adversarial machine learning.

I. INTRODUCTION

IN RECENT years, Internet traffic has grown due to the emergence of new applications and services in market. Hence, management of network traffic is more challenging than ever and the necessity of network traffic classification is obvious in many cases, such as quality of service provision, resource usage management, billing in ISPs, anomaly detection, and Policy Enforcement systems. So far, three main approaches to network traffic classification have been introduced, including port-based, payload-based, and ML-based

classification [1]. Although port-based and payload-based approaches have prevailed in many devices, an adversary can use random port numbers or well-known port numbers to evade port-based classifiers [2], and encrypt his/her network traffic or obfuscate the pattern in the content of packets to evade payload-based classifiers [1], [2].

ML-based approaches utilize ML algorithms for network traffic classification. The byte sequence and statistical features of packets or flows are the most common features that are used by ML-based network traffic classifiers. Although ML-based approaches have been put forward with claims of fixing the problems of previous approaches, the robustness of ML-based classifiers has not been evaluated. In this article, we will investigate the robustness of such classifiers in a comprehensive way. In recent years, Deep Neural Networks (DNNs), also called Deep Learning (DL), have shown great success in solving complex problems such as image classification [3], and speech recognition [4]. These advances have motivated the researchers to use DNNs in other domains, and recently, researchers have demonstrated DNNs have promising results in network traffic classification [5]–[10]. In this article, we focus on DL-based network traffic classifiers, which are state of the art in ML-based network traffic classification.

Based on previous studies, we divide the input space of DL-based network traffic classification into three categories: Packet Classification (PC), Flow Content Classification (FCC), and Flow Time Series Classification (FTSC). In PC, the byte sequence of a packet is given to a classifier, and each packet is labeled. In FCC, the byte sequence of the first n packets of a flow is fed to a classifier, and the class of each flow is predicted. In FTSC, the sequence of statistical features of the first m packets of a flow, such as packets size and inter-arrival times between packets, are passed to a classifier, and each flow is classified. We consider two classifiers in each input space category to assess the influence of various items in the input of classifiers. In this article, we use One-Dimensional Convolutional Neural Networks (1D-CNN) to classify ISCXVPN2016 dataset [11] in which multiple application, including Skype, Facebook, and Hangouts generate network traffic with various types. The target of network traffic classification in this study is traffic characterization in which the type of network traffic such as VoIP, streaming, email, and chat is determined.

After the success of DNNs in various classification tasks, their robustness has become the subject of much

debate [12]–[15]. In 2014, Szegedy *et al.* [12] have demonstrated that DNNs are vulnerable to adversarial examples, which are maliciously crafted inputs that cause a classifier to make a mistake. This vulnerability questioned the robustness of DL-based classifiers in adversarial environment in which an adversary can interfere in the training and the prediction phases. In this article, the robustness of DL-based network traffic classifiers is challenged against Adversarial Network Traffic (ANT). ANT is adversarially crafted network traffic in which the content or the statistical features of network traffic is perturbed.

There are two serious constraints to generate ANT. First, in some types, such as VoIP, email, and chat, the content and the statistical features of a flow are generated over time and are dependent to the user, and there is no access to the entire flow at the beginning of it. Second, a flow is generated by two entities, and neither of them knows the content and the statistical features of entire flow to make adversarial perturbation based on it. According to these constraints, we use Universal Adversarial Perturbation (UAP) generating methods to generate ANT. Moosavi-Dezfooli *et al.* [16] demonstrate that there is a perturbation that, if it is added to a set of data, it will cause DNNs to make incorrect predictions on most of them, and this perturbation is not unique. In ANT, UAP is generated on a pre-collected set of network traffic, and whenever a new flow or a new packet is being sent, the pre-made UAP is injected into it. This approach does not need to have access to the entire data to make perturbation, and UAP is made beforehand on a pre-collected set of network traffic. Consequently, ANT can be generated live, and there is no need to buffer target network traffic.

In this article, three new attacks for generating ANT are proposed, including Adversarial Pad (AdvPad) attack, Adversarial Payload (AdvPay) attack, and Adversarial Burst (AdvBurst) attack. AdvPad injects a UAP into the end or the start of packets payload to evaluate the robustness of packet classifiers. AdvPay injects a UAP into the payload of a dummy packet at the first n packets of a flow to evaluate the robustness of flow content classifiers. AdvBurst injects a specific number of dummy packets with crafted statistical features based on a UAP into the end of a selected burst of a flow to evaluate the robustness of flow time series classifiers. Experiments show that all classifiers are vulnerable to ANT, and by injecting a little UAP into network traffic, the performance of DL-based network traffic classifiers highly decreases. Results demonstrate that the researchers should investigate the methods for improving the robustness of DL-based network traffic classifiers.

The main contributions of this article are as follows:
- Network traffic is formally defined, including packets, unidirectional flows, bidirectional flows, and three input space categories of DL-based network traffic classifiers.
- The influence of input space categories on the performance and the robustness of DL-based network traffic classifiers is investigated.
- Adversarial Network Traffic (ANT) is proposed to evaluate the robustness of DL-based network traffic classifiers.

- Three new attacks are proposed to generate ANT, including AdvPad, AdvPay, and AdvBurst attacks.

The rest of this article is organized as follows. In Section II, network traffic, and three input space categories of DL-based network traffic classification are formally defined. Also, the DNNs and basic methods for generating adversarial examples are introduced. Section III reviews related works on DL-based network traffic classification. In Section IV, ANT will be presented, and AdvPad, AdvPay, and AdvBurst attacks are proposed. Section V evaluates the robustness of classifiers in three input space categories against three proposed attacks. Section VI analyses the results of experiments. Section VII discusses how to use ANT in practice. Finally, in Section VIII, the conclusions of this work will be discussed.

II. BACKGROUND

Network traffic, DL-based classifiers, and adversarial examples are the fundamental constituents of the DL-based network traffic classification and ANT. Accordingly, we introduce these ingredients in the following subsections.

A. Formal Specification of Network Traffic

Network traffic consists of bidirectional flows and packets, which are the common objects for network traffic classification. Each packet consists of multiple headers and a payload in which application data exists. We indicate the headers of a packet by 5-tuple information, which stems from the content of the IP layer and Transport Layer (TL) headers.

Definition 1: An IP packet P_i is a sequence of an IP layer header, a TL header, and a payload.

$$
\begin{aligned}
P_i &= <H_{p_i}^{IP}, H_{p_i}^{TL}, Pay_{p_i}>, \\
H_{P_i}^{IP} &= \left(IP_{P_i}^{src}, IP_{P_i}^{dst} \right), \\
H_{P_i}^{TL} &= \left(Port_{P_i}^{src}, Port_{P_i}^{dst}, Proto_{p_i} \right), \quad (1)
\end{aligned}
$$

where $IP_{P_i}^{src}$ and $IP_{P_i}^{dst}$ indicate source and destination IP addresses of the packet, respectively. $Port_{P_i}^{src}$ and $Port_{P_i}^{dst}$ are source and destination ports of the transport layer, respectively, and $Proto_{p_i}$ is the transport layer protocol of the packet. Unidirectional flow is a set of packets that share the same source and destination information and transport layer protocol.

Definition 2: A unidirectional flow UF_i is a set of an IP layer header, a TL header, a timeout value, and a sequence of packets.

$$
\begin{aligned}
UF_i &= \left\{ H_{UF_i}^{IP}, H_{UF_i}^{TL}, P_{UF_i}, TimeOut_{UF_i} \right\}, \\
H_{UF_i}^{IP} &= \left(IP_{UF_i}^{src}, IP_{UF_i}^{dst} \right), \\
H_{UF_i}^{TL} &= \left(Port_{UF_i}^{src}, Port_{UF_i}^{dst}, Proto_{UF_i} \right), \\
P_{UF_i} &= <P_1^i, P_2^i, \ldots, P_m^i>, \\
s.t. \quad H_{P_j^i}^{IP} &= H_{UF_i}^{IP}, \ H_{P_j^i}^{TL} = H_{UF_i}^{TL}, j \in [1, m], \\
IAT_{P_{k+1}^i, P_k^i} &\leq TimeOut_{UF_i}, k \in [1, m-1], \quad (2)
\end{aligned}
$$

where m is the number of packets, $IAT_{P_{k+1}^i, P_k^i}$ is inter-arrival time between k and $k+1$ packets of the packet sequence P_{UF_i} in the unidirectional flow UF_i. $IP_{UF_i}^{src}$ and $IP_{UF_i}^{dst}$ are source and destination IP addresses of UF_i, respectively. $Port_{UF_i}^{src}$ and $Port_{UF_i}^{dst}$ are source and destination ports of the transport layer, respectively, and $Proto_{UF_i}$ is the transport layer protocol of UF_i. A bidirectional flow is the union of two unidirectional flows, where IP addresses and ports have switched.

Definition 3: A bidirectional flow BF_i is the union of two opposite unidirectional flows UF_j and UF_k.

$$BF_i = UF_j \cup UF_k,$$
$$s.t. \quad IP_{UF_j}^{src} = IP_{UF_k}^{dst}, \; IP_{UF_j}^{dst} = IP_{UF_k}^{src},$$
$$Port_{UF_j}^{src} = Port_{UF_k}^{dst}, \; Port_{UF_j}^{dst} = Port_{UF_k}^{src},$$
$$Proto_{UF_j} = Proto_{UF_k}, \; TimeOut_{UF_j} = TimeOut_{UF_k}. \tag{3}$$

We focus on bidirectional flow in this article and use flow instead of bidirectional flow in the rest of this article for simplicity.

B. ML-Based Network Traffic Classification

Classification is a subset of supervised learning that categorizes a set of data into classes. A ML-based classifier is a function f which maps an input space $\mathcal{X}$ to an output space $\mathcal{Y} = \{y_1, \ldots, y_k\}$ where y_i is i^{th} class, and k is the number of classes. In the training phase of a classifier, we need to have a training set in which the true class of each sample has been given $\{(x_i, y_i)\}_{i=1}^n$, where n is the number of samples in the training set and y_i is the true class (label) of the sample x_i. A good classifier generalizes to unseen samples in the test set and classifies them correctly.

According to the previous studies, network traffic classification can be used in several domains, including protocol detection, application identification, website and video fingerprinting, and traffic characterization. In this article, we focus on traffic characterization, which specifies the type of network traffic, such as VoIP, streaming, file transferring, and email. Traffic characterization is fundamental in many scenarios, such as network traffic accounting, quality of service providing, and policy enforcement.

We partition the input space of network traffic classification into three categories: (*i*) packet classification (PC), (*ii*) flow content classification (FCC), and (*iii*) flow time series classification (FTSC). In what follows, each input space category will be explained. Since the information in the header of the IP layer is dependent on the machines that generate network traffic, we do not consider IP header for network traffic classification.

Packet Classification (PC): In packet classification, the byte sequence of a packet is given to a classifier, and it labels each packet separately. Based on the literature, we consider two versions of packet classification. In the first version (PC-HP), TL header and payload, and in the later version (PC-P), only payload of a packet is given to classifier. For packet P_i, we have:

$$\text{PC-HP Input for P}_i: \; <H_{p_i}^{TL}, Pay_{p_i}>,$$
$$\text{PC-P Input for P}_i: \; <Pay_{p_i}>. \tag{4}$$

Flow Content Classification (FCC): In this category, the byte sequence of the first n packets of a flow is given to a classifier, and each flow is labeled. FCC has two versions. In the first version (FCC-HP), TL headers and payloads, and in the second one (FCC-P), only payloads of the first n packets of a flow are given to a classifier. To improve the performance of classifiers, we multiply the sign of packet direction to the content of that packet. The direction sign is considered positive ($+1$) for packets from source (e.g., client) to destination (e.g., server) and negative (-1) for packets from destination to source. For flow F_i, we have:

$$\text{FCC-HP Input for F}_i: \; <H_{p_0^i}^{TL} \times D_{p_0^i}, Pay_{p_0^i} \times D_{p_0^i}, \ldots,$$
$$H_{p_{n-1}^i}^{TL} \times D_{p_{n-1}^i}, Pay_{p_{n-1}^i} \times D_{p_{n-1}^i}>,$$
$$\text{FCC-P Input for F}_i: \; <Pay_{p_0^i} \times D_{p_0^i}, \ldots, Pay_{p_{n-1}^i} \times D_{p_{n-1}^i}>, \tag{5}$$

where $D_{p_j^i} \in \{+1, -1\}$ is j^{th} packet direction of flow F_i.

Flow Time Series Classification (FTSC): In this category, the statistical features of the first m packets of a flow are given to a classifier. Two versions of FTSC have been considered in the previous studies. In the first version (FTSC-IAT), inter-arrival times between packets of a flow, and in the second version (FTSC-PS), packets sizes of a flow are given to a classifier in time-series format. Similar to FCC, we multiply the sign of packets direction to the statistical features of packets to improve the performance of the classifiers. For flow F_i, we have:

$$\text{FTSC} - \text{IAT Input for F}_i: \; <IAT_{(p_0^i, p_1^i)} \times D_{p_1^i}, IAT_{(p_1^i, p_2^i)}$$
$$\times D_{p_2^i}, \ldots, IAT_{(p_{m-2}^i, p_{m-1}^i)}$$
$$\times D_{p_{m-1}^i}>,$$
$$\text{FTSC} - \text{PS Input for F}_i: \; <PS_{p_0^i} \times D_{p_0^i}, PS_{p_1^i} \times D_{p_1^i},$$
$$\ldots, PS_{p_{m-1}^i} \times D_{p_{m-1}^i}>, \tag{6}$$

where $IAT_{(p_j^i, p_k^i)}$ is inter-arrival time between j^{th} and k^{th} packets, and $PS_{p_j^i}$ is the size of jth packet in flow F_i.

C. Deep Neural Networks

Deep neural networks (DNNs) are a function $y_i = f_{DNN}(x_i)$ which takes an input $x_i \in \mathcal{X}$ and returns an output $y_i \in \mathcal{Y}$. DNNs act on raw data and do not need feature engineering. Therefore, there is no need to have feature selection or an expert who extracts the most salient features of network traffic. Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Stacked Denoising Autoencoders (SDAE) are the three main DNNs which have been used in network traffic classification. Previous investigations [7], [8] have demonstrated that the One-Dimensional Convolutional Neural Networks (1D-CNNs) have delivered the best performance in network traffic classification. In this article, we use 1D-CNN to classify network traffic.

Each neural network includes an input layer, multiple hidden layers, and an output layer. Each layer consists of multiple neurons that are connected to neurons in the adjacent layers. Except for the input layer, the output of a given layer is calculated through applying a nonlinear function on an obtained value from multiplying the output of previous layer by the weights matrix of that layer. A DNN assigns the label $\hat{y} = \mathrm{argmax}_{0 \leq i < k} f_{DNN}(x)_i$ to the input x, where $f_{DNN}(x)_i$ is the ith element of DNN output. A convolutional layer uses filters that convolve on the input of the layer to extract local features. A deep convolutional neural network consists of multiple convolutional layers at the start of the network to extract features of the input and a fully connected neural network that uses extracted features by the convolutional layers to classify input data. In the training phase, weights matrix of all layers θ are adjusted to minimize a loss function J which measures the distance between predicted label $\hat{y}$ and the true label y. Hence, training is a minimization problem on the weights of classifier. The output of the training phase is θ^* that minimizes the loss function J. Stochastic gradient descent (SGD) and its versions are used for the minimization of loss function in DNNs. Previous studies have used batch normalization, dropout, and polling layers to improve the performance of DNNs in network traffic classification. (For more information, see [17]).

D. Adversarial Example

Although DNNs have achieved great success in solving complex problems, they demonstrate serious vulnerability against adversarial examples [12]. An adversarial example is a crafted input which causes the target classifier to make a mistake. Suppose that a classifier f^* assigns the true label to each data and the target classifier of adversarial example attack is f. For an adversarial example x', we have:

$$\begin{aligned} f^*(x') &= y, \\ f(x') &= y', \quad s.t.\ y' \neq y. \end{aligned} \tag{7}$$

The main method for creating an adversarial example is adding an adversarial perturbation ξ to a real data x. In this method, it is supposed that after adding perturbation to data x, the true class of $x' = x + \xi$ and x must be the same. Given that a large perturbation can change the true class of data, the amount of perturbation has to be limited in many contexts such as, image classification. The most common distant function is used in the literature is $L_p = \|x' - x\|_p$ so that P-norm $\|.\|_p$ is defined as:

$$\|d\|_p = \left(\sum_{i=0}^{D-1} |d_i|^p \right)^{1/p} \tag{8}$$

where D is the dimension of data d and $p \in [0, \infty]$. Generation of an adversarial example can be formulated as a minimization problem:

$$\begin{aligned} &\arg\min_{x'}\ \|x' - x\|_p \\ &s.t.\ f^*(x') = y,\ f(x') = y',\ y' \neq y,\ x' \in Domain(x). \end{aligned} \tag{9}$$

Goodfellow *et al.* [14] propose Fast Gradient Sign Method (FGSM) attack to solve the minimization problem. The authors use the gradient of DNN loss function to compute adversarial perturbation, and L_∞ to limit the size of perturbation. In FGSM, adversarial perturbation is calculated as:

$$\xi = \epsilon . sign(\nabla_x J(\theta, x, y)) \tag{10}$$

where $\nabla_x J(\theta, x, y)$ is the gradient of DNN loss function with respect to input x.

Moosavi-Dezfooli *et al.* [16] introduce the Universal Adversarial Perturbation (UAP) attack, which is different from previous attacks. In previous attacks, a perturbation is made for each data. However, the UAP attack adds a single perturbation ξ^u to a set of data and hops that they would be misclassified with a high probability.

To generate UAP, the authors used an iterative algorithm in which iteration $t > 0$, a data x is sampled from $\mathcal{X}$ and ξ^u_{t-1} is added to the data x. Then, the perturbation r is calculated using the following equation ($\xi^u_0 = 0$).

$$\begin{aligned} r_t &= \arg\min_r\ \|r\|_2 \\ &s.t.\quad f\left(x + \xi^u_{t-1} + r\right) \neq f(x). \end{aligned} \tag{11}$$

To calculate ξ^u_t, r_t is added to ξ^u_{t-1} in each iteration, and a projection function $\mathcal{P}_{p,\epsilon}(v)$ is used which maps the input v to a L_p ball of radius ϵ and is centered at zero to ensure that the constraint $\|\xi^u\|_p < \epsilon$ is satisfied. Therefore:

$$\xi^u_t = \mathcal{P}_{p,\epsilon}\left(\xi^u_{t-1} + r_t\right). \tag{12}$$

The authors have demonstrated that there is no need to have much data to generate universal adversarial perturbation, and this perturbation is not unique. In another work, Brown *et al.* [15] propose universal adversarial image patch attack, which injects a patch into an image and causes the classifier to predict wrongly.

III. RELATED WORKS

There are a few studies that use adversarial machine learning approaches to evade ML-based network traffic classifiers. Verma *et al.* [18] utilize Carlini and Wanger adversarial example generating method [13] and a fully connected neural network as the target classifier. They use packet size and inter-arrival time features to train the target classifier for traffic characterization. Usama *et al.* [19] propose a new method for generating adversarial examples using Mutual Information (MI) criterion. They use MI to discover the most discriminative features to perturb them. The authors consider a support vector machine and a fully connected neural network as the target classifiers for traffic characterization. The target classifiers are trained on more than 80 statistical features for each flow, such as IAT mean, PSH flag count, forward packets per second. Previous studies [18], [19] do not discuss the magnitude of overhead that their methods impose on network traffic, and their methods are not universal. Rahman *et al.* [20] introduce Mockingbird to evade website fingerprinting classifiers. They focus on Deep Fingerptinitng (DF) [8] as the target classifier. The sequence of packets direction called trace is used as the

TABLE I
OVERVIEW OF PREVIOUS STUDIES

Paper	PC	FCC	FTSC	Input data	Classifier	Classification target	Year
P. Wang et al. [21]	●	○	○	H^{IP}, H^{TL}, payload	SDAE, MLP, CNN	Application identification	2018
M. Lotfollahi et al. [10]	●	○	○	H^{TL}, payload	SAE, CNN	Application identification, Traffic characterization	2020
Z. Wang et al. [22]	○	●	○	H^{TL}, payload	SAE	Protocol detection	2015
W. Wang et al. [23]	○	●	○	H^{IP}, H^{TL}, payload	CNN	Traffic characterization	2017
S. Rezaie et al. [24]	○	●	●	H^{IP}, H^{TL}, payload, IAT, PS, Dir	CNN, LSTM	Application identification	2020
G. Aceto et al. [7]	○	●	●	H^{IP}, H^{TL}, payload, IAT, PS, Dir	SAE, CNN	Application identification	2019
J. Caicedo-Munoz et al. [6]	○	○	●	Time-related features	Bagging, Boosting	Traffic characterization	2018
G. Draper-Gi et al. [11]	○	○	●	Time-related Features	KNN, C4.5	Traffic characterization	2016
M. López Martín et al. [5]	○	○	●	PS, Dir, H^{TL}	LSTM, CNN	Protocol detection	2017
K. Abe And S. Goto [25]	○	○	●	Dir, IAT	SDAE	Website fingerprinting	2016
V. Rimmer et al. [9]	○	○	●	Dir	SDAE, CNN, LSTM	Website fingerprinting	2018
P. Sirinam et al. [8]	○	○	●	Dir	CNN	Website fingerprinting	2018

input of DF. Mockingbird starts with selecting a source and a target trace with different classes. It then gradually changes the source trace to become close to the target trace until the detector is deceived. Mockingbird decreases the accuracy of DF to 35.2% with almost 56% bandwidth overhead. Mockingbird is not universal and only works on flow time series classifiers.

The DL-based network traffic classification has been widely studied in the literature. A brief review of previous studies is provided in separate subsections and is summarized in Table I.

A. Packet Classification

Lotfollahi et al. present Deep Packet framework that embeds SDAE and CNNs to classify ISCXVPN2016 network traffic dataset [10]. The Deep Packet uses the byte sequence of the transport layer header and the payload of packets to classify packets. The results show that 1D-CNN classifier has better overall performance than SDAE in both application identification and traffic characterization, and the overall accuracy of 1D-CNN in application identification is 98%, while the precision in traffic characterization is 93%. Wang et al. introduce a DL-based encrypted traffic classifier called Datanet for better management of distributed smart home networks [21]. In this study, after packets pre-processing and removing the header of the Data-link layer, the byte sequence of packets are fed into Datanet. Datanet uses ISCXVPN2016 dataset and gains F-measure $\geq$ 96% for both the SDAE and CNN in distinguishing 15 applications.

B. Flow Content Classification

The first attempt to apply deep learning in network traffic classification is reported by Wang [22]. They recognize remarkable feature engineering ability of deep learning and try to detect protocols in a TCP flow dataset by means of SDAE. Their dataset consists of 300k pre-processed records in which exist 1000 bytes of TCP payload. They achieve more than 90% recall in protocol detection for all protocols. Wang et al. propose an end-to-end encrypted traffic classifier to traffic

characterization [23]. They use ISCXVPN2016 dataset and extract only 728 bytes of each flow as the input. They achieve the best result in the classification of the ISCXVPN2016 dataset compared to previous works that use the classical ML algorithms.

After several studies in the domain of mobile application identification [26], [27], Aceto et al. address mobile application identification through the Deep Learning approach for the first time [7]. They investigate type of input data that is fed into the DL-classifier and consider three types of input data: (i) first N bytes of payload in a flow, (ii) first N bytes of all headers and payloads in a flow, and (iii) source and destination ports, number of bytes in the transport-layer header, TCP window size, inter-arrival time, and direction of packets in a flow. As the best result, they achieve 83% overall accuracy for identifying 45 applications in both Android and iOS operating systems on the first type of input data. Also, they demonstrate that the overall performance of 1D-CNN and 2D-CNN are close in mobile application identification and suggest that traffic should be extracted by naturally considering data as one-dimensional. In a similar work, Rezaei et al. propose a new DL-based classifier to identify network traffic of mobile applications [24]. Their proposed classifier only needs the payload and statistical features of the first few packets of flows and achieves between 84% to 98% overall accuracy for the identification of 80 popular mobile applications.

C. Flow Time Series Classification

In [11], a set of time-related features such as duration of flow, byte per second, and inter-arrival times between packets is utilized to network traffic classification. They use C4.5 and K-nearest neighbors as classification techniques and achieve about 80% accuracy in the characterization of network traffics. Moreover, they generate and distribute a labeled dataset of encrypted network traffic is called ISCXVPN2016, which has become a criterion in network traffic classification. In [6], authors improve the performance of [11], and

achieve about 83% accuracy on ISCXVPN2016 dataset using bagging and boosting classifiers. Martin *et al.* [5] introduce a new hybrid classifier based on CNN and RNN. In this work, each flow comprises 20 packets. Regarding their results, whenever an RNN is combined with a CNN, success rate slightly improves.

The process of identifying network traffic of visited websites through privacy-enhancing technologies like Tor, which is also known as Website Fingerprinting (WF), has a background as long as encrypted traffic classification. Abe and Goto investigate the effectiveness of DL-based classifiers in WF for the first time [25]. In their experiment, network traffic of 100 websites that passed through Tor is monitored, and the statistical feature of each flow is extracted. They reveal that SDAE is useful in the detection websites only using direction and inter-arrival times of cells (Tor packets) with 86% success rate. Rimmer *et al.* indicate that DL is an effective tool in automating the process of features engineering, and their SDAE achieves 95.3% success rate only using direction of Tor cells [9]. In another work, Sirinam *et al.* design a deeper CNN classifier to outperform earlier studies, which increase the success rate to 98% [8].

IV. ADVERSARIAL NETWORK TRAFFIC

Adversarial Network Traffic (ANT) evaluates the robustness of DL-based network traffic classifiers, using the concept of adversarial example attack. In adversarial example attack, an adversary adds a little perturbation into the input data and causes a classifier to make a false prediction. In the same approach to deceive DL-based network traffic classifiers, ANT injects a perturbation into network traffic. Most of adversarial example attacks have been proposed in the context of image processing, and they are not directly applicable in the context of network traffic classification. There are some constraints to apply adversarial example attacks in the context of network traffic classification.

1) The content of packets must be preserved. If the content of a packet is modified, the functionality of the application that uses the packet is disrupted. Hence, we are not allowed to perturb the content of packets. The perturbation can only be injected into some specific parts of the input, such as the end of packets, or the content of a dummy packet.
2) In conventional machine learning tasks, like image classification, adversarial perturbation is made based on the entire data. However, two entities make entire data in the traffic classification task, and neither of them has access to the entire data.
3) In some network traffic types, like chat, email, and VoIP, the application that generates network traffic is not aware of the content of network traffic at the beginning of a flow, and the content of network traffic is dependent on the user. Hence, there is no access to the entire flow when the initial packets are being processed.
4) Making a perturbation for each packet or flow has high computational overhead.

Based on such constraints, ANT uses Universal Adversarial Perturbation (UAP) to cause the classifier to predict wrongly. In this approach, there is no need to have access to the entire new data to make adversarial perturbation, and there is no need to compute one perturbation for each input. Also, UAP is independent of the content of target data and is made on a pre-collected set of data. To build UAP, first, a set of flows or packets of the class from which we want to make UAP is collected, and after crafting UAP, it is injected into the new incoming network traffic of that class. For example, at the first step, VoIP traffic are collected, then a UAP is made using a pre-collected set of VoIP traffic. Afterwards, when a new VoIP flow or packet is being sent, UAP is injected into it. Based on the category of input space, the method of generating ANT and applying UAP is different. Hence, we propose three new attacks to evaluate the robustness of network traffic classifiers which are explained afterwards.

A. Adversarial Pad Attack

We design adversarial pad (AdvPad) attack to reduce the performance of two packet classifiers PC-HP, and PC-P. AdvPad injects a UAP into the specific location of packets payload. We only consider the start and the end of the payload of packets for injecting adversarial pad and call them Start_AdvPad and End_AdvPad Attacks, respectively. If UAP injects into the start of the payload, the structure of an adversarially padded packet P_i is as follows:

$$PC\text{-HP Input for } P_i: <H_{P_i}^{TL}, AdvPad, Pay_{P_i}>,$$
$$PC\text{-P Input for } P_i: < AdvPad, Pay_{P_i}>. \quad (13)$$

If UAP injects into the end of the payload, the structure of an adversarially padded packet P_i is as follows:

$$PC\text{-HP Input for } P_i: <H_{P_i}^{TL}, Pay_{P_i}, AdvPad>,$$
$$PC\text{-P Input for } P_i: <Pay_{P_i}, AdvPad>. \quad (14)$$

Algorithm 1 shows the process of making UAP for the AdvPad attack. This algorithm has T iterations, which in each iteration, a batch of packets are sampled from packets set P. All packets in P has the same label l. For i^{th} packet in the batch of packets p^{batch}, UAP ξ is injected into the start or the end of the payload, which the UAP location is specified by Loc_{AdvPad} parameter. A bandwidth overhead parameter *OH* is considered to control the bandwidth overhead of AdvPad attack. *OH* determines the percentage of bandwidth overhead that we want to add to each packet. Based on this parameter, we add the first *pad_size* bytes of ξ to a packet. In each iteration, the perturbation ξ is updated based on the gradient of loss function with respect to ξ, and ϵ controls the magnitude of changes in ξ. The function $Clip_{Domain(P)}(x)$ maps the input x to the packets domain $Domain(P)$ by clipping features that are not included.

B. Adversarial Payload Attack

Adversarial payload (AdvPay) attack aims to deceive two flow content classifiers FCC-HP, and FCC-P, through adding UAP to the payload of a dummy packet. In AdvPay attack,

---

Algorithm 1 Adversarial Pad

Input Packet set P of class l, packet classifier f_{1D-CNN} with weights θ, location of UAP Loc_{AdvPad}, overhead percentage OH, perturbation rate ϵ, number of iterations T, batch size $batch\_size$.

Output UAP ξ

1: $\xi \leftarrow Rand(Domain(P), Size = MaxPktSize)$
2: **for** $t \leftarrow 0, T$ **do**
3:     $P^{batch} \leftarrow sample\ batch\_size\ packets\ from\ P$
4:     **for** $i \leftarrow 0, batch\_size$ **do**
5:        $pad\_size \leftarrow sizeof(p_i^{batch}) \times OH/100$
6:        $\xi' \leftarrow \xi[0 : pad\_size]$
7:        **if** $Loc_{AdvPad} == Start$ and $func == PC - HP$ **then**
8:           $pkt\_byte\_seq_i^{batch} \leftarrow < H_{p_i^{batch}}^{TL}, \xi', Pay_{p_i^{batch}} >$
9:        **else if** $Loc_{AdvPad} == Start$ and $func == PC - P$ **then**
10:         $pkt\_byte\_seq_i^{batch} \leftarrow < \xi', Pay_{p_i^{batch}} >$
11:       **else if** $Loc_{AdvPad} == End$ and $func == PC - HP$ **then**
12:         $pkt\_byte\_seq_i^{batch} \leftarrow < H_{p_i^{batch}}^{TL}, Pay_{p_i^{batch}}, \xi' >$
13:       **else if** $Loc_{AdvPad} == End$ and $func == PC - P$ **then**
14:         $pkt\_byte\_seq_i^{batch} \leftarrow < Pay_{p_i^{batch}}, \xi' >$
15:       **end if**
16:     **end for**
17:     $\Delta\xi \leftarrow \epsilon \times \nabla_\xi J(\theta, pkt\_byte\_seq^{batch}, l)$
18:     $\xi \leftarrow Clip_{Domain(P)}(\xi + \Delta\xi)$
19: **end for**
20: **return** ξ

Algorithm 2 Adversarial Payload

Input Flow set F of class l, flow content classifier f_{1D-CNN} with weights θ, dummy packet index vector IND_{Advpay}, UAP size $AdvPay_{Size}$, perturbation rate ϵ, number of iterations T, batch size $batch\_size$.

Output UAP ξ

1: $\xi \leftarrow zeros(Size = AdvPay_{Size})$
2: **for** $t \leftarrow 0, T$ **do**
3:     $F^{batch} \leftarrow sample\ batch\_size\ flows\ from\ F$
4:     $IND^{batch} \leftarrow dummy\ packet\ index\ of\ F^{batch}\ in\ IND_{Advpay}$
5:     **for** $i \leftarrow 0, batch\_size$ **do**
6:        $f\_byte\_seq_i^{batch} \leftarrow <>$
7:        **for** $j \leftarrow 0, n - 1$ **do**
8:           **if** $j < IND_i^{batch}$ **then**
9:             $append(f\_byte\_seq_i^{batch}, < P_j^{F_i^{batch}} \times D_j^{F_i^{batch}} >)$
10:          **else if** $j == IND_i^{batch}$ **then**
11:             $append(f\_byte\_seq_i^{batch}, <$
                 $dummy\ packet\ with\ payload\ \xi \times D_{j-1}^{F_i^{batch}} >)$
12:          **else if** $j > IND_i^{batch}$ **then**
13:             $append(f\_byte\_seq_i^{batch}, < P_{j-1}^{F_i^{batch}} \times D_{j-1}^{F_i^{batch}} >)$
14:          **end if**
15:        **end for**
16:     **end for**
17:     $\Delta\xi = \epsilon \times \nabla_\xi J(\theta, f\_byte\_seq^{batch}, l)$
18:     $\xi = Clip_{Domain(F)}(\xi + \Delta\xi)$
19: **end for**
20: **return** ξ

---

a dummy packet is injected into a specified location $k \in [0, n - 1]$ among the first n packets of a flow, and UAP is injected into it. The direction of the dummy packet can be arbitrary; nonetheless, we choose the direction of the last packet before the dummy Packet $D_{p_{k-1}^i}$ as the direction of the dummy packet. The input of FCC-HP, and FCC-P for flow F_i are changed as follows:

FCC-HP Input for F_i:

$$< H_{P_0^i}^{TL} \times D_{p_0^i}, Pay_{P_0^i} \times D_{p_0^i}, \ldots, H_{P_k^i}^{TL} \times D_{p_{k-1}^i}, AdvPay_{P_k^i}$$
$$\times D_{p_{k-1}^i}, \ldots, H_{P_{n-2}^i}^{TL} \times D_{p_{n-2}^i}, Pay_{P_{n-2}^i} \times D_{p_{n-2}^i} >,$$

FCC-P Input for F_i:

$$< Pay_{P_0^i} \times D_{p_0^i}, \ldots, AdvPay_{P_k^i} \times D_{p_{k-1}^i}, \ldots,$$
$$Pay_{P_{n-2}^i} \times D_{p_{n-2}^i} >. \tag{15}$$

Algorithm 2 generates UAP that is used as an adversarial payload in this attack. In FCC-HP, the flow set F consists of the transport layer header and the payload of packets, and in FCC-P, it only consists of the payload of packets. Dummy packet index vector IND_{AdvPay} is considered to determine the location of the dummy packets being injected to the flow set F. The ith element of IND_{AdvPay} determines the index of the dummy packet being injected into the ith flow of the flow set F. For example, it can be the first or the second packet from source to destination or any other location in the first n packets of a flow. Algorithm 2 has T iterations, and in each iteration, $batch\_size$ flows from the flow set F is sampled F^{batch} and then the index of dummy packets of flows in F^{batch} is assigned to IND^{batch}. For flow i in F^{batch}, the place of dummy packet is determined by IND_i^{batch} and a dummy packet that carries UAP ξ with size $AdvPay_{Size}$ bytes is injected into that place in the byte sequence of flow $f\_byte\_seq_i^{batch}$. In each iteration, ξ is updated based on the

gradient of loss function with respect to ξ and ϵ tunes the size of change in ξ. The function $Clip_{Domain(F)}$ maps ξ to the domain of byte sequence of flows by clipping the value of ξ and controls the direction sign of dummy packet.

C. Adversarial Burst Attack

The flow time series contains the statistical features of packets in the order which they are received. A burst in a flow is a sequence of consecutive packets in one direction. The inputs FTSC-IAT and FTSC-PS for flow F_i can be expressed using the concept of bursts as follows:

FTSC-IAT and FTSC-PS Inputs for F_i:

$$< \left(Brs_0^i, D_0^i\right), \left(Brs_1^i, D_1^i\right), \ldots, \left(Brs_n^i, D_n^i\right) >$$
$$s.t.\ D_j^i \in \{-1, +1\},\ D_j^i = -1 \times D_{j+1}^i,\ j \in [0, n - 1],$$
$$\tag{16}$$

where n is the number of bursts in flow F_i and $\left(Brs_j^i, D_j^i\right)$ is defined as follows:

$$\left(Brs_j^i, D_j^i\right)\ \text{for FTSC-PS:}$$
$$< PS_{p_0^j} \times D_{p_0^j}, PS_{p_1^j} \times D_{p_1^j}, \ldots, PS_{p_m^j} \times D_{p_m^j} >,$$
$$\left(Brs_j^i, D_j^i\right)\ \text{for FTSC-IAT:} < IAT_{\left(p_{m'}^{(j-1)}, p_0^j\right)} \times D_{p_0^j},$$
$$IAT_{\left(p_0^j, p_1^j\right)} \times D_{p_1^j}, \ldots, IAT_{\left(p_{m-1}^j, p_m^j\right)} \times D_{p_m^j} >,$$
$$s.t.\ D_{p_k^j} \in \{-1, +1\},\ D_{p_k^j} = D_j^i,\ k \in [0, m], \tag{17}$$

where m is the number of packets in the Brs_j^i, $p_{m'}^{(j-1)}$ is the last packet of $Brs_{(j-1)}^i$ and if $j = 0$ then the $IAT_{\left(p_{m'}^{(j-1)}, p_0^j\right)}$ will be 0. In Adversarial Burst (AdvBurst) attack, multiple dummy packets with crafted statistical features are added to

Algorithm 3 Adversarial Burst

Input Flow set F of class l, flow content classifier f_{1D-CNN} with weights θ, selected burst indices vector IND_{SBurst}, number of dummy packets $Num\_of\_dummy\_pkts$, perturbation rate ϵ, number of iterations T, batch size $batch\_size$.

Output UAP ξ

1: $\xi \leftarrow Rand(Domain(F), Size = Num\_of\_dummy\_pkts)$
2: **for** $t \leftarrow 0, T$ **do**
3:    $F^{batch} \leftarrow$ sample $batch\_size$ flows from F
4:    $IND^{batch} \leftarrow$ selected burst index of F^{batch} in IND_{SBurst}
5:    **for** $i \leftarrow 0, batch\_size$ **do**
6:      $flows\_time\_series_i^{batch} \leftarrow flow\_to\_burst\_seq(F_i^{batch})$
7:      $Brs_s^i = flows\_time\_series_i^{batch}[IND_i^{batch}]$
8:      $Brs_{adv}^i = append(Brs_s^i, \xi)$
9:      $flows\_time\_series_i^{batch}[IND_i^{batch}] = Brs_{adv}^i$
10:    **end for**
11:    $\Delta\xi = \epsilon \times \nabla_\xi J(\theta, flows\_time\_series^{batch}, l)$
12:    $\xi = Clip_{Domain(F)}(\xi + \Delta\xi)$
13: **end for**
14: **return** ξ

---

the end of a selected burst of a flow. First, a burst Brs_s^i from the flow F_i is selected, and then, d dummy packets are appended to the end of it. The direction of dummy packets and the selected burst Brs_s^i is the same. The sequence of dummy packets $< p_1^{dummy}, p_2^{dummy}, \ldots, p_d^{dummy} >$ and the selected burst Brs_s^i build a new burst which is called adversarial burst Brs_{adv}^i. After applying the AdvBurst attack, the structure of flow F_i is as follows:

$$F_i: < \left(Brs_0^i, D_0^i\right), \left(Brs_1^i, D_1^i\right), \ldots, \left(Brs_{s-1}^i, D_{s-1}^i\right),$$
$$\left(Brs_{adv}^i, D_s^i\right), \ldots, \left(Brs_n^i, D_n^i\right)>, \quad (18)$$

where $\left(Brs_{adv}^i, D_s^i\right)$ for FTSC-IAT and FTSC-PS is defined as follows:

$\left(Brs_{adv}^i, D_s^i\right)$ for FTSC-PS:
$$< PS_{p_0^s} \times D_{p_0^s}, \ldots, PS_{p_m^s} \times D_{p_m^s}, PS_{p_0^{dummy}}$$
$$\times D_s^i, \ldots, PS_{p_d^{dummy}} \times D_s^i>,$$

$\left(Brs_{adv}^i, D_s^i\right)$ for FTSC-IAT:
$$< IAT_{\left(p_{m'}^{(s-1)}, p_0^s\right)} \times D_{p_0^s}, \ldots, IAT_{\left(p_{m-1}^s, p_m^s\right)} \times D_{p_m^s},$$
$$IAT_{\left(p_m^s, p_0^{dummy}\right)} \times D_s^i, \ldots, IAT_{\left(p_{d-1}^{dummy}, p_d^{dummy}\right)} \times D_s^i, >, \quad (19)$$

where Brs_s^i is the selected burst, and m is the number of packets in the selected burst. Algorithm 3 generates UAP that AdvBurst attack uses as the statistical features of dummy packets. To specify the place of adversarial burst in each flow, this algorithm gets the selected burst indices vector IND_{SBurst}, which determines the indices of the selected burst in each flow. Algorithm 3 runs in T iterations. In each iteration, $batch\_size$ number of flows are sampled F^{batch}, and the selected burst indices of these flows is assigned to IND^{batch}. For flow i in F^{batch}, first, the flow is transformed into burst sequences $flows\_time\_series_i^{batch}$, and then the selected burst Brs_s^i is replaced by the adversarial burst Brs_{adv}^i, which the index of selected burst Brs_s^i is determined by IND_i^{batch}. The number of dummy packets that are added to the end of the selected burst

is specified by $Num\_of\_dummy\_pkts$ parameter. The parameter ξ is updated according to the gradient of loss function with respect to ξ, and ϵ controls the rate of change in ξ. The function $Clip_{Domain(F)}$ makes sure that the perturbation ξ is in the domain of statistical features of flows, and the sign of ξ is the same as the direction of dummy packets.

V. EVALUATION

Three attacks have been proposed in order to evaluate the robustness of the DL-based network traffic classifiers with respect to the three categories of input space. In this section, the robustness of six different classifiers is examined against ANT. Moreover, the robustness of classifiers that get the transport layer header as the input, PC-HP and FCC-HP, are evaluated against port attack. In port attack, source and destination port numbers of flows are randomly assigned from a specific range of port numbers. Typical performance metrics for evaluating a classifier are precision, recall, and F-score. The recall shows what fraction of given class data is classified correctly. Since proposed attacks cause the classifiers to misclassify data of a given class, we focus on recall in the evaluation section. In the following experiments, we assume an adversary has white-box access to DL-based classifiers. In the white-box setting, an adversary (e.g., malicious cloud provider) knows the input space category, the architecture, the weights, and the outputs of the classifier. Papernot *et al.* [28] have shown that adversarial examples are transferable among different classifiers with different training data set. In the black-box setting, an adversary can use proposed attacks and run them on a substitute classifier, which is close to the target classifier, and he/she uses the generated perturbations to evade a target classifier. We evaluate the transferability of ANT in Section V-E.

A. Dataset

DL-based network traffic classifiers need to be trained using a labeled set of network traffic in the training phase. To the best of our knowledge, a few studies have a reliable labeling process, and some of them have not enough data to train a deep neural network. Draper-Gil *et al.* [11] have presented ISCXVPN2016 dataset that consists of six types of network traffic, and some of flows in the dataset have been sent through a VPN. They used different applications, such as Skype, Facebook, Hangouts, Youtube, and Bittorent, to generate traffic with different types, including chat, email, file transfer, streaming, torrent, and VoIP. This dataset has been labeled based on application and type, which have generated given network traffic. This dataset is highly imbalanced and it decreases the performance of DL-based classifiers. We use undersampling for classes having more data and oversampling for classes having fewer data to overcome this challenge [29]. The undersampling method eliminates a certain amount of examples from the training dataset that belong to the majority class. The oversampling method replicates examples from the training dataset that belong to the minority class until the amount of data for that class reaches a specific number. The dataset is split into training (60%), validation (20%), and

TABLE II
PACKET CLASSIFICATION DATASET

Type	Imbalanced Dataset		Balanced Dataset	
	Total Number	Training Set	Total Number	Training Set
Chat	68,864	41,646	68,864	41,646
Email	30,807	18,558	51,849	39,600
FileTransfer	313,916	188,349	100,796	60,505
Streaming	124,487	74,692	100,158	60,023
Torrent	162,479	97,487	101,164	60,625
VoIP	596,476	357,885	100,432	60,018

TABLE III
PERFORMANCE METRICS OF THE PACKET CLASSIFIERS
PC-HP, AND PC-P

Type	Precision(%)		Recall(%)		F-score(%)	
	PC-HP	PC-P	PC-HP	PC-P	PC-HP	PC-P
Chat	85.35	74.11	76.45	47.31	80.66	57.75
Email	65.77	59.97	75.35	69.84	70.23	64.53
FileTransfer	98.41	67.51	90.23	77.19	94.14	72.03
Streaming	79.96	48.08	91.50	41.88	85.34	44.77
Torrent	99.3	69.66	99.97	77.33	99.63	73.29
VoIP	87.24	76.33	84.47	85.15	85.83	80.50

test sets (20%). The dataset requires a preprocessing phase to clean some background network traffic, such as DNS and NETBIOS. Since there are three categories of input spaces, data padding and normalization of each category is different. Hence, we discuss dataset properties of each category in a separate subsection.

B. Packet Classification

In this section, the robustness of packet classifiers against port and adversarial pad (AdvPad) attacks is evaluated. We propose a random pad attack (RandPad) as a baseline to assess the effectiveness of the AdvPad attack. RandPad injects a random pad into the payload of packets. AdvPad must have a better performance than RandPad . In the AdvPad attack, for a given class, Algorithm 1 generates a UAP on the packets of that class in the validation set with the desired percentage of bandwidth overhead, and then the UAP is injected into packets of that class in the test set, and the performance of packet classifiers is evaluated against these packets.

1) Packet Classification Dataset: Packet classification dataset consists of the byte sequence of packets that have at least one byte of payload. Since the decimal domain of each byte of a packet is in the [0,255] interval, we divide the decimal value of each byte by 255 to normalize the byte sequence of packets. Hence, each packet becomes a sequence of numbers between zero and one. Also, we need to add zero pad to the end of packets until their length reaches to *MaxPktSize*. Table II shows the number of samples in each class.

2) Packet Classifiers Setup: A 1D-CNN is used to classify the byte sequences of packets. Given our computational resources, we trained 50 1D-CNNs with different architectures and hyper-parameters, and finally, the classifier with higher accuracy on the validation set has been selected. The overall accuracy of PC-HP and PC-P is 88.09% and 66.71%, respectively. The precision, recall, and F-score of these classifiers are reported in Table III. PC-HP has higher performance than PC-P because of the information in the transport layer header of packets.

3) Packet Classifiers Robustness Evaluation: AdvPad, RandPad, and port attack are applied to packet classifiers to evaluate their robustness. AdvPad has been conducted for 1000 iterations with *batch_size* = 128 and ϵ = 0.01. RandPad has been run 50 times, and the average results are reported in this section. Figure 1 shows the recall of PC-HP under proposed attacks for the various magnitude of BandWidth (BW) overhead for all classes.

Port attack randomly changes the transport layer port numbers of flows, and it does not impose any BW overhead on network traffic. The recall of chat, email, and streaming classes decreases considerably under port attack that shows PC-HP is highly sensitive to the transport layer port numbers for classifying these classes. However, the port attack has little effect on the recall of other classes.

End_AdvPad considerably decreases the recall of PC-HP in most classes with little bandwidth overhead, and recall is more decreased by increasing the size of the adversarial pad. If it combines with port attack (End_AdvPad+Port), the performance of attack is raised in most classes. End_RandPad injects a random pad at the end of packets payload, and End_RandPad+Port combines End_RandPad with port attack. End_AdvPad and End_AdvPad+Port have better performance than End_RandPad and End_RandPad+Port attacks, respectively. This observation shows the effectiveness of the AdvPad attack.

Start_AdvPad reduces the recall of all classes to under 25% with just 10% bandwidth overhead. The recall of PC-HP under Start_AdvPad gets close to 0% by increasing the size of adversarial pad. Adding Port attack to Start_AdvPad (Start_AdvPad+Port) does not improve the performance of Start_AdvPad in most classes. In the Start_RandPad attack, a random pad is injected into the start of packets payload, and Start_RandPad+port mixes Start_RandPad with port attack. There is a significant gap between the recall of PC-HP under Start_AdvPad and Start_RandPad, and also, there is a large gap between the recall of PC-HP under Start_AdvPad+port and Start_RandPad+port. These gaps show the remarkable effectiveness of the adversarial pad at the start of packets for all classes. The recall of PC-HP under Start_AdvPad and Start_AdvPad+Port is reduced more than the End_AdvPad and the End_AdvPad+Port for all classes, respectively. These observations show PC-HP is more sensitive to the information at the start of packets payload than the information at the end of it. The recall of PC-HP under the End_RandPad and Start_RandPad are very analogous to each other in some classes. The performance of random pad attacks is lower than the End_AdvPad and the End_AdvPad+Port attacks in most classes. These results show the effectiveness of adversarial pad attacks.

Figure 1(b) indicates the recall of PC-P under proposed attacks for different magnitudes of BW overhead for all classes. The performance of the End_AdvPad is variable in various classes. Although the decline of recall of PC-P under
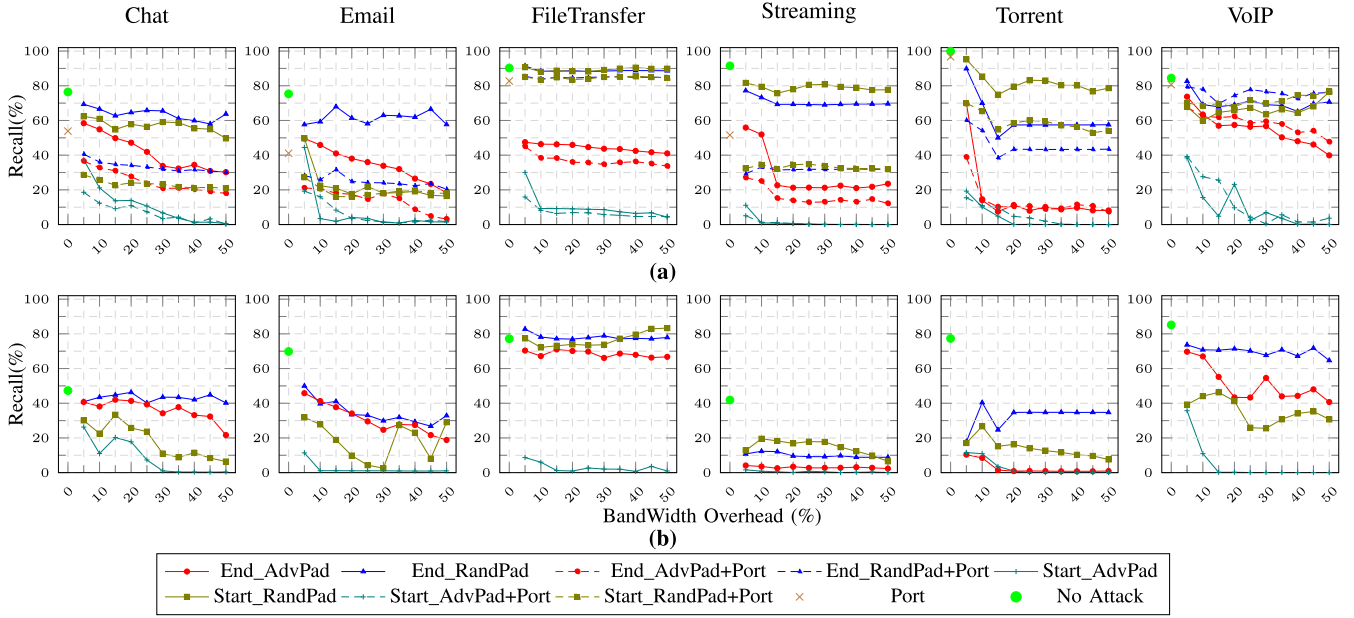
Fig. 1. The recall of PC-HP (a) and PC-P (b) under different attacks over various sizes of BandWidth Overhead (BWO) for all classes. The legends show various kinds of attacks that have been applied.

the End_AdvPad is low in file transfer and chat classes, the End_AdvPad highly decreases the recall of PC-P in other classes. The recall of PC-P under the End_RandPad is a little higher than the recall of PC-P under the End_AdvPad in most classes, which indicates low effectiveness of the End_AdvPad on PC-P. Start_AdvPad significantly decreases the recall of PC-P and indicates the best performance among all attacks on PC-P. It decreases the recall of all classes to less than 3% with just 30% BW overhead. Start_AdvPad has better performance than Start_RandPad in all classes. In some classes such as VoIP, email, chat, and file transfer, the recall of PC-P under the Start_RandPad is close or lower than recall of PC-P under the End_AdvPad. This observation demonstrates PC-P is very sensitive to the information at the start of packets.

C. Flow Content Classification

We evaluate the robustness of flow content classifiers against adversarial payload (AdvPay) attack. Also, we propose Random Payload (RandPay) attack as a baseline attack to show the effectiveness of the AdvPay attack. In the RandPay attack, a dummy packet that carries a payload with a random byte sequence is injected into the first n packets of a flow. In the AdvPay attack, for a given class, Algorithm 2 generates a UAP with the desired size on flows of that class in the validation set, then for each flow of that class in the test set, this perturbation is injected into the payload of a dummy packet, and the robustness of flow content classifiers is evaluated against them.

1) Flow Content Classification Dataset: Similar to packet classification, the decimal value of each byte is divided by 255 to normalize the byte sequence of packets between zero and one, and then, zero pad is added to the end of packets until their length reaches to *MaxPktSize*. Afterwards, the byte sequences of the first n packets of a flow which have at least one byte of payload are concatenated together, and this new sequence is called flow byte sequence. Table IV indicates the number of samples in each class before and after oversampling.

2) Flow Content Classifiers Setup: A 1D-CNN is used to classify the byte sequence of flows. Given our Computational resource, 150 1D-CNN with various architectures and hyperparameters have been explored to find the best classifier for FCC. Eventually, the classifier with higher accuracy on the validation set has been chosen. Based on experiments, we chose 10 packets to be in each flow byte sequence. The overall accuracy of FCC-HP and FCC-P is 81.52% and 80.6%, respectively. Other performance metrics of these classifiers are presented in Table V. The results demonstrate that the influence of the header of transport layer on FCC is low, and even in some classes, the performance of FCC-P is better than FCC-HP.

3) Flow Content Classifiers Robustness Evaluation: AdvPay, RandPay, and port attack have been applied to flow content classifiers to evaluate their robustness. AdvPay attack has been run for 1000 iterations with *batch_size* = 64 and $\epsilon = 0.001$. The dummy packet is injected after the first packet from source (e.g., client) to destination (e.g., server). RandPay

TABLE IV
FLOW CONTENT CLASSIFICATION DATASET

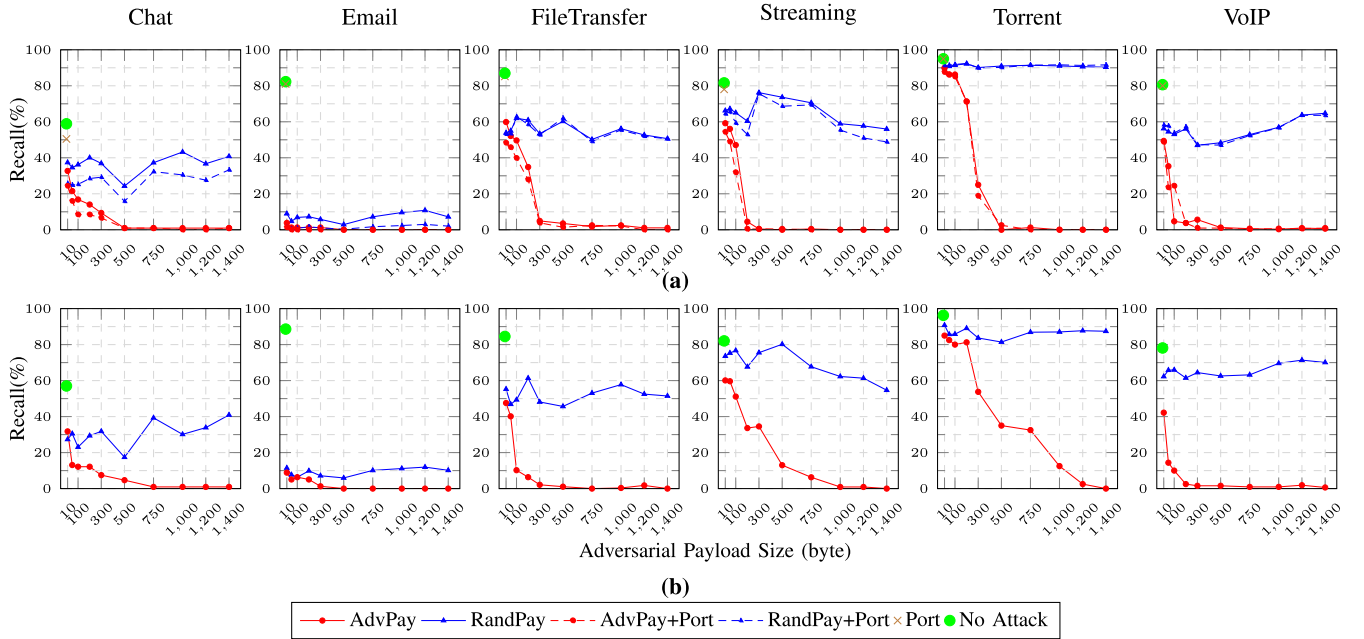| Type	Imbalanced Dataset		Balanced Dataset	
	Total Number	Training Set	Total Number	Training Set
Chat	536	321	1174	959
Email	392	235	1116	959
FileTransfer	1420	852	1527	959
Streaming	1114	668	1405	959
Torrent	400	240	1119	959
VoIP	1598	959	1598	959

Fig. 2. The recall of FCC-HP (a) and FCC-P (b) under different attacks over various sizes of adversarial payload for all classes. The legends show various kinds of attacks that have been applied.

TABLE V
PERFORMANCE METRICS OF THE FLOW CONTENT CLASSIFIERS
FCC-HP, AND FCC-P

Type	Precision(%)		Recall(%)		F-score(%)	
	FCC-HP	FCC-P	FCC-HP	FCC-P	FCC-HP	FCC-P
Chat	73.26	74.39	58.88	57.01	65.29	64.55
Email	89.04	85.37	82.28	88.61	85.53	86.96
FileTransfer	85.17	90.23	86.97	84.51	86.06	87.28
Streaming	79.82	74.69	81.61	82.06	80.71	78.25
Torrent	98.7	97.47	95.00	96.25	96.81	96.86
VoIP	76.11	73.75	80.62	78.12	78.30	75.87

attack has been conducted 50 times, and the average results are reported in this section.

Figure 2(a) shows the recall of FCC-HP under AdvPay, RandPay, and port attack over various sizes of adversarial payload for all classes. The recall of FCC-HP under port attack is slightly reduced, which shows FCC-HP does not rely too much on the information of the port numbers of transport layer header. The recall of all classes under AdvPay attack drops to less than 3.52% with just 500 bytes of adversarial payload. AdvPay+Port combines AdvPay with port attack. It increases the performance of the AdvPay attack a little bit and decreases the recall of all classes under AdvPay+Port to less than 1.41% with just 500 bytes of adversarial payload. The RandPay attack injects a dummy packet that carries a random byte sequence as the payload into a flow. Similar to AdvPay, the dummy packet is sent after the first packet from source to destination. Although the performance of RandPay is good, AdvPay attack has much better performance. RandPay+Port combines RandPay with port attack. The performance of RandPay+Port is very close to RandPay. Unlike AdvPay, by increasing the size of adversarial payload, the recall of FCC-HP under RandPay and RandPay+Port remains relatively

fixed. However, the recall of FCC-HP under AdvPay and AdvPay+Port is decreased by increasing the size of adversarial payload.

Figure 2(b) indicates the recall of FCC-P under AdvPay and RandPay over various sizes of adversarial payload for all classes. Except for the torrent class, the recall for all classes under AdvPay drops off to almost 0% with 1000 or fewer bytes of adversarial payload. The recall of the torrent class decreases to 0% with 1400 bytes adversarial payload. RandPad has poor performance on FCC-P in most classes, and by increasing the size of adversarial payload, the performance of RandPay does not improve very much. This observation demonstrates the effectiveness of the AdvPay attack on FCC-P. Although FCC-P has almost 1% less overall accuracy than FCC-HP, results in Figure 2 indicate FCC-P is slightly more robust than FCC-HP, and for reducing the performance of FCC-P, the size of adversarial payload needs to be increased more.

D. Flow Time Series Classification

We evaluate the robustness of flow time series classifiers against adversarial burst (AdvBurst) attack. Also, we conduct RandBurst on these classifiers to evaluate the effectiveness of the AdvBurst attack. In the RandBurst attack, the statistical features of dummy packets are randomly perturbed. AdvBurst must have better performance than RandBurst. AdvBurst using Algorithm 3 generates a UAP with a specific size on the flows of a given class in the validation set. Then, this perturbation is used as the statistical feature of dummy packets, which are appended to the end of selected bursts of flows of that class in the test set, and the robustness of classifiers is evaluated against these flows.

1) Flow Time Series Classification Dataset: The number of samples in the dataset of FTSC is very close to the FCC
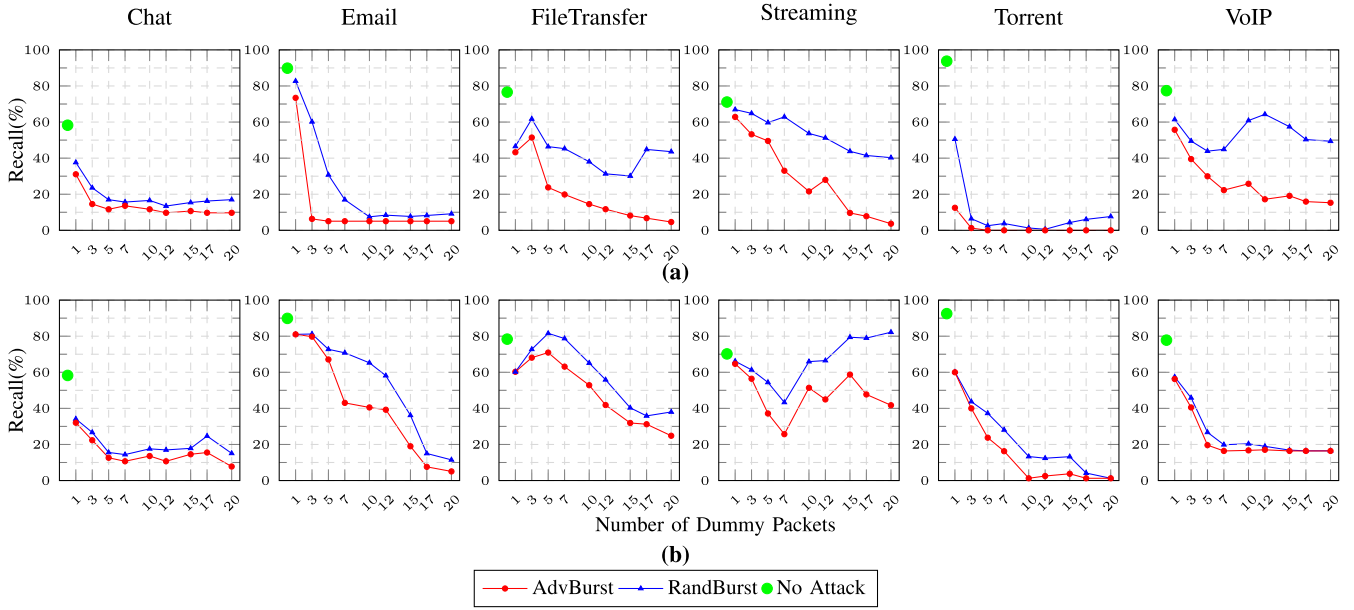
Fig. 3. The recall of FTSC-PS (a) and FTSC-IAT (b) under different attacks over various number of dummy packets for all classes. The legends show various kinds of attacks that have been applied.

TABLE VI
PERFORMANCE METRICS OF THE FLOW TIME SERIES CLASSIFIERS FTSC-PS, AND FTSC-IAT

Type	Precision(%)		Recall(%)		F-score(%)	
	FTSC-PS	FTSC-IAT	FTSC-PS	FTSC-IAT	FTSC-PS	FTSC-IAT
Chat	58.25	65.22	58.25	58.25	58.25	61.54
Email	92.21	85.54	89.87	89.87	91.02	87.65
FileTransfer	80.00	80.95	76.60	78.37	78.26	79.64
Streaming	78.28	75.00	71.10	70.18	74.52	72.51
Torrent	88.24	90.24	93.75	92.50	90.91	91.36
VoIP	70.85	71.39	77.39	77.81	73.98	74.46

dataset (Table IV). The packets size data are normalized as follows:

$$Normalized\ PacketSize = \frac{PacketSize - \mu_{PktSize}}{Std_{PktSize}} \quad (20)$$

where $\mu_{PktSize}$, and $Std_{PktSize}$ indicate mean and standard deviation of packets size data, respectively. The IAT data are as follows:

$$Normalized\ IAT = 2 \times \log_{IAT_{max}}^{(IAT+1\mu Sec)} \quad (21)$$

where IAT_{max} is the maximum of IAT data.

2) Flow Time Series Classifiers Setup: A 1D-CCN is used to classify flow time series. Given our computational resource, we have explored 150 different architectures and hyperparameters to build the final flow time series classifier. Based on experiments, we chose 100 packets to be in the time series of each flow. The overall accuracy of FTSC-PS and FTSC-IAT is 76.21% and 76.51%, respectively. The other performance metrics of these classifiers are reported on Table VI. Based on our experiments, FTSC-IAT has a little bit better performance than FTSC-PS.

3) Flow Time Series Classifiers Robustness Evaluation: AdvBurst and RandBurst have been applied to flow time series classifiers to evaluate their robustness. AdvBurst attack has

been conducted for 2000 iterations with $batch\_siz = 64$ and $\epsilon = 0.01$. RandBurst has been conducted 50 times, and the average results are reported in this section. The domain of IAT is between 0 and 180 seconds with microsecond granularity. If the IAT domain of dummy packets be equal to the domain of IAT in the dataset, AdvBurst can impose high time overhead on a flow. Therefore, we limit the IAT domain of dummy packets between 0.001 and 0.1 seconds, and this restriction is applied using the clip function in Algorithm 3.

We choose the first burst from source to destination as the selected burst in all flows to attack FTSC-PS. Figure 3(a) shows the recall of FTSC-PS under AdvBurst and RandBurst over different numbers of dummy packets for all classes. The RandBurst attack appends dummy packets with random statistical features at the end of the selected bursts. AdvBurst highly decreases the recall of FTSC-PS in all classes with just a few dummy packets. In chat, email, and torrent classes, the recall of FTSC-PS under AdvBurst drops to less than 11.65% using just five dummy packets. Also, RandBurst shows excellent performance in these classes. Although AdvBurst has better performance on these classes, results show those classes are highly vulnerable to little manipulation in the size of dummy packets, and even dummy packets with random packet sizes can highly decrease the recall for these classes. In contrast, although the performance of AdvBurst in file transfer, streaming, and VoIP classes, is not as fine as those classes, the distance between the recall of FTSC-PS under AdvBurst and RandBurst is large. This observation indicates the effectiveness of AdvBurst in these classes.

We consider the first burst from destination to source in a flow as the selected burst to attack on FTSC-IAT. Figure 3(b) shows the recall of FTSC-IAT under AdvBurst and RandBurst for all classes. AdvBurst highly reduces the recall of FTSC-IAT for all classes with just a few dummy packets. However, its performance is various in different classes. The recall

TABLE VII
THE RECALL AND THE OVERALL ACCURACY OF SIX NEW TARGET CLASSIFIERS AGAINST ANT. THE TARGET CLASSIFIERS ARE STACKED
AUTO-ENCODERS (SAEs), AND ANT IS CREATED USING THE 1D-CNNs CLASSIFIERS IN PREVIOUS EXPERIMENTS

| | Attack | Attack Parameter | Overall Accuracy(%) | Recall(%) | | | | | |
				Chat	Email	File Transfer	Streaming	Torrent	VoIP
PC-HP	No Attack		82.01	67.83	79.04	90.52	80.03	96.00	74.04
	End_AdvPad	20% BWO	47.95	42.75	75.91	71.44	63.11	0.65	51.87
	End_AdvPad + Port	20% BWO	41.52	19.67	57.90	69.63	46.13	0.27	62.84
	Start_AdvPad	20% BWO	42.82	30.66	13.05	79.27	77.19	0.61	34.81
	Start_AdvPad + Port	20% BWO	35.46	22.81	9.22	68.67	41.95	0.16	50.69
PC-P	No Attack		63.22	35.37	80.23	76.11	21.88	87.67	84.08
	End_AdvPad	20% BWO	36.15	31.64	68.70	76.62	9.29	0.36	51.33
	Start_AdvPad	20% BWO	8.42	21.09	2.91	12.63	10.20	0.23	2.19
FCC-HP	No Attack		77.40	50.47	86.08	80.63	78.03	95.00	76.56
	AdvPay	750 bytes	42.91	31.78	3.80	42.61	44.39	81.25	45.94
	AdvPay + Port	750 bytes	41.35	23.36	1.27	34.51	43.95	82.50	51.25
FCC-P	No Attack		76.85	47.66	84.81	80.99	78.03	95.00	75.62
	AdvPay	750 bytes	43.18	23.36	6.33	36.27	52.47	81.25	49.06
FTSC-IAT	No Attack		69.62	34.95	75.95	73.76	66.51	85.00	73.95
	AdvBurst	7 dummy pkts	31.31	5.83	37.97	19.50	81.65	3.75	20.58
FTSC-PS	No Attack		67.94	48.54	79.75	67.38	62.84	88.75	70.06
	AdvBurst	7 dummy pkts	30.76	7.77	6.33	78.01	22.94	0.00	15.29

of FTSC-IAT under AdvBurst becomes less than 16.4% for chat, torrent, and VoIP classes with just seven dummy packets. AdvBurst needs more dummy packets to highly decrease the recall of FTSC-IAT for email and file transfer classes. Although the performance of AdvBurst on FTSC-IAT is fine enough, the gap between the recall of FTSC-IAT under AdvBurst and RandBurst is relatively small in most classes. We think the effectiveness of AdvBurst has decreased because of the domain of IAT has been limited between 0.001 and 0.1 seconds, and this domain is remarkably shorter than the real domain of IAT in the dataset. The recall of streaming class under AdvBurst has a strange behavior by increasing the number of dummy packets. The recall decreases continuously until the seventh dummy packet is added, and it begins to rise after that. We think the reason for this behavior is that the size of the burst heading from destination to source is often large in streaming class, and when the size of adversarial burst gets increased by increasing the number of dummy packets, FTSC-IAT recognizes these flows as streaming.

E. Transferability of ANT

Adversarial examples are transferable among various classifiers and training sets. Since the limitation of the dataset, we only evaluate the transferability of ANT among various classifiers. We train six new Stacked Autoencoder (SAE) classifiers as the new target classifiers and use ANT that has been generated in previous experiments to evaluate the transferability of ANT. We utilize the SAE architecture of [21] for the new packet and flow content classifiers and [9] for the new flow time series classifiers. In this experiment, there is no knowledge about the parameters of the target classifiers in the generation process of ANT. Table VII indicates the performance of the new target classifiers against ANT. The results demonstrate that ANT is transferable among various DNNs architectures in all input space categories. The

results also demonstrate that when there is no attack, the performance of 1D-CNNs is better than SAEs in network traffic classification.

VI. RESULTS ANALYSIS

The results demonstrate that ANT is effective against all classifiers in various input space categories. However, its effectiveness is not the same in different experiments. The results indicate that the port numbers in the transport layer headers are very sensitive for PC-HP classifier, and also packet classifiers are more vulnerable to Start_AdvPad attack than End_AdvPad attack. These observations reveal that when the transport layer header is included in the input, the classifier selects port number as a good feature for network traffic classification. Also, the start of payloads is more sensitive to classifiers than the end of them. We think this is because the start of the payloads has a specific structure in most protocols, but the end of them is often filled by data, which can be different between the same class packets. Since AdvPad attack imposes bandwidth overhead to each packet, reducing the performance of packet classifiers requires more bandwidth overhead among other classifiers. However, the performance of PC-P is low, and PC-HP is vulnerable to port attack, which is a very simple attack and does not impose any bandwidth overhead to network traffic.

AdvPay attack imposes the least bandwidth overhead into network traffic among all attacks, and it significantly decreases the recall of all classes. Hence, flow content classifiers have the lowest robustness among all classifiers. However, flow content classifiers are almost robust against the port attack. Since AdvBurst attack only injects some dummy packets with predetermined packet size or inter-arrival time into network traffic, the content of dummy packets can be anything. Therefore AdvBurst attack is the most challenging attack to detect or mitigate.

VII. ANT IN PRACTICE

In the image classification domain, humans determine the true class of image, and adversarial perturbation must be imperceptible to humans and does not change the true class of a sample. However, in network traffic classification, the true class of network traffic is determined by the application that generates and receives that network traffic. If the functionality of the application does not disrupt, the true class of network traffic is preserved. Proposed attacks change the content of network traffic to evade DL-based network traffic classifiers. However, since they do not change or remove the content of original network traffic, it can be retrieved. There are two approaches to retrieve original network traffic from perturbed network traffic. First, applications that generate and receive network traffic must be aware of the perturbation being added to network traffic and must remove this perturbation to retrieve original network traffic. Second, there must be a proxy to add or remove perturbation to network traffic. This proxy can be in the adversary's device or middleware in the adversary' network. The second approach is more convenient and is independent of applications that generate and receive network traffic. Although some proposed attacks may not be applicable to any network situations, they have the potential to adapt to the various network situations by a little change. For example, when a firewall in the adversary's network only passes the encrypted flows that start with SSL/TLS handshake packets, the adversarial pad can only be added to the end of the handshake packets, or dummy packet can be added after the handshake packets.

Previous studies [5]–[7], [10], [11], [21], [24], [26] have demonstrated that approaches being used to evade payload-based and port-based classifiers are not effective against the DL-based network traffic classifier. For example, although ISCXVPN2016 dataset consists of encrypted and tunneled network traffic, it is classified by the ML-based classifiers with high accuracy in this study an previous studies [5], [6], [10], [11], [21]. Similarly, in website fingerprinting attack, although the network traffic is generated by privacy-enhancing technologies such as Tor, it can be classified with high accuracy using time-series features of flows [8], [9]. Therefore proposed attacks have only focused on the evading of the DL-based network traffic classifiers, and if an adversary also wants to evade port-based or payload-based network traffic classifiers, (s)he must use appropriate approaches such as dynamic port assignment, payload encryption, or tunneling mechanism along with our proposed approaches to evade these classifiers.

VIII. CONCLUSION

In this article, the robustness of DL-based network traffic classifiers against Adversarial Network Traffic (ANT) has been evaluated. Because of using universal adversarial perturbation generating methods in ANT, there is no need to have access to target network traffic in advance, and it is generated live. Based on the literature, we considered three input space categories in DL-based network traffic classification, including packet classification, flow content classification, and flow time series classification. Based on these categories, we proposed three new attacks to make ANT. The results indicate the robustness of network traffic classifiers in all input space categories are very low in facing ANT, and also, their performance considerably decreases by little random perturbation. The Robustness of DL-based network traffic classifiers is a critical issue, and by continuing researches in this area, we aim to increase the robustness of these classifiers.

REFERENCES

[1] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1988–2014, 2nd Quart., 2019.

[2] M. Dusi, F. Gringoli, and L. Salgarelli, "Quantifying the accuracy of the ground truth associated with Internet traffic traces," *Comput. Netw.*, vol. 55, no. 5, pp. 1158–1167, 2011.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*. Red Hook, NY, USA: Curran, 2012, pp. 1106–1114.

[4] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[5] M. L. Martín, B. Carro, A. Sánchez-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.

[6] J. A. Caicedo-Muñoz, A. L. Espino, J. C. Corrales, and Á. R. Gallón, "QoS-classifier for VPN and non-VPN traffic based on time-related features," *Comput. Netw.*, vol. 144, pp. 271–279, Oct. 2018.

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

[8] P. Sirinam, M. Imani, M. Juárez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, Toronto, ON, Canada, Oct. 2018, pp. 1928–1943.

[9] V. Rimmer, D. Preuveneers, M. Juárez, T. van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proc. 25th Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, Feb. 2018.

[10] 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, 2020.

[11] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," in *Proc. 2nd Int. Conf. Inf. Syst. Security Privacy (ICISSP)*, Rome, Italy, Feb. 2016, pp. 407–414.

[12] C. Szegedy *et al.*, "Intriguing properties of neural networks," in *Proc. 2nd Int. Conf. Learn. Represent. (ICLR)*, Banff, AB, Canada, Apr. 2014.

[13] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Symp. Security Privacy (SP)*, San Jose, CA, USA, May 2017, pp. 39–57.

[14] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, May 2015, pp. 1–11.

[15] T. B. Brown, D. Mané, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," 2017. [Online]. Available: https://arxiv.org/abs/1712.09665.

[16] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 86–94.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[18] G. Verma, E. N. Ciftcioglu, R. Sheatsley, K. S. Chan, and L. Scott, "Network traffic obfuscation: An adversarial machine learning approach," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, Los Angeles, CA, USA, 2018, pp. 1–6.

[19] M. Usama, A. Qayyum, J. Qadir, and A. I. Al-Fuqaha, "Black-box adversarial machine learning attack on network traffic classification," in *Proc. 15th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Tangier, Morocco, 2019, pp. 84–89.

[20] M. S. Rahman, M. Imani, N. Mathews, and M. Wright, "Mockingbird: Defending against deep-learning-based website fingerprinting attacks with adversarial traces," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 1594–1609, 2020.

[21] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in SDN home gateway," *IEEE Access*, vol. 6, pp. 55380–55391, 2018.

[22] Z. Wang, *The Applications of Deep Learning on Traffic Identification*, vol. 24, BlackHat, San Francisco, CA, USA, 2015.

[23] 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. Security Informat. (ISI)*, Beijing, China, 2017, pp. 43–48.

[24] S. Rezaei, B. Kroencke, and X. Liu, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2020.

[25] K. Abe and S. Goto, "Fingerprinting attack on tor anonymity using deep learning," *Proc. Asia–Pac. Adv. Netw.*, vol. 42, pp. 15–20, Aug. 2016.

[26] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, pp. 63–78, 2018.

[27] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *Proc. IEEE Eur. Symp. Security Privacy (EuroS&P)*, Saarbrücken, Germany, 2016, pp. 439–454.

[28] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: From phenomena to black-box attacks using adversarial samples," 2016. [Online]. Available: https://arxiv.org/abs/1605.07277

[29] P. Branco, L. Torgo, and R. P. Ribeiro, "A survey of predictive modeling on imbalanced domains," *ACM Comput. Surveys*, vol. 49, no. 2, pp. 1–50, 2016.

Amir Mahdi Sadeghzadeh received the B.Sc. degree in information technology engineering from the Isfahan University of Technology in 2014, and the M.Sc. degree in information technology engineering from the Sharif University of Technology in 2016, where he is currently pursuing the Ph.D. degree with the Department of Computer Engineering. His research interests include deep learning security, adversarial deep learning, and network traffic classification.

Saeed Shiravi received the B.Sc. degree in computer engineering from the University of Isfahan in 2015, and the M.Sc. degree in information technology engineering from the Sharif University of Technology in 2018. His research interests include anonymity network, network traffic classification, and adversarial machine learning.

Rasool Jalili received the B.Sc. degree in computer science from the Ferdowsi University of Mashhad in 1985, the M.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 1989, and the Ph.D. degree in computer science from the University of Sydney, Australia, in 1995. He then joined the Department of Computer Engineering, Sharif University of Technology in 1995, where he is currently an Associate Professor and the Director of Data and Network Security Lab. He has published more than 150 papers in computer security and pervasive computing in international journals and conferences proceedings. His research interests include access control, vulnerability analysis, database security, and machine learning security.