IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 7, 2020, accepted June 25, 2020, date of publication July 6, 2020, date of current version July 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3007577

GAN Tunnel: Network Traffic Steganography by Using GANs to Counter Internet Traffic Classifiers

SINA FATHI-KAZEROONI AND ROBERTO ROJAS-CESSA, (Senior Member, IEEE)
Networking Research Laboratory, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA
Corresponding author: Sina Fathi-Kazerooni (sina.fathi.kazerooni@njit.edu)

ABSTRACT In this paper, we introduce a novel traffic masking method, called Generative Adversarial Network (GAN) tunnel, to protect the identity of applications that generate network traffic from classification by adversarial Internet traffic classifiers (ITCs). Such ITCs have been used in the past for website fingerprinting and detection of network protocols. Their use is becoming more ubiquitous than before for inferring user information. ITCs based on machine learning can identify user applications by analyzing the statistical features of encrypted packets. Our proposed GAN tunnel generates traffic that mimics a decoy application and encapsulates actual user traffic in the GAN-generated traffic to prevent classification from adversarial ITCs. We show that the statistical distributions of the generated traffic features closely resemble those of the actual network traffic. Therefore, the actual user applications and information associated with the user remain anonymous. We test the GAN tunnel traffic against high-performing ITCs, such as Random Forest and eXtreme Gradient Boosting (XGBoost), and we show that the GAN tunnel protects the identity of the source applications effectively.

INDEX TERMS Generative adversarial networks, GAN, Internet packet classification, machine learning, online privacy, statistical traffic properties, WGAN.

I. INTRODUCTION

Internet traffic classifiers (ITCs) use the extracted statistical data from packets of a network to classify them and infer information from them [1], [2]. Internet service providers (ISPs) use ITCs to manage and monitor network traffic and provide quality of service (QoS) guarantees to network flows. Here, a flow is the set of packets generated by an application of a computing system that is identifiable by source/destination IP addresses and port numbers [1]. QoS-provisioning mechanisms prioritize some flows over others based on preset criteria. For instance, ISPs may prioritize multimedia traffic over mail traffic by assigning more bandwidth to the former traffic. For operation and management, ISPs also use ITCs to identify visited websites, services, or applications that generate the network traffic [2]. ITCs may also be used to detect network intrusions and distributed denial of service (DDoS) attacks [3], [4]. On the other hand, ITCs can also be used for adversarial purposes, such as determining the behavioral patterns of an online user. Therefore, ITCs may be a potential threat to privacy [5]. It is then not surprising that adversarial ITCs have been used for performing Internet traffic censorship [6].

The widening adoption of encrypted communications and the use of unknown port numbers by various applications make it difficult to perform direct packet classification based on port numbers or packet payloads [7], [8]. Instead, ITCs use the traffic statistics, including packet length, packet timings, and transport layer information, such as Transmission Control Protocol (TCP) window sizes, for classification [1]–[4]. Such ITCs primarily use machine learning (ML) algorithms, such as random forest (RF), neural networks, and support vector machines (SVM) [9]–[15] to perform classification. They have been used to identify traffic classes such as web browsing, email, P2P, video and music streaming, gaming, and file transfer, among others [9]–[15]. Moreover, ML-based ITCs may be used to profile online activity of a user by identifying popular applications, such as Google Chrome, WhatsApp, etc. [2].

A method used to undermine ITC classification is traffic obfuscation. This method bases its operation on performing packet padding and delaying transmission of packets to modify the traffic's profile [16]–[19]. However, these methods are easy to circumvent because traffic modification focuses on

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

a few and crude features. An adversary who uses a variety of ML algorithms can detect those changes as different ML algorithms are sensitive to different traffic features [2]. Therefore, there is a need for a method that can circumvent packet classification by obfuscating a holistic traffic profile.

To address this need, we propose using a generative adversarial network (GAN) model [20] to design a method that holistically modifies statistical traffic features to circumvent classification. Considering that GANs were originally used to generate fake images that are indistinguishable from real ones [20], we apply them but for traffic generation. In this paper, we design a GAN that generates traffic flows that mimic the traffic generated by an actual application. This actual application, used as a decoy, can be a selectable one. We then use the generated flows to encapsulate and transmit the actual traffic with protected privacy against adversarial ITCs, as a tunnel. We call this approach GAN tunnel.

A GAN tunnel encapsulates actual network traffic and represents it as one generated by a different application, the decoy application. The traffic generated mimics the statistical characteristics of the decoy application. To test the efficiency of the proposed GAN tunnel, we use high performing ITCs, such as RF [21] and eXtreme Gradient Boosting (XGBoost) [22]. These ITCs aim to detect the source application of the transmitted traffic with high accuracy. We show that the ITCs classify the GAN tunnel traffic as belonging to the decoy applications.

The main contributions of this paper are: We 1) design a network traffic generator based on GAN, 2) propose a novel GAN tunnel method to protect user traffic from classification, 3) demonstrate that the GAN tunnel traffic is effective in anonymizing the traffic against different ITCs. To best of our knowledge, this is the first work that proposes the use of a GAN for tunneling network traffic.

The remainder of this paper is organized as follows. In Section II, we describe our system design. In Section III, we present our evaluations results and discussions. In Section IV, we review related works. We conclude the paper in Section V.

II. MODEL DESCRIPTION

In our system, we create a tunnel between an online user and an application server, as Figure 1 shows. The GAN client and server embed actual network traffic into the generated traffic for the GAN tunnel. Here, the user sends its original traffic packets to a GAN client, which encapsulates them for transmission through an open network towards the destination. The GAN server extracts the packets from the GAN tunnel traffic and transmits them to the application server. The application server provides the service the users originally intended, such as transferring files from Google Drive. In the opposite direction and, similarly to the operation of the GAN client, packets originating from the application server are forwarded to the GAN server, and after encapsulation, the GAN tunnel server sends them to the GAN client.

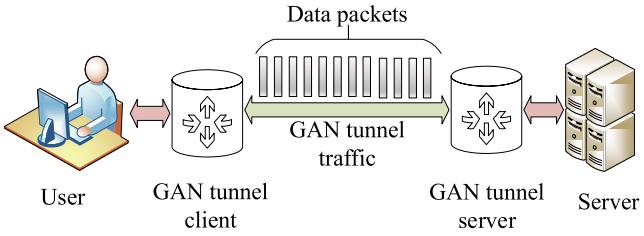

FIGURE 1. WGAN tunnel system overview.

After that, the GAN client extracts these packets and sends the decapsulated packets to the users' devices.

A. DESIGN OF A WASSERSTEIN GAN

To model a target distribution, $\mathbb{P}_r$, first, we define a parametric family of densities (P_g), $g \in \mathbb{R}^d$ [23]. Then, we choose the density that maximizes the likelihood on real data as the target distribution model. With real data samples set of $\{x^{(i)}\}_{i=1}^{m}$, the target distribution is calculated by solving [23]:

$$\max_{g\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} \log P_g(x^{(i)}). \tag{1}$$

GANs model a target distribution by training two competing neural networks, namely a generator and a discriminator. The goal of a generator is to output data samples that closely follow $\mathbb{P}_r$. The generator uses a random variable z with a distribution $p(z)$, such as a uniform distribution, and passes z through a neural network that generates samples with a distribution $\mathbb{P}_g$. The uniform and normal distributions are the common practices for input noise distributions of GANs and it has been shown that they lead to similar results [20], [23]–[26]. For consistency, with most previous works, we use a uniform distribution.

$\mathbb{P}_r$ and $\mathbb{P}_g$ are the distributions of real and generated data, respectively. The goal of the discriminator is to check if $\mathbb{P}_g$ is close to $\mathbb{P}_r$ by using a loss function, such as the cross-entropy [27]. If the discriminator can detect the generated samples from the actual ones, the generator updates its neural network model to improve the generated samples. This competition between the generator and discriminator is a *minimax* game-theory optimization [20]:

$$\begin{aligned}\min_G \max_D V(D,G) &= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] \\ &\quad + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))],\end{aligned} \tag{2}$$

where V is the value function, $D(x)$ is the probability calculated by discriminator that x is from $\mathbb{P}_r$, and $G(z)$ is the generator mapping from the noise distribution to $\mathbb{P}_g$. Here, the generator produces synthetic data from a uniform distribution. Meanwhile, the discriminator learns to differentiate generated and real samples with higher accuracy. The discriminator increasingly fails to distinguish the genuine samples from the synthetic ones as the training of the GAN progresses and the generator improves its modeling of the

traffic distribution. At some point, the generator samples are satisfactory and the discriminator is no longer needed.

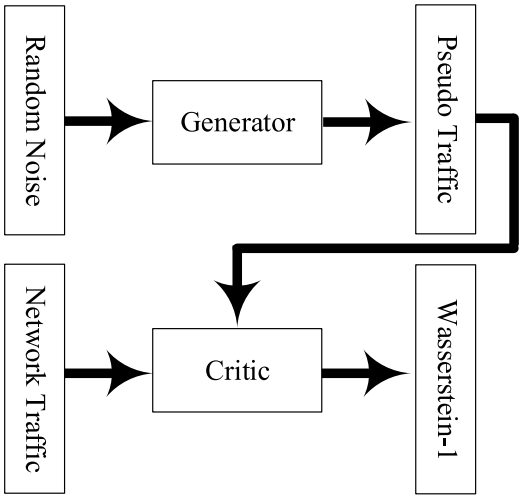

FIGURE 2. WGAN architecture adapted for network traffic generation.

In our model, we implement a version of GAN called Wasserstein GAN (WGAN) [23]. The key advantage of WGAN over GAN is the correlation of its loss function to the quality of generated data [23]. Figure 2 shows an example of our WGAN implementation. In this figure, a random noise is used as the input to the generator. The generator outputs the pseudo traffic (generated traffic). This generated traffic and actual network traffic are the two inputs of the critic. The critic then evaluates the similarity level between generated and actual traffic. WGAN differs from GAN in the way it calculates the difference between synthetic and real data distributions; the loss function in GAN is standard cross-entropy, but in WGAN, the loss function is the Earth-Mover (EM) distance, or Wasserstein-1 [20], [23]. Additionally, in WGAN, the discriminator is called the critic, and it estimates the EM distance between the distributions of actual data and the generated ones, as [23]:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right]. \tag{3}$$

Here, $\prod \left(\mathbb{P}_r, \mathbb{P}_g\right)$ is the set of all joint distributions, $\gamma\ (x, y)$, whose marginal distributions are $\mathbb{P}_r$ and $\mathbb{P}_g$. In other words, the similarity between real and generated data is calculated by finding the infimum of the expected values of distances between data points from the distributions of real and generated data.

B. DATASET INFORMATION AND PREPROCESSING

In our analysis, we use the dataset from [2], which includes about 27,000 TCP flows. We define here a TCP flow as a stream of TCP packets exchanged between a client application and a server. These packets have the same source/destination addresses and port numbers in each direction of the flow [2]. For our WGAN model, we extract a feature set from the header fields of packets in TCP flows.

TABLE 1. Training dataset information before balancing.

	Number of records
Google Chrome	14,561
Google Drive	3,074
One Drive	756
OneNote	4,193
Spotify	3,179
WhatsApp	961

Our feature set includes packet length, packet inter-arrival time, and TCP window size. Afterward, we construct new features that comprise total flow length, packet count, and the minimum, maximum, median, mean, and variance of packet length, packet inter-arrival time, and TCP window size of each flow. We label the flows after the generating applications, namely Google Chrome, Google Drive, One Drive, OneNote, Spotify, and WhatsApp. Table 1 shows the distribution of flows per application. We scale the dataset features to the range $(-1, 1)$ before training our WGAN by using the following normalization function [28]:

$$\frac{X - X_{\min}}{X_{\max} - X_{\min}}, \tag{4}$$

where X represents the values of one feature, and $X_{\min}$ and $X_{\max}$ are the minimum and maximum of X. By fitting the features to be in the same range, the neural network produces results with small errors [29].

C. FLOW GENERATION BY USING WGAN

We build a generator network that outputs TCP flows with 17 constructed features. Table 2 shows the structure of the layers in our WGAN. The generator and critic models in our WGAN are multi-layer perceptron (MLP) neural networks with four and three hidden layers, respectively. We selected these numbers of hidden layers and their units by using the Random Search method [30]. In each hidden layer of the generator, we use the dropout regularization [31]. The dropout regularization prevents overfitting and improves the performance of neural networks [31]. In our WGAN, we use the dropout regularization to keep 80% of units active and to reduce the chance of co-adapting between units, and in turn, to create more robust features [31], [32]. Dropping 20 and 50% of layer units is often found to be optimal for avoiding overfitting [31]. We also use the rectified linear unit (ReLU) and the Leaky ReLU as activation functions in the critic and the generator, respectively. The ReLU is the most commonly used activation function in neural networks [33], [34]. The ReLU assigns a zero gradient to neural network units with negative or zero inputs, and therefore, the units are

TABLE 2. WGAN design.

		Number of Units	Regularization	Activation
Generator Layers	**Input**	100	-	-
	Layer 1	500	20% Dropout	Leaky ReLU
	Layer 2	2,000	20% Dropout	Leaky ReLU
	Layer 3	1,000	20% Dropout	Leaky ReLU
	Layer 4	500	20% Dropout	Leaky ReLU
	Output	17	-	tanh
Critic Layers	**Input**	17	-	-
	Layer 1	500	-	ReLU
	Layer 2	300	-	ReLU
	Layer 3	100	-	ReLU
	Output	1	-	Linear

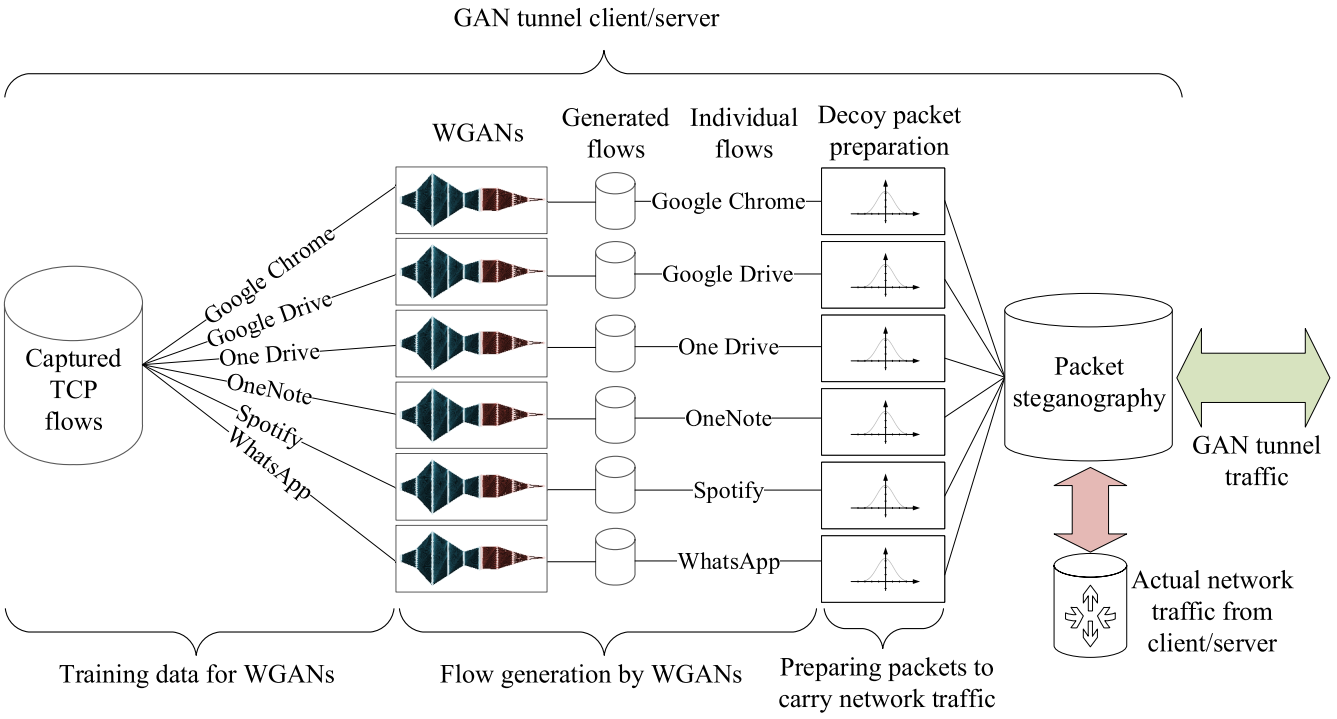

FIGURE 3. Sequence of processes in the GAN Tunnel client/server.

deactivated with such inputs. Simply put, for an input value of x, the ReLU outputs $max(x, 0)$. We use Leaky ReLU functions in the generator layers. Unlike ReLU, the Leaky ReLU allows adding a small gradient to the units when they are inactive. That gradient improves the performance of the neural network by increasing sparsity and dispersion of hidden units activation [35]. The Leaky ReLU's output for an input value of x is αx for $x < 0$ and x otherwise, where α is a small slope coefficient for negative inputs. At the output of the generator, we use hyperbolic tangent (tanh) as the activation function to set outputs between (-1, 1), similar to the scaled input, as discussed in Section II-B. The output layer of the generator has 17 units, each representing a feature of the generated traffic.

In the GAN tunnel, we train a WGAN for each studied application, as Figure 3 shows. The captured TCP flows in

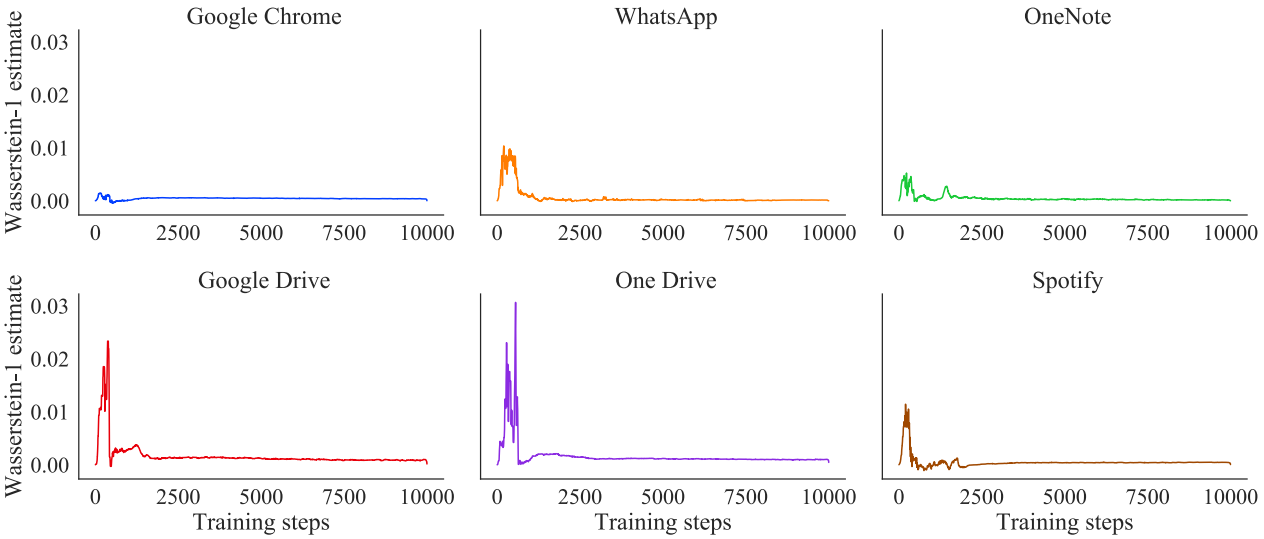

FIGURE 4. Wasserstein-1 values per application.

our training dataset are divided by source applications to train each WGAN; one class per each WGAN. Each WGAN uses the design described in Table 2. Our system comprises six WGANs that are trained and operated in parallel. After training, the WGANs generate synthetic traffic flows, marked as a flow generation stage by WGANs in this figure. The generated flows are then individually fed to a packet preparation module, which generates artificial packets based on the features of the flow. The packet preparation module calculates the packets' requirements, such as their average sizes and the time gap between them matching the flows' features. At the output of the packet preparation module, the information about the generated flows, such as the start and end of flows with TCP handshake packets, and information about data and acknowledgment packets are made available. After that, the detailed information about the packets of the generated flow is fed to the packet steganography module, as Figure 3 shows. At this stage, a connection is created between the GAN tunnel client and server based on the input information of this module. The payloads of the generated packets encapsulate the actual network traffic from client/server at this stage and before transmission. Here, we consider that packet payloads are encrypted. The encapsulated traffic is then transmitted over the tunnel whose flows resemble the decoy application's flows.

D. PACKET STEGANOGRAPHY AND GAN TUNNEL

This module generates the packets as described by the statistics that the WGAN outputs. Then, it encapsulates the packets of the actual traffic with the generated packets. The module generates the information and fields used by the tunnel protocol and the fitting of the packet lengths and inter-arrival gaps. It also provides reassembly information needed after performing segmentation, padding, or concatenation, and adds higher layer protocols needed for securing packet payloads.

As an example, the GAN tunnel may transmit traffic that resembles that originated by Google Chrome (i.e., the decoy application), which carries Spotify data (the actual traffic source). If this module segments the encapsulated traffic over several generated flows, it also creates a control connection between the GAN tunnel client and server by using one of those flows. The control connection carries information about the start and end packets of the segmented flows of actual traffic. The control connection uses transport layer security (TLS) to secure its traffic.

If the GAN tunnel uses TLS to transmit flows, the module performs the required encryption. Without TLS, the tunnel uses a control connection to exchange header encryption keys between the GAN tunnel client and server. We provide an example of packet encapsulation in Section III-F.

III. PERFORMANCE EVALUATION

We run our WGAN on a PC with Microsoft Windows 10. This PC uses an Intel Core i7-8700K processor with 32 GB of memory and an NVIDIA Geforce GTX 1070 graphics card with 8 GB of dedicated memory. We use Keras 2.3 library [36] with TensorFlow 2.0 backend [25] to test our WGAN designs, and Scikit-learn library [28] for ML classification.

A. EVALUATION OF IMPLEMENTED WGANs

To investigate the performance of WGANs (one for each application), we check the values of Wasserstein-1 (3) between the actual and generated traffic at every training step. Figure 4 shows the calculated Wasserstein-1 values for each WGAN after applying a median filter similar to the one used in [23]. Here, the Wasserstein-1 values fluctuate between 0 and 0.03 in the first 2,000 training steps or epochs. An epoch is a cycle of training during which the WGANs go through all records of the training dataset. Afterward, the Wasserstein-1

values converge to almost zero (i.e., smaller than 0.002) for the six considered applications. We train the WGAN up to 10,000 steps because the Wasserstein-1 values are minimal and show enough convergence at this point. These results show that the distributions of the generated network traffic closely follow each of those of the actual traffic.

After calculating the Wasserstein-1 values as shown in Figure 4, we further evaluate the performance of our WGANs by comparing the distributions of generated features with those of the actual network traffic. Figure 5 shows the kernel density estimation (KDE) of the actual and generated network flows. In this figure, we include five features for comparison: packet count, total flow size, mean interarrival time, mean packet length on wire, and mean TCP window size. We show these features to compare the performance of the WGANs for the studied applications. This figure shows that the distribution on each of the traffic features of the WGAN generated traffic closely resembles that of the actual traffic.

As the graphs show, the generated and actual traffic have similar minimum and maximum values. The distributions of the generated packet counts for each of the considered applications are very similar to those of the actual network traffic. For instance, the most frequent number of packets per WhatsApp flow in the actual and generated traffic are 23 and 24, respectively. We also see such similarities in the packet count distributions of each of the considered applications. For instance, OneNote flows have a mean size of 25 packets per flow in both generated and actual traffic.

The number of packets per flow in each of the applications is also similar for the generated and actual traffic. For example, most of the actual Google Drive traffic flows have a total flow size of 10,000 bytes and have a global maximum in the distribution graph at this number. This figure also shows that the generated traffic distribution of Google Drive has local maxima at the same points as the actual traffic. This graph also shows that most of the flows in the actual and generated traffic have similar distributions of flow sizes.

Except for OneNote, other applications have similar minima and maxima in the actual and generated traffic with slightly different kurtosis in their mean interarrival times. Kurtosis is the fourth moment of a distribution that measures how much the data is spread in the tails of the data distribution [37]. We later show that the small discrepancy of OneNote interarrival times between the actual and generated traffic distributions does not affect the performance of the WGAN tunnel against adversarial ITCs.

As Figure 5 shows, the mean packet lengths of the generated and actual traffic for the considered applications have similar distributions. Here, for OneNote, we can see that the generated traffic is more widely spread around local maxima and have a smaller kurtosis than the actual traffic. Despite the differences, the generated and actual traffic have the same local maxima. The mean window sizes of the generated traffic for all applications correctly follow the distribution of those of the actual traffic. Here, the WhatsApp, One Drive, and Spotify WGANs generate more packets at local maxima. Therefore, they have a slightly higher peak value and a larger kurtosis than the actual traffic.

B. TIME COMPLEXITY AND RUN TIME

The time complexity of MLP networks with backpropagation is reported to be $O(PLn^2)$, where P is the dimension input data, L is the number of layers, and n is the number of units in each layer [38]. Here, $O(n^2)$ represents the matrix multiplication time complexity. With the parallelization of matrix multiplication, the time complexity for this function becomes $O(n)$. Subsequently, the time complexity of the MLP network becomes $O(PLn)$. However, with a large n, the processor may not have enough resources to perform the matrix calculations in $O(n)$ time. With parallelization on GPUs, the training time is significantly shorter than that for CPUs [39]. In our work, the time to generate flow features to avoid delays in the GAN tunnel operation is critical. After training our WGANs on the considered applications, we measured the time it takes to generate flow features. Table 3 shows the run time that a trained WGAN takes to generate 1,000 flows. Here, the average time to generate one flow is 7.03×10^{-5} s.

C. EVALUATION OF WGANs AND ITCs TRAINED ON THE SAME DATASET

To evaluate the performance of the GAN tunnel, we test whether the ITCs classify the generated flows as generated by the intended applications and not by the actual ones. In this test, we balance the actual data for each considered application by using a combination of *Synthetic Minority Over-sampling Technique* (SMOTE) [40] and Tomek-links under-sampling method [41] implemented by Imbalanced-learn library [42]. Balancing here refers to changing the number of records in each class (i.e., a category or label) so that they can have a similar number of records to other classes. In SMOTE, the minority class is over-sampled by taking each minority class sample and adding synthetic samples along the line segments joining any or all of the k minority class nearest neighbors [40], [43]. A minority class is a class with the smallest number of records. The combination of SMOTE and the under-sampling method yields effective classification results [40]. In Tomek-links under-sampling, one of the two nearest samples is removed [41]. Moreover, to avoid overfitting the classifier on the training data, we use a stratified 10-fold cross-validation for training the ML algorithms of ITCs. Data balancing is applied to the training portion of each cross-validation fold. After balancing, the number of training samples for each application is 12,000. The proportion of test samples in each fold follows the distribution of the original dataset (Table 1).

We train RF and XGBoost classifiers on this dataset and achieve 0.99 average precision and recall for every category of applications. These classifiers achieve high precision and recall in detecting network-flow patterns [2]. They achieve an average accuracy of 0.99 on training data folds. We then calculate the precision and recall of the classifiers for the

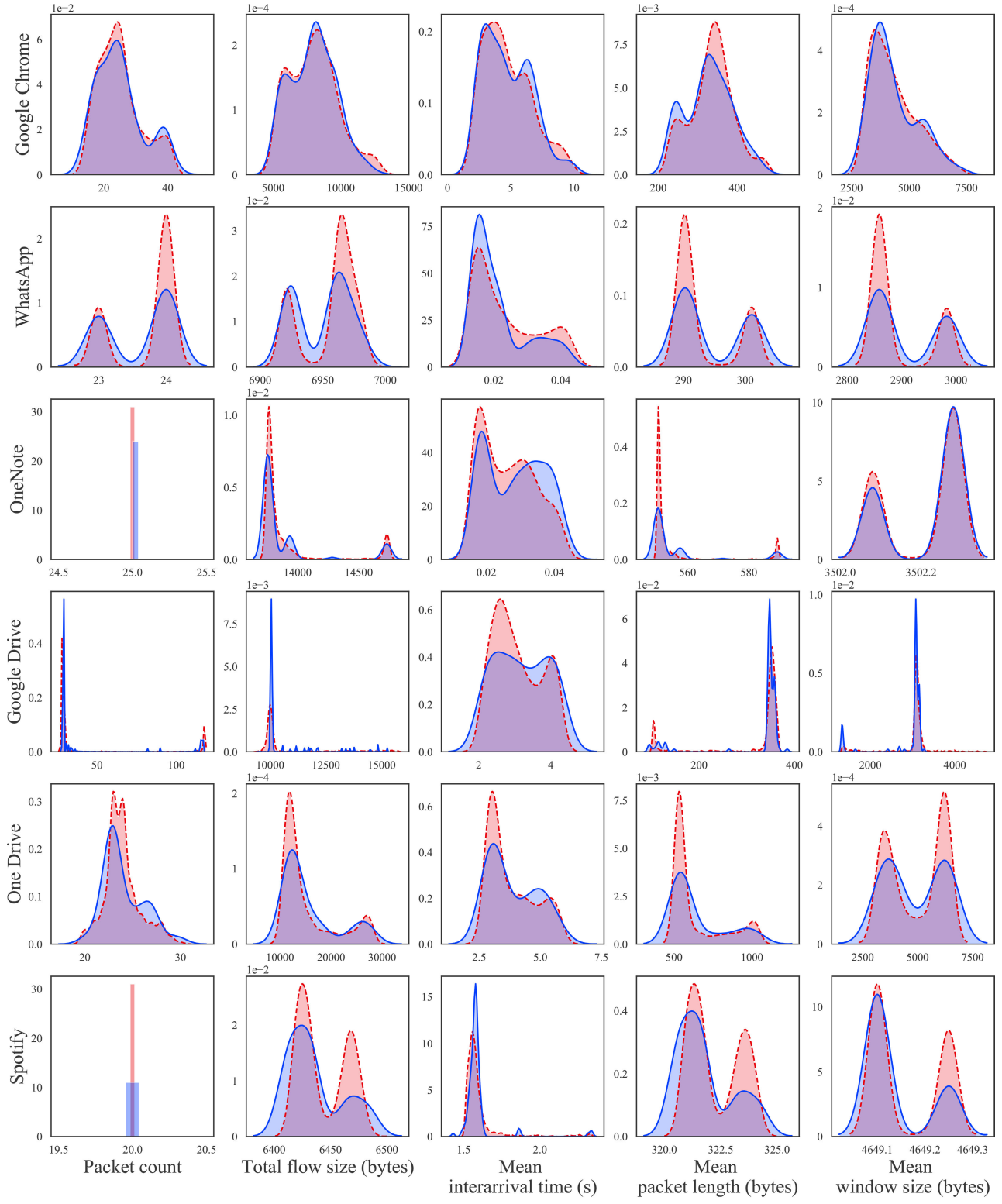

FIGURE 5. Kernel density estimations of generated and actual traffics' features for each application.

TABLE 3. Run time to Generate 1,000 flows.

	Run time (s)
Google Chrome	0.071
Google Drive	0.075
One Drive	0.071
OneNote	0.069
Spotify	0.068
WhatsApp	0.068

generated flows. To exhaustively test our generator, we generated a large number of samples for each application (about 1 million). With more generated samples, the probability of generating noisy samples is inevitable. In this way, we stress test our generator model to assess the quality of our results and ensure that the achieved precision and recall are justified. We found that regardless of the number of generated samples, our generated flows closely resemble actual network traffic.

TABLE 4. RF results on generated data.

	Precision	Recall	F-1 score	Number of generated samples
Google Chrome	1.0	1.0	1.0	998,900
Google Drive	1.0	1.0	1.0	907,200
One Drive	1.0	1.0	1.0	965,200
OneNote	1.0	1.0	1.0	805,100
Spotify	1.0	1.0	1.0	995,500
WhatsApp	1.0	1.0	1.0	970,700

Table 4 shows the classification results for the RF. Here, the table shows that generated traffic fully mimics the traffic of actual traffic. The generated flows for all applications are identified as authentic traffic of the decoy applications by a RF ITC in our tests. This result also shows that the actual application, whose packets are encapsulated by the GAN Tunnel, is not detectable by adversarial ITCs. Table 5 shows the classification results for the XGBoost classifier on the generated flows. XGBoost is a boosting algorithm that calculates the final prediction based on error residuals of prior decision-trees classifiers [22]. XGBoost uses a gradient descent algorithm to minimize a loss function when adding a new decision trees (DT) classifier [22]. XGBoost is found to

TABLE 5. XGBoost results on generated data.

	Precision	Recall	F-1 score	Number of generated samples
Google Chrome	1.0	1.0	1.0	998,900
Google Drive	1.0	0.97	0.99	907,200
One Drive	0.98	1.0	0.99	965,200
OneNote	1.0	1.0	1.0	805,100
Spotify	1.0	1.0	1.0	995,500
WhatsApp	1.0	1.0	1.0	970,700

be a highly sensitive classifier [1]. Here, XGBoost achieves a near-perfect overall accuracy of 0.99. The precision of XGBoost falls to 0.98 for One drive, and its recall falls to 0.97 for Google Drive. These performance losses are negligible, and the ML algorithm still detects the traffic as the decoy traffic with very high accuracy. Therefore, XGBoost is neutralized.

D. EVALUATION OF WGANs AND ITCs TRAINED ON DIFFERENT DATASETS

Here, we demonstrate that the traffic that travels through the GAN tunnel remains anonymous, even when we train the WGANs on data that is not available for training the ITCs. For this, we split our original dataset into two parts, in a stratified manner. With the first part of the dataset, we train our WGANs, and with the second part, we train the ITCs. We balance the portion of the dataset to train ITCs using the combination of SMOTE and Tomek-links under-sampling. We also use a 10-fold stratified cross-validation. The accuracies of both RF and XGBoost on training dataset folds are, on average, 0.99; similar to the previous test.

Table 6 shows the results of a RF ITC on the generated flows based on the split dataset. Here, the precision on Google Chrome is slightly lower than that in the previous test. The achieved precision means that this ITC detects a few false positives in its classification. The recall of Google Drive becomes 0.84, which is lower than that of the previous test. This performance decrease shows that RF ITC detects false negatives in the test results. Overall, with an average accuracy, precision, and recall of 0.97, 0.98, and 0.98, respectively, adversaries who analyze the traffic passing through the GAN tunnel cannot identify the actual application that generated the traffic and the private information it conveys.

Table 7 shows the precision, recall, and F-1 score of XGBoost on the generated flows based on the split dataset. Similar to the results of RF, the performance of XGBoost decreases in this test as compared to those where WGANs and

TABLE 6. RF results on generated data by using split dataset.

	Precision	Recall	F-1 score	Number of generated samples
Google Chrome	0.87	1.0	0.93	999,300
Google Drive	1.0	0.84	0.91	970,700
One Drive	1.0	1.0	1.0	960,900
OneNote	1.0	1.0	1.0	994,800
Spotify	1.0	1.0	1.0	986,400
WhatsApp	1.0	1.0	1.0	974,700

TABLE 7. XGBoost results on generated data by using split dataset.

	Precision	Recall	F-1 score	Number of generated samples
Google Chrome	0.85	1.0	0.92	999,300
Google Drive	1.0	0.82	0.90	970,700
One Drive	1.0	1.0	1.0	960,900
OneNote	1.0	1.0	1.0	994,800
Spotify	1.0	1.0	1.0	986,400
WhatsApp	1.0	1.0	1.0	974,700

ITCs are trained on the same dataset. The average accuracy, precision, and recall of XGBoost in this test are 0.97, 0.98, and 0.98, respectively, and the actual network traffic the GAN tunnel transports remains anonymous.

E. PARAMETER TUNING OF WGAN

As mentioned in Section II-C, we use Random Search to choose the configuration of the WGANs. Table 8 lists the range of parameters for our search. We test the generated samples from the WGAN with selected parameters by RF using the split dataset. Table 9 lists some of the different selected parameters for WGANs. Figure 6 shows the F-1 scores of the RF classifier on generated samples from these WGANs. As the figure shows, WGAN 6 achieves the highest overall F-1 score among the tested designs. Therefore, WGAN 6 is our design choice.

F. PACKET GENERATION FROM WGAN FLOWS

To create the encapsulating packets, we use the flow information generated by the WGAN: the number of packets, flow length, and descriptive statistics of packet lengths, packet interarrival times, and window sizes. For instance,

TABLE 8. Range of parameters of WGANs designs.

Generator hidden layers range	2 - 4
Number of generator units	10, 20, 50, 100, 200, 300, 1,000, 2,000
Generator hidden layers activations	Linear, ReLU, Leaky ReLU
Generator hidden layers regularizations	None, Dropout
Critic hidden layers range	2 - 3
Number of critic units	10, 20, 30, 50, 100, 300, 400, 500

TABLE 9. WGANs with different parameters.

	Generator hidden layers units	Generator activations/ regularizations
WGAN 1	20, 10, 5	Linear/None
WGAN 2	20, 50, 100	Linear/None
WGAN 3	100, 200, 300, 1000	Linear/None
WGAN 4	10, 100, 200, 500	Linear/None
WGAN 5	10, 1,000, 200, 500	Leaky ReLU/ None
WGAN 6	500, 2,000, 1,000, 500	Leaky ReLU/ Dropout
WGAN 7	500, 3,000 , 2,000, 1,000	Leaky ReLU/ Dropout
	Critic hidden layers units	Critic activations
WGAN 1	10, 20	ReLU
WGAN 2	10, 30	
WGAN 3	30, 50, 100	
WGAN 4	10, 30, 50	
WGAN 5	10, 300, 500	
WGAN 6	500, 300, 100	
WGAN 7	500, 400, 300	

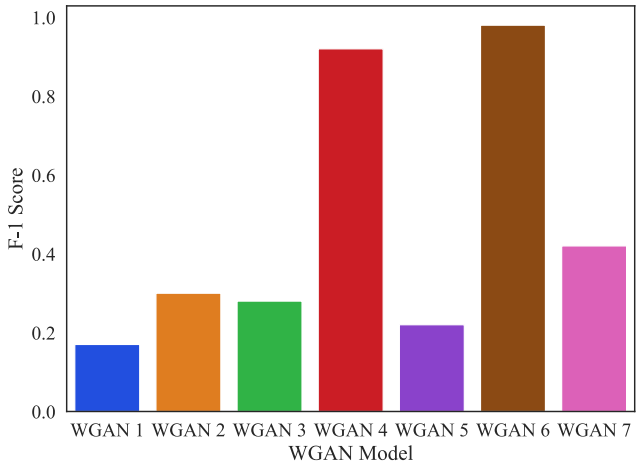

FIGURE 6. F-1 scores of RF on different WGANs. WGAN 6 achieves the highest F-1 score.

we consider a generated Google Drive flow with the features listed in Table 10. Here, we can encrypt the header information of another application, such as Spotify, and encapsulate the entire packet (header and payload) as payloads of the packets of generated packets. In this case, the packet preparation module (Figure 3) reserves six packets for TCP handshakes of the generated flows for data transmission between the GAN tunnel server and client. Let us assume that the first two handshake packets are 66 bytes on wire, and the third one is 54 bytes. Therefore, a total of 372 bytes are subtracted from the total flow length (10,179 bytes), and the remaining 9,807 bytes are used for data transmission. With 23 remaining packets and an average packet length of 352 bytes, the GAN client/server sends packets that have lengths close to the average packet length, except for at least one packet with a size of 1,440 bytes to meet the maximum packet size of the generated flow as Table 10 shows. The interarrival times are, on average, close to 2.5 s (Table 10); therefore, the GAN client/server sends the packets at a fast rate to approach the mean. For the advertised window sizes, we use descriptive statistics of the generated flow to include minimum, maximum, and mean values in the exchanged packets. The median value of window sizes determines the number of packets used for the smallest and largest window sizes of the WGAN generated traffic. For instance, if the average window size is larger than its median (as Table 10 shows), the distribution of the window sizes are skewed to the right, and therefore, we have more packets with window sizes close to the smallest size. Table 11 shows a break-down example of a generated flow of Google Drive (included in Table 10) without using TLS.

With TLS between the GAN tunnel client and server, additional packets are required for TLS establishment. Therefore, we have fewer packets and a smaller number of payload bytes for encapsulating actual traffic. Following the same example of a generated Google Drive flow, as shown in Table 10, from 29 packets, the packet preparation module excludes six packets for connection establishment of TCP and TLS, and three packets for connection termination [44]. The key exchange between a TLS server and client also

TABLE 10. A sample of a synthetic flow of Google drive.

Feature	Value	Feature	Value
Packet count	29	Average packet length	352.2117 bytes
Total flow length	10,179 bytes	Median packet length	91.9119 bytes
Min interarrival time	$9.999\times10^{-}4$ s	Variance packet length	234,802.5781 bytes^2
Max interarrival time	71.2822 s	Min window size	256 bytes
Average interarrival time	2.5568 s	Max window size	64,241 bytes
Median interarrival time	0.02227 s	Average window size	3,081.2285 bytes
Variance interarrival time	165.7221 s^2	Median window size	1,007.7315 bytes
Min packet length	54 bytes	Variance window size	136,675,952 bytes^2
Max packet length	1,439 bytes		

TABLE 11. Overview of packets needed for handshakes and data in a generated flow by our WGAN without using TLS.

Packet Type	Count	Required Length (bytes)
TCP handshake	6	372
Data (max size)	1	1,440
Data	22	8,367

uses about ten packets based on observations of our experimental traces. Therefore, the system has ten packets for data transfer between the GAN tunnel client and server. The payload of these data packets carries actual packets from other applications, such as WhatsApp. The first two TCP handshake packets are 66-byte long, and the last one is 54 bytes, which satisfies the minimum packet length of the flow. A client TLS hello packet may have different sizes. In this example, we consider a 200-byte TLS hello packet. The size of the client hello packet depends on the used cipher suite. An acknowledgment packet from the server follows the hello packet, with a size of 60 bytes. The certificates exchange between server and client may differ slightly in size. A self-signed certificate is about 800 bytes. Here, we assume the use of 1,500 bytes per certificate. Overall, the TLS establishment and termination may require about 6,000 bytes, which are about half of the total length of the generated flow. Therefore, the GAN client/server can only deliver about 4,000 bytes (Table 10) encapsulated data in this flow. From the remaining ten packets, at least one has the maximum packet length unless the TLS certificate already has satisfied this length. Moreover, those remaining packets use the average packet length. The interarrival times and advertised window sizes follow the same principles as described for TCP flows without TLS. Table 12 shows a sample of flow break down in the number of packets for a WGAN-generated flow when we apply TLS to GAN tunnel traffic.

TABLE 12. Overview of packets needed for handshakes and data in a generated flow by our WGAN with TLS.

Packet Type	Count	Required Length (bytes)
TCP handshake	19	6,000
Data (max size)	1	1,440
Data	9	2,560

If the size of the data that a generated flow encapsulates exceeds the byte limit of that flow, the GAN client/server may either pick another and more suitable generated flow, or segments the data and transmit the segments by using more than one flow.

IV. BACKGROUND AND RELATED WORKS

Network traffic generation using GANs has been considered for detecting intrusion, dataset augmentation, or illegality detection [6], [45], [46]. Ring *et al.* [47] introduced flow-based traffic generation using GAN. This work aims to generate realistic network traffic to aid intrusion detection systems (IDS). The challenge that current intrusion detection systems face is the availability of labeled data and the high cost of false positives [47]. Charlier *et al.* [45] used GAN to generate DDoS traffic to improve the accuracy of ITCs in detecting such an attack.

Network traffic data is mostly unbalanced among different categories. For instance, some applications may generate more packets than others. This feature of traffic affects the efficiency of ML-based ITCs as they may tend to favor a category of traffic when performing classification, and that leads the ITC to having biased decisions. To avoid such a problem, balancing methods, such as oversampling the minority categories (i.e., categories with the smallest number of records) or undersampling the majority categories, are employed for classification [40]. There are also advanced techniques that combine oversampling and undersampling methods [48] for data balancing and to achieve higher classification accuracy. An example of this is the combination of SMOTE and Tomek-links under-sampling method, as adopted in this paper. Vu *et al.* [46] used a GAN to generate data for minority categories and augmenting the dataset to create a balanced dataset for classification by ML algorithms. This approach showed that ML classifiers can achieve more effective results in situations when sampling methods underperform.

ITCs may be used for website fingerprinting, which is the process of identifying websites visited by online users. Internet censorship may use such ITCs to identify and censor specific types of traffic. Li *et al.* [6] use GAN to generate features of uncensored traffic and morph the features of censored traffic, to avoid censorship. In this way, censored traffic looks as uncensored traffic. The authors change the features of traffic that goes to *sp0.baidu.com* to those of the traffic that goes to *www.baidu.com*. However, details on the implementation of the system are missing. In our work, rather than just focusing on packet count, which would be easily detectable by a ML classifier, we argue that many traffic features must be considered for an effective classification countermeasure. We use 17 constructed features to generate highly realistic flows, rather than features that do not represent the flows in their entirety. Moreover, we propose that the generated flows carry the actual network traffic as payloaåd; as a tunnel.

Table 13 compares previous works concerning computer and network security systems that use GAN. Here, we describe them with more detail and indicate the difference with the approach proposed in this paper. Cheng [52] use a GAN model to generate network traffic at the network layer. The generated traffic consists of Internet control message protocol (ICMP) packets, domain name system (DNS) queries, and hypertext transfer protocol (HTTP) web requests. Unlike this work, our GAN is trained to generate TCP flow attributes to create a GAN tunnel for data transmission. Our main goal is to counter adversarial Internet traffic classifier. Rigaki and Garcia [51] and Lin *et al.* [50] proposed a GAN model that intrudes a network and evades network intrusion detection systems. Unlike these works, we use a GAN model to create a secure communication tunnel that online users can employ to protect the identity of the application they use. Hu and Tan [49] propose a GAN model to change the API calls of malware applications that look like normal API calls to evade antivirus software. However, this model is only related to computer system security.

TABLE 13. Comparison of objectives of security systems employing GAN.

References	Objective	Defensive / Offensive
Ring et al. [47]	Dataset augmentation to aid IDS	Defensive
Hu et al. [49]	Mask API calls of malware to evade antivirus software	Offensive
Lin et al. [50]	Generating malicious traffic that evades IDS	Offensive
Rigaki et al. [51]	Modify malware network traffic to resemble Facebook chat traffic and evade IDS	Offensive
Cheng et al. [52]	Generation of realistic ICMP pings, DNS queries, and HTTP requests	N/A
Taheri et al. [53]	Dataset augmentation to aid anti-label flipping systems	Defensive
Li et al. [6]	Avoid censorship	Defensive
Our work	Privacy protection against adversarial ITCs	Defensive

TABLE 14. Comparison between architecture and success rates of security systems employing GAN. * Success rate is calculated based on number of flows rather than ML performance results.

References	GAN Architecture	Test Mechanism	Main Measurement Variable	Success Rate (%)
Ring et al. [47]	MLP	Domain knowledge test	Ratio of correctly generated flows/ all generated flows	96-99% *
Hu et al. [49]	MLP	RF, SVM, DT, MLP, LR, and Voting	True positive rate	93-95%
Lin et al. [50]	MLP	RF, SVM, DT, MLP, NB, LR, and KNN	True positive rate	29-100%
Rigaki et al. [51]	LSTM	Stratosphere Linux IPS [54]	Ratio of successful/generated flows	55-90% *
Cheng et al. [52]	CNN	Experimental TCP/UDP test	Ratio of successful/generated packets	66-99% *
Taheri et al. [53]	CNN	RF, SVM, DT, MLP, and CNN	Average F-1 score	5-85%
Li et al. [6]	MLP	SVM, NB, and Jaccard's coefficient	Area under receiver operating characteristic	2-88%
Our work	MLP	RF and XGBoost	Average F-1 score	97-98%

Table 14 compares the GAN architecture and test mechanisms of the works presented in Table 13. Depending on the type of features that we seek to generate, different types of GAN may enhance the generation of new traffic. For instance,

Cheng [52] generate network traffic at the byte-level while others generate traffic at the packet and flow levels.

As shown in this table, the test mechanisms considered in these works are domain knowledge test, ML algorithms, and experimental tests. Ring *et al.* [47] test generated flows based on the domain knowledge, such as checking that UDP packets do not include TCP flags. They measure the success rate by calculating the ratio of correctly generated flows over the total number of generated flows, and they achieve 96 to 99% success. Hu and Tan [49] test the effectiveness of their model by using ML algorithms to classify their generated application programming interface (API) calls as non-malicious. The ML algorithms used are: RF, SVM, DT, MLP, logistic regression (LR) [55], and a voting classifier [56]. In this work, 93 to 95% of the generated API calls is classified as non-malicious. Lin *et al.* [50] also use ML algorithms as their test mechanism to check how much of the generated traffic is classified as normal. They use RF, SVM, DT, MLP, LR, Naive Bayes (NB) [57], and K-nearest neighbors (KNN) [43]. They achieve a true positive rate that ranges from 29 to 100%. Rigaki and Garcia [51] use a third-party software named Stratosphere Linux IPS [54] to find out the proportion of the generated flows that passes as normal traffic through this intrusion prevention software. Here, the success rate is measured by the ratio of successfully generated flows over the total generated ones resulting in 55 to 90% success rate. Cheng [52] test the generated packets by sending them out as actual network packets and test them by observing if they receive a successful response. In this test, the packet generation achieves 66 to 99% success. Li *et al.* [6] measure the success rate of their model by using SVM, NB, and Jaccard's coefficient [58] to classify the generated traffic. The measurement variable they have used is the area under operator characteristic (true positive rate vs. false positive rate) curve. With results closer to 0.5, the classifier is unable to identify the generated traffic from uncensored traffic and they achieve results in the range of 0.56 to 0.99 (2-88% success). Taheri *et al.* measure the effectivity of their anti-label flipping mechanism by using ML algorithms. They achieved average F-1 scores that range from 5 to 85% in defeating data flipping systems. In our work, we use XGBoost and RF to test the detectability of the generated flows. XGBoost is a robust tool to classify network traffic [2]. However, other works in Table 14 have not considered XGBoost for testing.

In our tests, we showed that the generated flows are identified as decoy application traffic with average F-1 scores of 97 and 98%. Compared to previous works that focus on network traffic generation [47], [50]–[52], the range of our success rate is more effective because it has smaller variability.

Convolutional neural networks (CNN) are a good choice for byte-level traffic generation because they are mainly used in image classification, where the input data are formed from vectors of RGB values for each image pixel. Another choice for traffic generation is long short-term memory (LSTM) [51], [59]. LSTMs are mainly used to model time-series data, where the model benefits from both historical and recent data. In other words, LSTM aims to learn the time evolution of sequences of data [60]. Such a model may be fit to generate a user datagram protocol (UDP) stream of data as UDP streams are not divided into flows of traffic. In our work, we use a fully connected network or MLP. In such a network, each unit of each layer is connected to all the units in the next layer. Each layer of a fully connected network represents a different transformation of data in feature space. Functions, such as Fourier and Laplace transforms, are used to simplify complicated equations and to bring them into a suitable format for analysis. Similarly, a fully connected network helps to find a suitable transformed data representation that simplifies tasks of classification or data generation [60]. Therefore, we use MLP to represent a vector of input noise, as flows, in our dataset.

Other previous works resort to non-dynamic methods of countering ITCs. Fu *et al.* [18] analyzed packet padding and delay to counter ping-probing attacks, which are used to identify user traffic rate based on ping round-trip time. The authors concluded that randomly delaying the packets is effective in countering ping probing attacks. This work, however, does not consider ML-based ITCs. Fu *et al.* [19] studied the effectiveness of packet padding to counter statistical analysis on traffic such as sample mean, sample entropy, and sample variance. They show that packet padding can decrease detection rates to about 40%. Wright *et al.* [16] used the chi-squared test to specify the language spoken during a voice over IP (VoIP) call as a binary classification. For instance, they predict whether the language used in a VoIP call is English or not. They applied packet padding to VoIP packets to reduce the classification accuracy to about 27%.

V. CONCLUSIONS

In this paper, we introduced a GAN tunnel to counter adversarial Internet traffic classifiers (ITCs). Our proposed GAN tunnel uses generated network traffic that mimics actual network traffic to avoid identification/classification of the originating application by adversarial ITCs. We designed a Wasserstein GAN (WGAN) with a generator comprising four hidden layers and a critic with three hidden layers. The WGAN uses traffic from original user applications for modeling the traffic of decoy applications. In this way, a user may configure the WGAN to generate traffic of a selectable decoy application.

We used a dataset with 27,000 captured TCP flows consisting of traffic of six different users for a collective duration of 28 days. We use the TCP flows generated by the WGAN to carry the actual network traffic as payload. In this way, an adversary who aims to detect the originating applications would classify the WGAN-generated flows as being generated by the decoy applications.

We compared the distribution of generated flows to the distribution of actual network traffic of six popular applications, and we showed that the distribution of the generated flows closely follows that of the actual network traffic. We tested

the efficacy of our proposed GAN tunnel traffic by evaluating the classification performance of RF and XGBoost, which are qualified examples of effective adversarial ITCs. The results show that these classifiers detect the flows as being generated by the decoy applications with an average accuracy of 0.99 when these classifiers and the WGAN are trained on the same dataset, and 0.97 when these classifiers and WGAN are trained on two separate datasets. Our tests show that the GAN tunnel effectively masks the statistical properties of the traffic generated by user applications and that it can make generated traffic effectively appear as generated by a selectable decoy application.

ACKNOWLEDGMENT

The authors would like to thanks the anonymous reviewers for their technical comments and suggestions, which helped to improve this article.

REFERENCES

[1] S. Fathi-Kazerooni, Y. Kaymak, and R. Rojas-Cessa, "Tracking user application activity by using machine learning techniques on network traffic," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Feb. 2019, pp. 405–410.

[2] S. Fathi-Kazerooni, Y. Kaymak, and R. Rojas-Cessa, "Identification of user application by an external eavesdropper using machine learning analysis on network traffic," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, May 2019, pp. 1–6.

[3] P. Perera, Y. C. Tian, C. Fidge, and W. Kelly, "A comparison of supervised machine learning algorithms for classification of communications network traffic," in *Neural Information Processing* (Lecture Notes in Computer Science), vol. 10634, D. Liu, S. Xie, Y. Li, D. Zhao, and E. S. El-Alfy, Eds. Cham, Switzerland: Springer, 2017.

[4] T. T. T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 4th Quart., 2008.

[5] B. A. Forouzan, *TCP/IP Protocol Suite*. New York, NY, USA: McGraw-Hill, 2002.

[6] J. Li, L. Zhou, H. Li, L. Yan, and H. Zhu, "Dynamic traffic feature camouflaging via generative adversarial networks," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Jun. 2019, pp. 268–276.

[7] Z. Cao, G. Xiong, Y. Zhao, Z. Li, and L. Guo, "A survey on encrypted traffic classification," in *Applications and Techniques in Information Security* (Communications in Computer and Information Science), vol. 490, L. Batten, G. Li, W. Niu, and M. Warren, Eds. Berlin, Germany: Springer, 2014.

[8] J. Khalife, A. Hajjar, and J. Diaz-Verdejo, "A multilevel taxonomy and requirements for an optimal traffic-classification model," *Int. J. Netw. Manage.*, vol. 24, no. 2, pp. 101–120, Mar. 2014.

[9] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–5.

[10] F. Zhang, W. He, X. Liu, and P. G. Bridges, "Inferring users' online activities through traffic analysis," in *Proc. 4th ACM Conf. Wireless Netw. Secur. WiSec*, 2011, pp. 59–70.

[11] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," 2017, *arXiv:1709.02656*. [Online]. Available: http://arxiv.org/abs/1709.02656

[12] A. K. Sharma and P. S. Parihar, "An effective dos prevention system to analysis and prediction of network traffic using support vector machine learning," *Int. J. Appl. or Innov. Eng. & Manage.*, vol. 2, no. 7, pp. 249–256, 2013.

[13] G. Sun, T. Chen, Y. Su, and C. Li, "Internet traffic classification based on incremental support vector machines," *Mobile Netw. Appl.*, vol. 23, pp. 789–796, Feb. 2018.

[14] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proc. 10th Annu. ACM Workshop Privacy Electron. Soc. WPES*, 2011, pp. 103–114.

[15] Z. Fan and R. Liu, "Investigation of machine learning based network traffic classification," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2017, pp. 1–6.

[16] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *Proc. 16th Netw. Distrib. Secur. Symp.* IEEE, 2009, pp. 237–250.

[17] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 332–346.

[18] X. Fu, B. Graham, R. Bettati, and W. Zhao, "Active traffic analysis attacks and countermeasures," in *Proc. Int. Conf. Comput. Netw. Mobile Comput. ICCNMC*, Oct. 2003, pp. 31–39.

[19] X. Fu, B. Graham, R. Bettati, W. Zhao, and D. Xuan, "Analytical and empirical analysis of countermeasures to traffic analysis attacks," in *Proc. Int. Conf. Parallel Process.*, Oct. 2003, pp. 483–492.

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[21] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[22] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

[23] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 214–223.

[24] B. Bailey and M. J. Telgarsky, "Size-noise tradeoffs in generative networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 6489–6499.

[25] M. Abadi *et al.* (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software. [Online]. Available: http://tensorflow.org

[26] E. Heim, "Constrained generative adversarial networks for interactive image generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10753–10761.2015.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[29] M. Shanker, M. Y. Hu, and M. S. Hung, "Effect of data standardization on neural network training," *Omega*, vol. 24, no. 4, pp. 385–397, Aug. 1996.

[30] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] P. Baldi and P. J. Sadowski, "Understanding dropout," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2814–2822.

[33] K. Jarrett, K. Kavukcuoglu, M. A. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. IEEE 12th Int. Conf. Comput. Vis.*, Sep. 2009, pp. 2146–2153.

[34] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 807–814.

[35] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, no. 1, 2013, p. 3.

[36] F. Chollet *et al.* (2015). *Keras*. [Online]. Available: https://keras.io

[37] K. P. Balanda and H. L. Macgillivray, "Kurtosis: A critical review," *Amer. Statistician*, vol. 42, no. 2, pp. 111–119, May 1988.

[38] J. J. Jenq and W. Li, "Feedforward backpropagation artificial neural networks on reconfigurable meshes," *Future Gener. Comput. Syst.*, vol. 14, nos. 5–6, pp. 313–319, Dec. 1998.

[39] K.-S. Oh and K. Jung, "GPU implementation of neural networks," *Pattern Recognit.*, vol. 37, no. 6, pp. 1311–1314, Jun. 2004.

[40] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002.

[41] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, no. 6, pp. 448–452, Jun. 1976.

[42] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *J. Mach. Learn. Res.*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html
[43] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Trans. Syst., Man, Cybern.*, vols. SMC–6, no. 4, pp. 325–327, Apr. 1976.
[44] I. Grigorik, *High Performance Browser Networking: What Every Web Developer Should Know About Networking and Web Performance*. Newton, MA, USA: O'Reilly Media, 2013.
[45] J. Charlier, A. Singh, G. Ormazabal, R. State, and H. Schulzrinne, "SynGAN: Towards generating synthetic network attacks using GANs," 2019, *arXiv:1908.09899*. [Online]. Available: http://arxiv.org/abs/1908.09899
[46] L. Vu, C. T. Bui, and Q. U. Nguyen, "A deep learning based method for handling imbalanced problem in network traffic classification," in *Proc. 8th Int. Symp. Inf. Commun. Technol. SoICT*, 2017, pp. 333–339.
[47] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Comput. Secur.*, vol. 82, pp. 156–172, May 2019.
[48] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD Explor. Newslett.*, vol. 6, no. 1, pp. 20–29, Jun. 2004.
[49] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," 2017, *arXiv:1702.05983*. [Online]. Available: http://arxiv.org/abs/1702.05983
[50] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative adversarial networks for attack generation against intrusion detection," 2018, *arXiv:1809.02077*. [Online]. Available: http://arxiv.org/abs/1809.02077
[51] M. Rigaki and S. Garcia, "Bringing a GAN to a knife-fight: Adapting malware communication to avoid detection," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2018, pp. 70–75.
[52] A. Cheng, "PAC-GAN: Packet generation of network traffic using generative adversarial networks," in *Proc. IEEE 10th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Oct. 2019, pp. 0728–0734.
[53] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, A. Miri, and M. Conti, "On defending against label flipping attacks on malware detection systems," *Neural Comput. Appl.*, pp. 1–20, Mar. 2020.
[54] S. Garcia, "Modelling the network behaviour of malware to block malicious patterns. the stratosphere project: A behavioural ips," in *Proc. Virus Bull.*, Sep. 2015, pp. 1–8. [Online]. Available: https://www.virusbtn.com/pdf/conferenceslides/2015/GarciaVB2015.pdf
[55] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic Regression*. New York, NY, USA: Springer, 2002.
[56] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Mach. Learn.*, vol. 36, nos. 1–2, pp. 105–139, 1999.
[57] I. Rish, "An empirical study of the naive Bayes classifier," in *Proc. IJCAI Workshop Empirical Methods Artif. Intell.*, vol. 3, no. 22, pp. 41–46, 2001.
[58] R. Real and J. M. Vargas, "The probabilistic basis of Jaccard's index of similarity," *Systematic Biol.*, vol. 45, no. 3, pp. 380–385, Sep. 1996.
[59] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[60] B. Ramsundar and R. B. Zadeh, *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. Newton, MA, USA: O'Reilly Media, 2018.

SINA FATHI-KAZEROONI received the B.Sc. degree in electrical engineering from Shiraz University, Shiraz, Iran, in 2014, and the M.Sc. degree in electrical engineering from the New Jersey Institute of Technology (NJIT), Newark, NJ, USA, in 2016, where he is currently pursuing the Ph.D. degree with the Networking Research Laboratory, Helen and John C. Hartmann Department of Electrical and Computer Engineering. He is also a Research Assistant with the Networking Research Laboratory, Helen and John C. Hartmann Department of Electrical and Computer Engineering, NJIT. His research interests include wireless communications networks, deep learning, and statistics.

ROBERTO ROJAS-CESSA (Senior Member, IEEE) received the B.S. degree from Universidad Veracruzana, Mexico, the M.S. degree from the Research and Advanced Studies Center, Mexico, and the Ph.D. degree in electrical engineering from Polytechnic University (now the New York University Tandon School of Engineering, Polytechnic Institute), Brooklyn, NY, USA. He was a Visiting Professor with Thammasat University, Thailand, and a Researcher with the University of Electro-Communications, Tokyo, Japan. He is currently a Professor with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology. He is also the Director of the programs M.S. in telecommunications and M.S. in Internet engineering. He has authored the books *Advanced Internet Protocols, Services, and Applications* (Wiley and Sons, 2012) and *Interconnections for Computer Communications and Packet Networks* (CRC Press, 2017). His research interests include networking, cyber-physical systems, energy, sustainability, and intelligent systems. He was an Invited Fellow of the Japanese Society for the Advancement of Science, in 2009. He is a member of the ACM, IEEE, and other societies. He serves in different capacities for the IEEE conferences and IEEE journals and as a Panelist for the U.S. National Science Foundation and the U.S. Department of Energy. He was a recipient of the Excellence in Teaching Award 2013 from the Newark College of Engineering, NJIT. He was a recipient of the New Jersey Inventors Hall of Fame—Innovators Award, in 2013. He was the Chair for the IEEE Sarnoff Symposium 2012 and the IEEE International Conference of High-Performance Switching and Routing 2020 and the TPC Chair for the flagship conferences of the IEEE Communications Society, the International Conference on Communications, and GLOBECOM. Email: rojas@njit.edu.