PAC-GAN: Packet Generation of Network Traffic using Generative Adversarial Networks

Adriel Cheng
Cyber and Electronic Warfare Division
Defence Science and Technology, Department of Defence, Australia
Adriel.Cheng@dst.defence.gov.au

Abstract — **Generative adversarial networks (GANs) have proven extremely successful in creating artificial yet highly realistic media data such as images, text, audio and videos. In this paper, we adapt and describe a GAN method for creating network traffic data at the IP packet layer. Generating realistic network traffic is essential for development and testing of techniques in Cyber and network security related tasks such as anomaly or intrusion detection. The effectiveness of such network monitoring techniques depends directly on the traffic data. Despite this, traffic generation is often considered low priority because it is difficult. Using GANs, we prototype and prove feasibility in the generation of real traffic flows such as ICMP Pings, DNS queries, and HTTP web requests. Using Convolutional Neural Network (CNN) GANs, we propose an alternative encoding of network traffic data into the CNN model. Experiments show our generated traffic can be successfully transmitted through the Internet eliciting desired responses from the network. The work described in this paper is the first step in demonstrating successful creation and transmission of network traffic; it forms the basis for future GAN based traffic generators at scale.**

Keywords — Deep-Learning, Generative Adversarial Networks, Convolutional Neural Networks, Test Generation, Traffic Generation, Packet Encoding, Network Security, Cyber Security

I. INTRODUCTION

Enforcing network security requires development and ongoing maintenance of anomaly and intrusion detection tools. Many such Cyber security tools employ machine-learning (ML) algorithms to learn and differentiate various forms of traffic (both normal and abnormal) [1,2]. For example, supervised ML classifiers rely on procuring examples of different traffic types (including malicious traffic), each labelled appropriately so the ML classifier can learn how to identify these traffic flows.

Synthetic traffic generators are currently available, but traffic is often unrealistic or suffer from data generation bias [3]. Commercial test generators can be effective but are costly [4]. On the other hand, capturing realistic network traffic from the network is easy (e.g. at border gateways of Enterprise networks), however, it is difficult to establish ground-truth for such traffic. Publicly available data sets also suffer from similar shortcomings such as incorrect or insufficient labelling, can become outdated, or are obfuscated for privacy reasons such that useful real traffic collected in its original form cannot be shared. The challenge with traffic generation for network security R&D is to produce a large corpus of realistic and labelled traffic data using cost-effective means.

Generative adversarial networks (GANs) have proven extremely effective at generating seemingly realistic but artificial images, videos, audio, etc from prior-known sources [5-9]. For example, a fake but highly realistic video of former president Obama was recently created using GANs [8]. The strength in GANs lies with learning the intricate data distributions within an application domain and reproducing meticulously similar but subtle variants of the data. For instance, image GANs learn the complex inter-pixel patterns and relations to produce similar but user/algorithmic-directed variants of the same image (Fig. 1).

From a network traffic generation perspective, our aim is to investigate if GANs can learn the diverse types of traffic (and intra traffic variants) instigated from users and machines on a network. We tackle the following research question: "*Can realistic network traffic be regenerated from GANs?"* (Fig. 1). Inspired by image based GANs, we devise a method to create real network traffic using GANs. The focus of this paper is to demonstrate that real network traffic flows can be generated, and can be generated realistically to be successfully transmitted out through the Internet to elicit appropriate network responses.

The contributions of this paper are:

- GAN techniques are proposed, devised and adapted to generate realistic traffic flows at the IP packet level. Existing GAN traffic generators only operate at the traffic flow or metadata level [10,11,12].
- We demonstrate how convolutional neural network (CNN) GANs, which are popular for image generations [7], can be applied to network traffic generation. To facilitate this, a network traffic encoding scheme is proposed to convert and map network traffic data from the IP domain into image based matrix representations typically used in CNN frameworks.

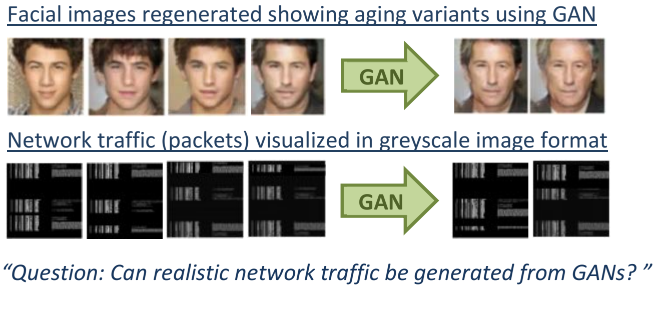
Facial images regenerated showing aging variants using GAN
GAN
Network traffic (packets) visualized in greyscale image format
GAN
"Question: Can realistic network traffic be generated from GANs? "

Fig. 1. Traffic generation versus image generation using GANs

978-1-7281-2530-5/19/$31.00 ©2019 Crown

We prototype our CNN GAN traffic generator, named PAC-GAN, to generate network packets. Using PAC-GAN, we conduct experiments to demonstrate and measure performance in successfully generating and sending traffic to the wider Internet. For initial prototyping, we create three common network traffic types, namely ICMP Pings, DNS queries and HTTP requests.

The work described in this paper aims to validate the concept of using GANs for creating real network traffic packets that adhere to network standards/protocol-layers for transmitting through actual networks. The paper is our first step towards large scale traffic generation using GANs.

The next section provides a brief literature survey followed by background preliminaries. Sections IV to VI describe our GAN network traffic generator and prototype. Section VII presents experimental results before the paper concludes.

II. Literature Survey

Using GANs for creating network traffic is a relatively new concept, with few published work reported. Thus far, only five papers from 2018 and one from 2017 have been uncovered. Of closest relevance to this paper is the flow-based traffic generator research by Ring et al. [10]. Using a GAN, their traffic generator learns characteristics of collected network traffic to reproduce synthetic traffic for testing intrusion detection systems. Their contributions focus on three pre-processing schemes to transform continuous data into categorical attributes common in traffic flows. However, existing GAN systems are already available to deal with discrete categorical data. Furthermore, applying discrete data to GANs can be tackled using other less complex means such as the network packet encoding scheme described in Section V. Their generator also does not produce network traffic directly; instead only aggregating IPs, port numbers, number of packets and bytes, and flow timing metadata values mimicking traffic flows.

In [1], Rigaki et al. propose using GANs to generate traffic at the flow parameter level. Like [10], their generator does not produce actual network traffic. Instead, flow timing parameters such as flow duration and frequency, and number of bytes are generated. Their GAN learns the traffic profiles of benign applications such as Facebook messaging, and uses the generated flow parameters to manipulate the characteristics of malicious traffic flows such that command-and-control (C2) traffic are disguised as Facebook traffic. GANs implemented using Recurrent Neural Networks (RNNs) are utilized to learn sequential timing properties critical in disguising C2 traffic. In this paper, we focus on CNN GANs instead. Unlike our approach where different traffic types can be generated, their GAN focuses solely on this single Facebook traffic type.

In other related GAN traffic generation research, generating realistic or usable traffic is considered low priority. GANs are used simply to produce supplementary test traffic to enhance intrusion detection systems (IDS). For example, IDSGAN [2], Bot-GAN [11], and MalGAN [12] are frameworks that generate synthetic malicious traffic specifically for improving and testing the robustness of their IDS against a wider variety of (unknown) malicious attacks. Unlike this paper, we observe that the focus of those papers is solely the discriminators rather than generator models of the GAN.

The papers surveyed construct GANs using either simplistic feed-forward multi-layer neural networks or RNNs. In this paper, we employ CNN GANs based on our prior work which showed that network traffic at the packet level is well suited for CNNs [13]. Interestingly, Rigaki et al. also plan to implement CNN GANs as future work [1].

In comparison to previous work, our GAN traffic generator produces network traffic at the individual packet byte level. All existing GAN traffic generators generate traffic at the metadata or flow-level. However, generating traffic at metadata/flow-level offers much less control (and flexibility) in the types of traffic that can be synthetically produced. The papers surveyed can only influence high-level traffic profile characteristics like timing or overall traffic flow size parameters. In contrast, our GAN can learn and manipulate packet byte values to re-create realistic variants of different types of traffic. As far as we are aware, using GANs for network traffic generation at the IP packet level is an approach that has not been attempted previously.

III. Background Preliminaries

A. Generative Adversarial Networks

Generative Adversarial Networks (GANs) are generative models based on adversary and deep learning methodologies. The primary goal of GANs is to learn and mimic the distribution (and characteristics) of some data of interest, and re-create similar (or directed variants) of the same data type. GANs consist of two neural network models, a discriminator and a generator (Fig. 2). The discriminator is trained to classify authenticity of its input data. The generator is trained to re-produce new synthetic data that is indistinguishable from authentic data. Together, the discriminator and generator models are trained in step with one another, but with opposing objective (and loss) functions to each other.

In Fig. 2, the discriminator's objective function is to maximize its likelihood of distinguishing between real data (from the training set) and fake data from the generator. Using cross-entropy to measure loss, $\log(D(x))$, $D(x)$ is ideally 1 for labelled real training data x; and for the fake data z generated from the generator G, its loss is $\log(1 - D(G(z)))$. The training process then involves maximizing the discriminator's objective for classifying real and fake data driven by its loss measurement feedback loop (Equation (1) in Fig. 2). Training of the generator is driven by another backpropagation feedback loop from the discriminator, where the objective is to minimize the success of the generator's fake data being detected (Equation (2) in Fig. 2). The complete GAN objective function is encapsulated in Equation (3) in Fig. 2.

GAN training involves the discriminator and generator working against one another, similar to a zero-sum mini-max game. Training continues until the GAN model converges whereby (i) the discriminator can detect sufficiently minor differences between real and fake data, and (ii) the generator produces data that the discriminator has very low success of detecting. When only either (i) or (ii) is achieved, the GAN model has failed and re-training is required.

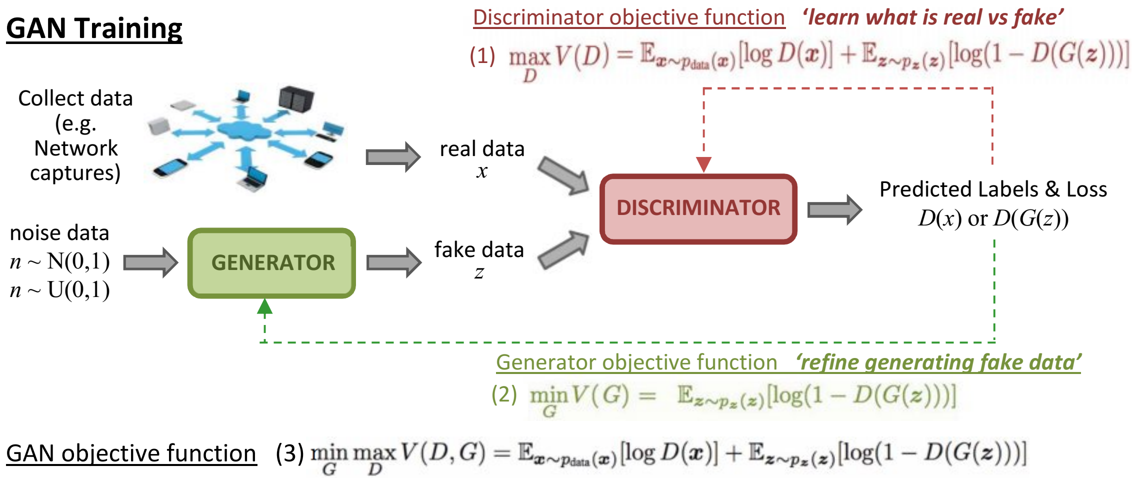
GAN Training
Discriminator objective function 'learn what is real vs fake'
(1) $\max_D V(D) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$
Collect data (e.g. Network captures)
real data x
noise data $n \sim \mathrm{N}(0,1)$ $n \sim \mathrm{U}(0,1)$
GENERATOR
fake data z
DISCRIMINATOR
Predicted Labels & Loss $D(x)$ or $D(G(z))$
Generator objective function 'refine generating fake data'
(2) $\min_G V(G) = \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$
GAN objective function (3) $\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$

Fig. 2. Objective Functions and Procedural Flows in Training Generative Adversarial Networks

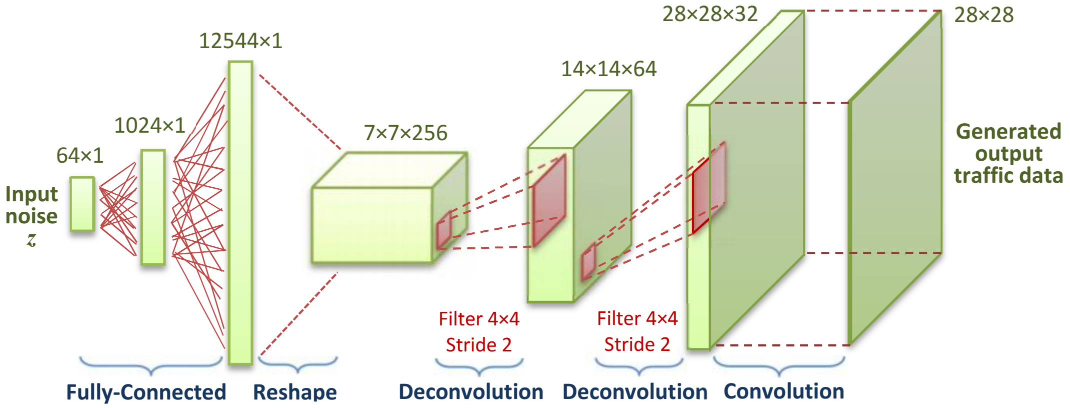

Fig. 3. An Example Generator Model Architecture with Inverse Convolutional Neural Nets

B. Convolutional Neural Network GANs

In this paper, convolutional neural network (CNN) GANs are utilized. CNN GANs use CNN models for both the discriminator and generator. The discriminator uses conventional CNNs commonly deployed for supervised classification [13]. For the generator, an 'inverse' CNN architecture is implemented (Fig. 3). Instead of standard convolutions, filtering and max-pooling operations which down-sample data from higher dimensions to a single classifier prediction value, the generator model consists of multiple layers of deconvolution operations. These deconvolution operations take lower dimensional data vectors and up-sample the data into higher dimensional data until the desired output sized data is produced.

The CNN GAN generator model is often described as a CNN classifier constructed and operating in 'reverse'. In an image based GAN, the generator up-samples the generator noise input to generate desired synthetic images of required dimensional sizes. For our traffic generator, we up-sample from input vector z to generate the required network packet byte values of appropriate sizes for the network traffic type.

IV. Technique

The proposed framework for generating network traffic using GANs is depicted in Fig. 4. First, the GAN must be trained such that the generator model has learned how to reproduce sufficiently realistic and network compliant traffic data. In our case, training of the discriminator is considered secondary, and its purpose is to aid training of the generator. In order to generate network traffic, real network traffic is required as a training dataset. The training data enables the GAN generator to learn how to re-construct real traffic as closely as possible; whilst the GAN discriminator learns how to discern real training traffic from synthetic variants. The random noise initially supplied to the generator could be sampled from the latent space of the training data. After training, the resultant GAN generator model is decoupled from the GAN framework and then tested (or deployed) to generate traffic to be dispatched out through the network and/or Internet.

A. Convolutional Neural Network GANs for Network Traffic

CNN GANs have proven extremely successful at re-producing (and converting) highly realistic images for different applications.

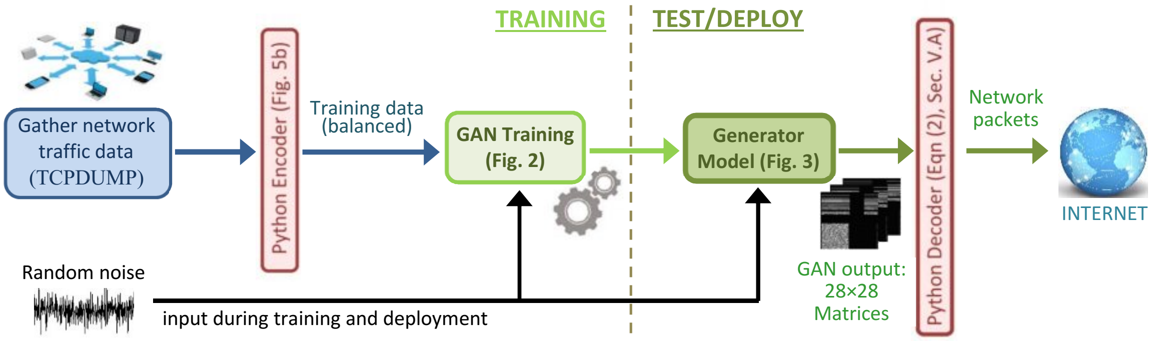

Fig. 4. CNN GAN Traffic Generator Framework with GAN Training and Generator Test/Deployment Phases

An initial objective of our research was to investigate if such image regeneration success could be replicated for the network traffic generation domain; hence we implemented similar CNN models typically used for image generation.

Additionally, the use of CNNs for classifying network traffic has been studied extensively and proven effective (which aids the GAN discriminator). Our prior work in this area also supported the use of CNN for dealing with network traffic data. In [13], we analyzed how network traffic (e.g. network packet byte values) can be represented directly as $n \times n$ square matrix pixel images, and were successfully classified using CNNs in the same way images are classified.

The goal of our GAN traffic generator is to generate the actual byte values of network packets traversing through the network. Initially, we encoded our training and GAN generated network traffic data as $n \times n$ pixels images (i.e. CNN square matrices). Every single network packet byte value was simply mapped to a single pixel in the corresponding square matrix. The sequence of packet byte values were mapped in series along each row of pixels (Fig. 5a).

Using this initial traffic representation, preliminary CNN GANs were created. However, the traffic generated was very poor. For instance, less than 30% accuracy in the generated packet byte values was achieved, and none of the generated packets achieved a successful response.

Examining why such poor generations were produced, we make two observations. First, for many packets, the CNN GAN was able to re-create packet bytes close to the desired values as per the training dataset. Upon closer inspection, these generated packets differed by very small margins from their desired value. Intuitively, from an image generation perspective, this is not unexpected. CNN GANs are able to re-produce strikingly realistic re-creation of images to the casual observer viewing the training and re-created images side-by-side. However, if one were to examine the actual pixel values, it is unlikely the pixel values between these images would be exactly the same. There is likely some very minor variation in pixel values, but to the naked-eye, the images appear identical.

Whilst such behavior is acceptable for GAN image generation, with network packet generation, even some very minor difference in some packet byte values renders the generated traffic unusable (i.e. malformed packets). With network traffic, the generated packet has to adhere to strict network protocols specified in the standardized OSI stack.

Secondly, with images, the position of some pixels located close to each other (i.e. clusters of pixels) can also hold values very similar to one another. CNNs specifically exploit this property with their convolutional, filter and max-pooling operations; e.g. to learn to detect differences in an image's shapes, edges, etc when object (or brightness/colored) boundaries are highlighted by different sets of similar values between pixel clusters. This property was also observed and described in relation to network traffic classification in our earlier work [13]. For traffic generating CNN GANs, it would also be beneficial if clusters of similar packet byte values (and boundaries) exist for the CNN generator model to exploit.

With these observations in mind, to facilitate GAN traffic generations, we propose an alternative scheme for encoding network traffic packets into input CNN matrices.

V. Encoding Network Traffic in CNN GANs

The encoding scheme consists of two steps: (1) conversion of individual packet byte values for representation by sub-ranges of sequential values, and (2) duplicating these converted values and mapping them multiple times into the CNN input square matrix (i.e. create 'clusters' of similar converted packet byte values). Fig. 5b shows an example encoding process.

A. Step 1: Packet Byte Value Conversion

In the conversion step, the hexadecimal digits of each packet byte value are first separated. The intention is for each digit to be represented by a sub-range of serial values; so that whenever the CNN GAN recreates any value from this sub-range, the corresponding digit value is considered re-produced. For base-n numbering, each digit value can be assigned to sub-ranges acquired from dividing the maximum value of a packet byte by n. This approach is similar to the role of the *word2vec* process in linguistic RNNs.

Referring to Fig. 5b, an example conversion for byte value 0x1F is as follows: First, 0x1F is broken down into digits '1' and 'F'. Using base-16 hexadecimals, the digit '1' would be represented by the range 0x10 to 0x1F (i.e. 16_{10} to 31_{10}), and 'F' represented by 0xF0 to 0xFF (i.e. 240_{10} to 255_{10}). When assigning the converted value into the CNN matrix, the midpoint of the sub-range is used, i.e. 0x18 and 0xF8.

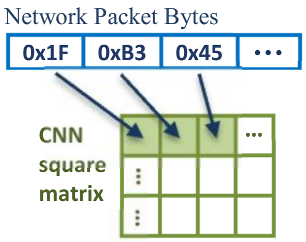

Fig. 5a. One-to-One Mapping

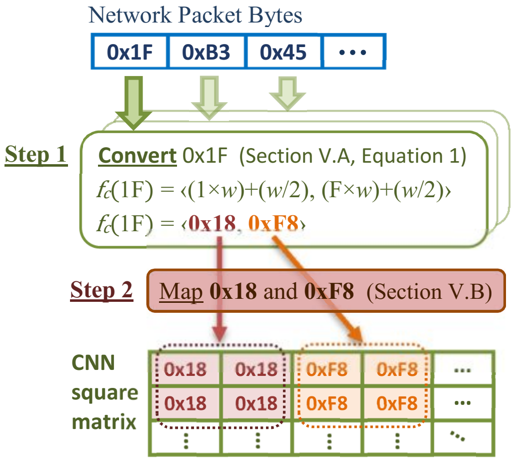

Fig. 5b. Convert & One-to-Multi Map Encoding Process, w=16 d=2

The conversion provides a one-to-many mapping of one value to multiple values, allowing the CNN GAN to recreate values from this small sub-range that corresponds to the desired value. Addressing the first observation in Section IV.A, the goal of converting packet bytes in this way is to cater for the small margin of error in attempting to reproduce similar byte (i.e. pixel) values in CNN GANs.

Formally, the conversion process is expressed as follows. Let $X = x_n \ldots x_1 x_0$ be the string of n digits (e.g. of a packet byte value), let w represent the size of each sub-range of values a digit may be assigned to, and let $Y = \langle y_n, \ldots, y_1, y_0 \rangle$ be the tuple containing the converted string of byte value digits. The conversion process is $Y = f_c(X)$ with

$$f_c(X) = \langle [(x_n \times w) + (w/2)], \cdots, [(x_1 \times w) + (w/2)], [(x_0 \times w) + (w/2)] \rangle \quad (1)$$

From the GAN generator output, in order to extract the actual packet byte value from the converted byte digits, the reverse operation can be performed.

$$f_c^{-1}(Y) = (\lfloor y_n / w \rfloor \times w^n) + \cdots + (\lfloor y_1 / w \rfloor \times w^1) + (\lfloor y_0 / w \rfloor \times w^0) \quad (2)$$

In equation (1), the addition of (w/2) ensures that the converted byte digit value holds a value in the middle of its assigned sub-range. This enables the GAN generator to recreate values targeted at this midpoint value with error margins on either side. For instance, the digit ‘F’ would be converted to value 0xF8 (248_{10}) such that values ±7 of this would still be extracted as ‘F’.

B. *Packet Byte Value One-to-Multi Mapping*

In the second step, the converted byte values are duplicated and mapped multiple times in the input CNN square matrix. The goal of this mapping procedure is to enable the CNN to learn and exploit the positional properties of cluster(s) of similarly converted byte digit values. Fig. 5b depicts one example of how the duplication and mapping is performed. For each converted byte digit, its value is duplicated d number of times across both the rows and columns of the input matrix; i.e. $d \times d$ number of duplications. The duplicated value is simply mapped into the input matrix in sub-matrix blocks of size $d \times d$. The pseudo-code for this operation is outlined in Fig. 6.

Using this encoding scheme, the generated GAN output aims to produce similar sub-matrix blocks of duplicated values. In order to extract the packet byte value using f_c^{-1} (Section V.B, Equation (2)), the mean of these multiple sub-matrix block values are used.

Fig. 5b shows an example of the complete encoding process. For our implementation, the sub-range size was chosen as $w = 16$ in Step 1. This is well-suited for hexadecimal packet byte values. A sub-range of 16 appears more than sufficient for dealing with the error margins of byte values which the GAN attempts to learn and recreate from.

For mapping multiple byte values to the CNN input matrix, we choose to duplicate each value 4 times (d = 2), i.e. twice across a row and a column. The duplicated mapping of each 2×2 sub-matrix block in the input matrix is simplistic yet effective. After implementing Steps 1 and 2 of the encoding scheme, immediately some generated network traffic packets become sufficiently accurate for sending out through the network.

The encoding scheme suffers from a significant drawback involving scalability. Compared to one-to-one mapping, for each packet byte value, multiple entries in the CNN input matrix is required. However, as proof-of-concept for demonstrating traffic generation using CNN GANs, we consider such encoding scheme acceptable for now. Observing preliminary experiments, we suspect reducing the sub-range size w in Step 1 whilst maintaining generation performance is possible. Alternatively, other higher dimensional encoding schemes could also be applied. We leave this as future work.

```
Input
    Number of times $d$ to duplicate value across each row and col.
    Square matrix of size $n$ with $z_{i,j}$ elements at $i^{th}$ row and $j^{th}$ col.
Output
    Square matrix with duplicated and mapped byte digit values

Initialize $i = 0, j = 0$
for every converted byte digit $y_k$ do
    for next sub-matrix $S_{d \times d}$ of $d$ rows and $d$ columns in $M$ do
        Assign $z_{i,j} = y_k$ to all elements of submatrix $S_{d \times d}$
        Increment $i,j$ to the start of next submatrix $S_{d \times d}$
Pad all non-assigned elements $z_{i,j}$ in $M_{n \times n}$ to 0
```

Fig. 6. Packet Byte One-to-Multi Mapping Pseudo Code

VI. Prototype implementation

We prototyped our CNN GAN traffic generator (PAC-GAN) using 28×28 square matrices for encoding both the training data and GAN generated traffic output. For our discriminator, we implement a simple CNN classifier model. It consists of two 2D-convolutional layers with 4×4 filters of stride 2, followed by a fully-connected layer, and a final linear layer classifying the network packet matrix as real or fake.

The generator consists of six layers (Fig. 3). The first two layers are fully-connected layers which take in a noise vector of size 1024 and output a vector of 12544. A reshape layer is then used to form 256 instances of 7×7 square matrices. Two deconvolution layers are then applied to up-sample these matrices to sizes 14×14 and then 32 instances of the desired 28×28 output size. A final convolutional layer is then used to produce the single instance of the generated network packet traffic in encoded matrix form.

Both the discriminator and generator use L2 regularization with weight decay of 2.5×10^{-5}, leaky rectified linear unit (ReLU) as its activation function, Adam optimization with 0.001 learning rate and 0.5 beta1 exponential decay rate, and loss is measured using the Wasserstein loss function with gradient penalty of 1.0. Training of the GAN typically required 9,600 to 28,000 iterations depending on the type (and complexity) of the network traffic generated.

Using the above discriminator and generator models, the complete CNN GAN traffic generator is shown in Fig. 4. The training data was created directly from creating real traffic flows and capturing (and filtering) the relevant raw traffic using TCPDUMP. The network traffic encoder (Section V) is implemented in Python for seamless integration with the GAN which was implemented using TensorFlow [14]. The generated network packet traffic matrices from the GAN are first decoded (Equation 2, Section V.A) to extract the relevant network packet byte field values. These packet byte fields are then packaged into network packets and dispatched out through the network (i.e. Internet).

With the exception of checksums, all packet byte fields are created directly by our GAN. There was no need to cater for unnecessary generation errors if checksum fields such as the IP, TCP or UDP checksums could be computed from the other generated packet byte field values.

VII. Experiments and Results

For validation, we use our CNN GAN to generate three different types of traffic: ICMP Ping, DNS queries and HTTP Get requests. Each of these traffic types share common IP header specifications, but require different ICMP, UDP, TCP, DNS headers and payload formats. Table I shows some example variant destination Ping IPs, and DNS query and HTTP hostnames of our traffic types.

With Ping traffic, typically 84 bytes per packet are generated by the CNN GAN. DNS query produces 60 to 66 bytes depending on the length of the queried hostname, and similarly HTTP Get requires 40 to 83 bytes per packet. With HTTP Get, packets establishing a TCP 3-way handshake are a prerequisite and we reuse generated byte fields to create and send the initial SYN, ACK of SYN-ACK, and finally the HTTP Get request. Fig. 1 showed visual examples of some different traffic type representations in the 28×28 CNN matrices using greyscale values.

Different metrics have been defined to measure the performance of GANs. However, the majority of these are specific to popular GAN application domains such as image generation. For network traffic generation, we measure the performance of our GAN by how successful it generates network traffic that achieves a successful response from the network. We define this *success rate* as the number of packets that are successfully sent divided by the total number of packets generated by our GAN.

A packet is considered successfully generated when it has been dispatched to the Internet and a valid network response is received; for example, a Ping reply, returned DNS hostname queried IP or expected text/content from the HTTP server. With GAN traffic generators, this success rate is considered the true performance measure. A GAN which cannot produce traffic that can be sent through the network is essentially useless for traffic/test generators. We note that none of the surveyed papers in Section II measure such success rate.

Beyond the success rate, we also measure byte error. This is defined as the number of byte field values in a packet that were incorrectly generated (i.e. non-compliant to network standards), averaged over all generated packets. The byte error gives guidance as to how erroneous a packet is compared to a properly formed packet.

Table II shows our results using our CNN GANs to generate each traffic type individually, followed by generating different combinations of traffic. Generally, individual traffic generation achieves higher success rates compared to generating various mix of traffic types. This is not unexpected given the additional complexities and packet variations that the GAN must learn from across the different traffic types. Compared to Ping and DNS traffic, HTTP traffic also achieves lower success rate. Again, this is expected due to the additional complexities of HTTP traffic where 3-way handshake of multi-sequence packets is required. In comparison, Ping or DNS traffic require a single packet generation and send. Despite this, up to 79% success rate for generating HTTP traffic is still considered favorable. In terms of training iterations and duration, generally, the more complex and greater combinations of traffic types, the larger the training steps and time required.

Surprisingly, for DNS traffic generation, we found that injecting some small samples of Ping traffic into its training data improves its success rate (and likely resolved DNS generator model overfitting).

TABLE I. Example Destination Ping IPs, DNS and HTTP hostnames

Ping IPs (organization)	**DNS and HTTP hostnames**
8.8.8.8 (Google)	www.google.com.au
150.101.21.210 (Internode)	www.internode.on.net
58.96.3.34 (Exetel)	www.qut.edu.au
208.67.222.222 (OpenDNS)	www.adelaide.edu.au
9.9.9.9 (Quad9)	www.bing.com
185.228.168.9 (CleanBrowsing)	www.telstra.com.au
1.1.1.1 (Cloudflare)	www.facebook.com
176.103.130.130 (AdGuard)	www.atlassian.net

TABLE II. GENERATIVE ADVERSARIAL NETWORK TRAFFIC GENERATOR RESULTS

	Ping	DNS	HTTP	Ping/DNS	Ping/HTTP	DNS/HTTP	Ping/DNS/HTTP
Success Rate[1]	76% - 90%	95% - 99%	76% - 79%	75% - 86%	71% - 85%	70% - 88%	66% - 87%
Byte Error[2]	24	0.1	0.4	P:36 D:0.6	P:36 H:1.1	D:0.6 H:0.9	P:12 D:0.2 H:1.9
# Training Steps	12800	19200	19200	20000	22000	24000	28000
Training Time	258mins	313mins	313mins	300mins	369mins	377mins	400mins

([1] - Best and worst success rates shown. [2] - P, D and H denotes average number of byte field errors per packet for Ping, DNS and HTTP respectively)

Inspecting the byte field errors for standalone DNS traffic generation, DNS packet generation commonly suffered from a few incorrectly created fields in the IP header. However, by including samples of Ping traffic, which share common IP header fields (e.g. IP header version, etc) between Ping and DNS traffic, the GAN generator was able to learn that these DNS IP header fields should also hold the same values as Ping traffic. It would appear that the GAN has undergone a form of 'reinforcement learning' from inclusion of Ping traffic; specifically, reinforcing the creation of correct byte field values identified across different traffic types.

The above observations lead us to hypothesize that injection of noise or other traffic type variants into the training set could be beneficial to GAN traffic generation (e.g. regularization). The GAN is able to learn and decipher between common or dissimilar byte field values to not only create a greater variety of traffic types, but aid generation of more accurate traffic. The lower success rate for combinations of Ping/DNS/HTTP generations is likely an attempt by the GAN to exploit the above property. We aim to confirm this hypothesis and such GAN characteristics as future work.

For Ping traffic generation, we note that the byte error results are much higher. This is due to greater errors in the generated payload byte values from the GAN. On average, approximately 40% of the payload was not accurately recreated. Despite this, the success rate for sending Ping traffic was still favorable to high. Regardless of payload, a Ping with the correct packet header can still be sent to initiate a reply from the target server. The GAN generator has likely learned this and rather than focus on learning or recreating the payload section accurately, attention is directed to learning and recreating the Ping header byte fields instead.

VIII. DISCUSSION

Our experiments confirm using CNN GANs for network packet traffic generation is a viable alternative to existing traffic generators. Up to 99% success rate can be achieved for individual traffic type generations, whilst at best, generation of different traffic mix produces 88% success. It could be argued that lower success rate indicates the GAN attempting to learn and seek out diverse variants in the types of packet traffic for generation. By doing so, it is not unexpected for some percentage of the generated traffic to be created with non-conforming packets that cannot be dispatched through the network. Whilst this property could make GANs useful for fuzzing network services and applications, the key is to strike an appropriate trade-off between transmission success accuracy and generation diversity to mimic real network traffic profiles and network conditions. However, our focus in this paper was to examine packet sending success rates. Finally, we note that the results of the CNN GAN prototype were achieved without any in-depth refinements of the GAN models themselves. With additional analysis and adjustments to the models' hyper-parameters and CNN architecture, further improvements are likely.

IX. CONCLUSIONS AND FUTURE WORK

This paper proposed and prototyped a GAN model for generating realistic network traffic at the IP packet level. The use of GANs for generating network packets is novel compared to existing flows-based GAN generators [1,10]. A new technique for encoding network data specifically for use in CNN GANs was introduced. Based on generated Ping, DNS, and HTTP traffic, our experimental results prove GANs are a viable alternative for network traffic generators. The work described in this paper forms the basis for further developing GAN based traffic generators at scale. The GAN described in this paper dealt primarily with non-sequential individual network packet generation. In the future, we intend to generate multi serial network packets from greater variety of network traffic types using recurrent neural network GANs.

REFERENCES

[1] M. Rigaki and S. Garcia, "Bringing a GAN to a knife-fight: adapting malware communication to avoid detection," in IEEE Security and Privacy Workshops (SPW), San Francisco, CA, 2018, pp. 70-75.

[2] Z. Lin, Y. Shi and Z. Xue, "IDSGAN: Generative adversarial networks for attack generation against intrusion detection," unpublished, 2018.

[3] M. Nour and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 Data Set)," in Military Communications & Information Systems Conference, 2015.

[4] IXIA BreakingPoint, https://www.ixiacom.com/products/test, Oct 2019.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and V. Bengio, "Generative adversarial nets," in Advances in Neural Information Processing Systems, 2014.

[6] G. Antipov, M. Baccouche and J. Dugelay, "Face aging with conditional generative adversarial networks," in IEEE International Conference on Image Processing (ICIP), Beijing, 2017, pp. 2089-2093.

[7] J. Zhu, T. Park, P. Isola and Al. Efros. "Unpaired image-to-image translation using cycle-consistent adversarial networks", in IEEE International Conference on Computer Vision (ICCV), 2017.

[8] H. Kim, P. Garrido, A. Tewari, W. Xu, J. Thies, M. Niessner, P. Perez, M. Zollhöfer and C. Theobalt, "Deep video portraits," in ACM Transactions on Graphics, 2018, pp. 163:1-163:14.

[9] M. Olof, "Continuous recurrent neural networks with adversarial training," in Constructive Machine-Learning Workshop, 2016.

[10] M. Ring, D. Schlör, D. Landes, A. Hotho, "Flow-based network traffic generation using generative adversarial networks," unpublished, 2018.

[11] C. Yin, Y. Zhu, S. Liu, J. Fei and H. Zhang, "An enhancing framework for botnet detection using generative adversarial networks," in IEEE International Conference on Artificial Intelligence & Big Data, 2018.

[12] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," unpublished, 2017.

[13] K. Millar, A. Cheng, H.G. Chew and C.C. Lim, "Using convolutional neural networks for classifying malicious network traffic," in press, Deep Learning Application for Cyber Security, Book, Springer, 2019.

[14] Tensorflow, https://www.tensorflow.org, Oct 2019.