Generative Adversarial Networks (GANs): A Survey on Network Traffic Generation

Tertsegha J. Anande and Mark S. Leeson

***Abstract*—Generating network traffic flows remains a critical aspect of developing cyber and network security systems. In this survey, we first consider the history of network traffic generation methods and identify the weaknesses of these. We then proceed to introduce more recent approaches based on machine learning (ML) models. In particular, we focus on Generative Adversarial Network (GAN) models, which have developed from their initial form to encompass many variants in today's ML landscape. The use of GANs for generating traffic flows that have appeared in the literature are then presented. For each instance, we present the architecture, training methods, generated results, identified limitations and prospects for further research. We thus demonstrate that GANs are key to future developments in network traffic generation and secure cyber and network systems.**

***Index Terms*—Generative adversarial networks, network traffic, network traffic generation, neural networks.**

I. Introduction

In developing, analyzing, and appraising secured networks and cyber monitoring systems, network traffic flows play a crucial role. Accessing sufficient real network traffic that is appropriate for this purpose has remained a challenge due to existing and increased privacy and security concerns. Publicly available real traffic is largely inconsistent, insufficient, or incomplete thus limiting how much is achieved relying on it. This drives a need to generate synthetic traffic, especially for research and analytical purposes.

The process of generating synthetic traffic involves extracting key characteristics of real network traffic and using these to generate similar network traffic flows. Although a very complex process, research has shown that this is possible. Several generation techniques have been implemented over time with each successive method attaining improved generation levels over previous approaches, albeit with associated limitations. This has recently culminated in the use of Deep Learning models, particularly Generative Adversarial Networks (GANs), which are the particular focus in this survey.

In this study, we consider the range of existing network traffic generation methods, and further make the following key contributions:

- We highlight various evolutionary methods developed and implemented for network traffic generation, and the extent of successful generation achieved using each method.
- We discuss the limitations encountered by the various approaches and show how each successive method overcomes these limitations.
- We show how Deep Learning techniques, particularly GANs, have surpassed previous state-of-the-art methods, delivering enhanced results culminating in packet byte level generation. This is despite their implementation the network traffic generation domain and has suggests further research is needed on the application of GANs in this area.

To this end, the paper is divided into sections as follows. Section I.A surveys earlier traffic generation methods and their shortfalls, Section I.B introduces GANs and highlights seven key models that form the basic architecture of most evolving models. Five GAN models trained for network traffic generation are presented in Section II, and further discussed in detail showing their architecture, training process and results. The survey concludes in Section III with discussion of several observations and conclusions highlighting the prospects for GANs is this area.

A. The Evolution of Traffic Generation

Early traffic generation methods adapted the established Erlang telephony model [1] to attempt to reproduce traffic that was observed. This method used the Poisson distribution for packet arrivals [2], [3] and allowed configuration of a transfer probability matrix that could transfer protocol and port [4]. The method worked well if the network application was simple, but the performance became inconsistent with complex network traffic (particularly the assumptions concerning the packet arrival process and the Poisson distribution) [5], giving rise to Self-Similar models [6]. The ON/OFF traffic self-similar model generated traffic by aggregating multiple sub-streams and each sub-stream cycle (either ON or OFF) was seen to follow the Pareto distribution [7]. Multi Fractal measure, another self-similar model, applied a continuous spectrum to generate non-uniform fractal traffic [8], while the Fractal Gaussian Noise (FGN) model used the Fast Fourier Transform to generate asymptotic self-similar traffic [9]. Self-similar models showed good consistency in results but did not reflect the true characteristics, particularly for packets and network flows [5].

To capture traffic characteristics during generation, various methods were implemented for flow-level and packet-level generation. Harpoon [10], [11] was used to generate representative packet traffic based on empirical distributions (*file size*, *inter-connection times*, and *number of active sessions*) to match *byte*, *packet*, and *flow volumes* of the original data at the Internet Protocol (IP) flow-level, this did, however, exclude *packet loss* and *flow duration*. Flow-level

Manuscript received November 2, 2021; revised February 28, 2022. This work was supported in part by the School of Engineering, University of Warwick.

T. J. Anande and M. S. Leeson are with the University of Warwick, Coventry, CV4 7AL U.K. (e-mail: Tertsegha-Joseph.Anande@warwick.ac.uk).

matrices combined a random number generator of a Poisson distribution, a Pareto distribution for *flow duration*, and a Weibull distribution for *flow size*, to generate traffic that reflected flow attributes such as *occurrence rate*, *ratio*, *duration*, and *size* at the flow-level and not the packet level [12]. Multi threads and interdomain traffic simulation were other methods used for generating network traffic at the flow-level that were also not implemented at the packet-level [5] but several tools were proposed to generate network traffic that level. A packet-level traffic generator was incorporated in the ON/OFF model that enabled it to generate traffic corresponding to the user's configuration with Inter-Departure Time (IDT) and Packet Size (PS) distributions [13]. *Plab* captured Hypertext Transfer Protocol (HTTP) traffic and determined that Inter-Packet Time (IPT) followed a Weibull distribution and PS followed a Lognormal distribution [5]. Swing [14] could extract the characteristics of each HTTP session from real flows such as *source IP* and *destination IP*, PS, *packet arrival* distributions, *request size* and *response size*, although it could only achieve generation for Transmission Control Protocol (TCP). Generated traffic accuracy for packet-level models varied as the distributions that traffic characteristics should follow were not defined nor was the number of characteristics to be considered [5].

A wide range of traffic generation tools have been developed and used for generating traffic tailored to different application protocol traffic. The NS-2 Simulator (subsequently NS-3 and NS-4) provided the function *Tmix* that was used to generate TCP application workload [15]. *Iperf* was used to test the performance of network parameters and report bandwidth, delay jitter and packet loss [5]. *TCP trace replay* used the open-access C/C++ library *libpcap* [17] to capture packet traces, analyse extracted link delays, packet losses, bottleneck bandwidth, packet MTUs and HTTP event timings [16]. *Dummynet*, in [18], was implemented to generate traffic conditions with response times and network delays that mimicked real TCP traces. *Surge* [19] and *Geist* [20] used ON/OFF processes [22] to generate traffic for testing Web Server pressure and realizing HTTP traffic aggregation, whereas *Gismo* [22] applied a similar modelling philosophy to streaming media access. Thus, application protocol-based traffic generation methods have produced network traffic that was close to original network traffic, but only for particular (specified) application protocols and in a more general sense [5].

Learning tools have been developed and used on network traffic but largely for the statistical classification of network traffic flows relying on meta-data payloads. This is unlike protocol-based classification that uses derived heuristics or knowledge of information about IP, port numbers and signature protocols [23], [24]. Bayesian methods (particularly neural networks), modified Association algorithms, Support Vector Machines, Venn Probability Machines, k-Nearest Neighbour and k-means clustering algorithms are among several methods implemented for generating class probability distributions [23], [24]. While most of these methods are vulnerable to overfitting with low spatial, temporal stability and poor data collection even though achieving high classification accuracy [23], neural networks (especially multi-layered or deep neural networks) have shown great promise when implemented for generation and classification tasks [25].

GANs, a category of deep generative neural networks, have shown great potential in their ability to learn intricate data distributions (up to packet-level) and reproduce the same (with subtle variants) within an application domain. This provides the motivation to study and review existing methods implemented for network traffic generation while identifying areas for further research.

II. GANs

The basis of GAN operation is the use of signal backpropagation to train two models, the Generator and the Discriminator, simultaneously pitting them against each other such that both models competitively strive to outdo the other in proving that the generated data is real or fake [26]. They have continued to gain increased attention due to their versatility and dynamic applicability. Several improvements have been made to the initial model of Goodfellow *et al.* (Vanilla GAN) [26], shown generically in Fig. 1 [28]. These include adding class conditions to enhance data generation representations, the incorporation of convolutional layers to enhance better data generation and regeneration, inference network extensions, and adversarial training for enhanced robustness and model training convergence speed [27]. With increasing and evolving applications, GANs have shown highly significant untapped potential in network traffic generation [28], [29] as well as a scalable hybrid architecture which is able to incorporate other model components (both supervised and unsupervised) while providing a network training platform.

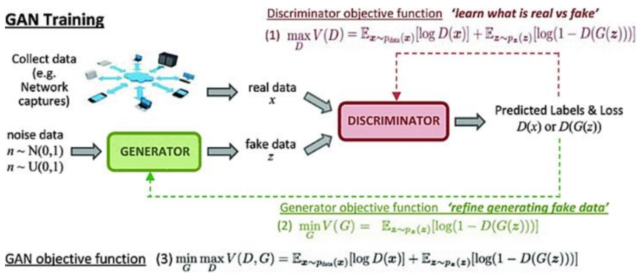

Fig. 1. GAN objective functions and training procedure [[28]].

A. *The Vanilla GAN [26]*

The Vanilla GAN incorporates two models in a corresponding minimax two-player game framework where the Generator (G) models a transform function that strives to fool the Discriminator (D) into mistaking generated data samples for real samples while D models a discriminative function that estimates the probability that the sample data is from generated data or from the true data distribution. The input to G is a low dimensional noise vector ($P_z(z)$), which it transforms into a data vector ($G(z;\ \theta g)$) that is presented to D as a potential data sample. The input to D comprises $G(z)$ and samples of real data ($P_{data}(x)$), and it produces an output that is a single scalar ($D(x;\ \theta d)$) with a score that shows the likelihood of $G(z)$ being from the original data distribution. The minimax objective function is:

$$\min_G \max_D V(D,G) = E_{x\sim P_{data(x)}}[\log D(x)] + E_{z\sim P_z(z)}[\log(1 - D(G(z)))]$$

G targets the minimization of $\log D(x) + \log\left(1 - D\left(G(z)\right)\right)$, aiming to make both $G(z)$ and $P_{data}(x)$ equal to 0.5 to confuse D. At the same time, D strives to correctly classify the fake versus the real data samples by maximizing

$\log D(x) + \log\left(1 - D\left(G(z)\right)\right)$ and forcing $D(x)$ to equal 1. Goodfellow *et al.* [26] optimized the model training using Stochastic Gradient Descent (SDG) and the minimax loss function in equation (1).

B. Conditional GAN (CGAN) [29]

The Vanilla GAN model was extended by Mirza and Osindero [29] by the inclusion of extra (auxiliary) information to condition the model. This extra information, y, which is data from class labels or other modalities is combined as additional input layer and fed as input for G and D. CGAN, modified from (1), is represented in the two-player minimax objective function by:

$$\min_G \max_D V(D,G) = E_{x \sim P_{data(x)}}[\log D(x|y)] + E_{z \sim P_z(z)}\left[\log\left(1 - D\left(G(z|y)\right)\right)\right]$$

here, $P_z(z)$ and y are combined in a joint hidden representation as inputs for G whilst $P_{data}(x)$ and y are presented as explicit inputs for D. CGAN also optimizes model training using the SDG method.

C. Deep Convolutional GAN (DCGAN) [30]

To enhance stable GAN training, Radford *et al.* [30] incorporated Convolutional Neural Network (CNN) components into the GAN architecture by introducing constraints on its topology. To perform its convolutions, a CNN shifts a number of pixels, say n, over the input matrix and n is known as the *stride* [31].

DCGAN introduced fractional-strided convolutions, where a coarser output is connected to denser pixels by interpolation (that can be described as a fractional input stride, producing the name used) [32]. These allowed G to learn its own spatial *upsampling*, and strided convolutions for D to learn *downsampling*. G also uses *batch normalization* and rectified linear unit (*ReLU*) *activation* [33] (at all layers except a hyperbolic tangent function for output) while D applies *batch normalization* and *LeakyReLU* [34] (for all layers). Fully connected hidden layers are also removed from deeper architectures, and models are trained with mini-batch SDG.

D. Wasserstein GAN (WGAN) [35]

The Wasserstein GAN, proposed in by Arjovsky *et al.* [35], made fundamental architectural changes to the Vanilla GAN which included replacing the Discriminator with a Critic (C) that does not have to output the *Sigmoid* function and replaced the Minimax (BCE) loss function with the Wasserstein loss (*W-Loss*) [36] that approximates the distance between $P_{data}(x)$ and *G(z)*, and the amount moved. The objective function, which is modified from the standard GAN in (1), is thus represented by [37]

$$\min_G \max_C \left\{ \mathop{\mathrm{E}}_{x \sim P_r}[C(x)] - \mathop{\mathrm{E}}_{x \sim P_g}\left[C\left(G(\hat{x})\right)\right], \mathop{\mathrm{E}}_{\hat{x} \sim P_{\hat{x}}}[(\|\nabla_{\hat{x}} C(\hat{x})\|_2 - 1)^2] \right\} \tag{3}$$

where P_r is the real distribution, P_g is the generated distribution, and $P_{\hat{x}}$ are uniformly sampled data points ($\hat{x}$) between P_r and P_g. The 1 – Lipschitz continuous condition is included for training to ensure that the *W-Loss* correctly estimates the Earth Mover's Distance (EMD) [38], which measures the distance between two probability distributions over a given region.

1) Wasserstein GAN with Gradient Penalty (WGAN-GP) [39]

Gulrajani *et al.* [39] proposed an alternative method of enforcing the Lipschitz constraint on C, which was based on weight clipping for the WGAN model that resulted in convergence failure or undesired behaviour. This approach, modified from the default WGAN model (3), is represented thus:

$$\min_G \max_C \left\{E[C(x)] - E\left[C\left(G(z)\right)\right] + \lambda E[(\|\nabla C(\hat{x})\|_2 - 1)^2]\right\} \tag{4}$$

WGAN with Gradient Penalty (WGAN-GP) performs random interpolation between real and fake samples during training while penalizing the C's gradient norm with respect to its input. This is represented with a penalty coefficient parameter λ, that scales the gradient penalty.

2) Conditional Wasserstein GAN (CWGAN) [40]

In [40], Fabbri proposed the Conditional WGAN (CWGAN) with improvements to the WGAN and WGAN-GP models incorporating the DCGAN architecture. The model included additional data as input for both G and D or C while training applied the *W-Loss* function and the established Adam optimizer [41].

E. Bidirectional GAN (BiGAN) [42]

Donahue *et al.* [42] incorporated an Encoder (A) into the Vanilla GAN model that enabled it to learn the inverse of G. The proposed model, Bidirectional GAN (BiGAN), learns data mapping inversely for auxiliary supervised discrimination tasks. The objective function, based on (1), is given by:

$$\min_{G,A} \max_D V(D,A,G) = E_{z \sim p(x)}\left[E_{z \sim p(A(\cdot|x))}[\log D(x,z)]\right] + E_{z \sim p(z)}\left[E_{z \sim p(G(\cdot|Z))}[\log(1 - D(x,z))]\right] \tag{5}$$

A is included with G for data mapping to latent representations, while D jointly discriminates in data and latent space where the latent component is either the Encoder output *($A(x)$)* or the Generator input *($G(z)$)*. A is a non-linear parametric function, as are G and D, so is trained using gradient descent; D, G and A are updated simultaneously at each iteration in alternating Stochastic Gradient steps.

III. GANs for Network Traffic Generation

GANs have been extensively applied for data classification and regression, image generation and synthesis, image-to-image translation, text-to-image generation, and enhanced image resolution generation [43]. Dewi *et al.* implemented various GAN architectures for the generation of improved and advanced traffic sign recognition [44], and synthetic prohibitory sign images [45], [46]. When evaluated with real data, results showed high resemblance and recognition accuracy.

These and several other recent works show that GANs have significant untapped potential in their ability to generate high quality network traffic flows, making them highly relevant for network traffic analysis and synthesis [47]. This section discusses models that have been trained to generate network traffic flows, and to what extent generation has been achieved.

A. Model Architectures

Although the use of GANs to generate and analyze network traffic is a relatively new application, there have been several architectures designed and applied with some success. We now summarize these, concentrating mainly on their structure.

1) Imbalanced Traffic Classification (ITCGAN) [48]

This recent development in GANs addresses the problem that typical Internet traffic has very different proportions of traffic from different applications, leading machine learning training to be dominated by the most commonly seen type.

ITCGAN, inspired by the triple-GAN [48] framework, is structured to include three modules as shown in Fig. 2. These are the Traffic Vectorization module that sorts and isolates a vectorized representation of imbalanced traffic features (training set), the Pre-training module that uses Net (a superior network) to train on the vectorized set and stores the pre-trained architecture parameters which are subsequently used as initial states for the Formal Training module. The last of these comprises the GAN framework that includes G, D and a Classifier (Cl).

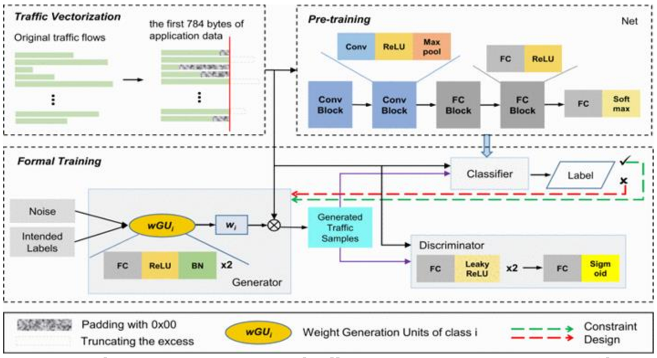

Fig. 2. The ITCGAN Framework, illustrating its constituent parts (the Traffic Vectorization module, the Pre-Training module and the Formal Training module) [48].

G is designed with Weight Generation Units (wGU), which each correspond to a minority class and learn a latent space's conditional mapping g_i to vector $w_i = g_i(z/i)$ of weights N_i [48]. Unlike [26] that trains so as to map a uniform random distribution that is similar to $P_{data}(x)$ to target data, G is trained to learn and synthesize minority samples that fit the original distribution even though this differs from $P_{data}(x)$. This is optimized and represented thus:

$$\min_G V(G) = (V_{i1} - V_{i2} - V_{i3}) \quad (6)$$

where,

$$V_{i1} = \frac{N_n - N_i}{N} E_{G(z|i)\sim p_i^g}[\log(1 - D(G(z|i)))], \quad (7)$$

$$V_{i2} = \frac{N_n - N_i}{N} E_{G(z|i)\sim p_i^g}\left[\log Cl_i\left(G(z|i)\right)\right], \quad (8)$$

$$V_{i3} = \sum_{j\in L\{i\}} \frac{N_n - N_j}{N} E_{G(z|j)\sim p_j^g}[\log(1 - Cl_i(G(z|j)))], \quad (9)$$

p_i^d and p_i^g respectively indicate the real and synthetic conditional probability distributions of class i, and $N_n - N_i$ is the class size. ITCGAN attempts to minimize (7) to fool D, and maximize (8) and (9) to enable Cl predict the synthetic samples as real labels [48].

D is designed similarly to the Vanilla GAN [26] and expressed thus:

$$\max_D V(D) = \sum_{i\in L}(V_{i1} + V_{i4}) \quad (10)$$

where, $$V_{i4} = \frac{N_i}{N} E_{x\sim p_i^d}[\log D(x)], \quad (11)$$

Cl is obtained from the Pre-training module and is represented thus:

$$\max_{Cl} V(Cl) = \sum_{i\in L}(V_{i1} + V_{i2} + V_{i5} + V_{i6}) \quad (12)$$

where, $$V_{i5} = \frac{N_i}{N} E_{x\sim p_i^d}[\log Cl_i(x)], \quad (13)$$

$$V_{i6} = \sum_{j\in L\{i\}} \frac{N_j}{N} E_{x\sim p_i^d}[\log(1 - Cl_i(x))] \quad (14)$$

The GAN architecture incorporates facilitation of correct classification of the imbalanced set while serving as a constraint to guide G during training, and also providing an indication of successful generation thereby eliminating the need to focus on training convergence [48].

2) Packet generation of network traffic GAN (PAC-GAN) [28]

An improvement to the CGAN framework and a hybrid of CNN with the GAN architecture [28], PAC-GAN implements an inverse CNN architecture for G, while D uses the conventional CNN architecture usually employed for supervised classification. Network traffic packets are encoded by G after first converting individual packet byte values for representation by subranges of sequential values and then duplicating the converted values for one-to-multi mapping (see Fig. 3 [28]).

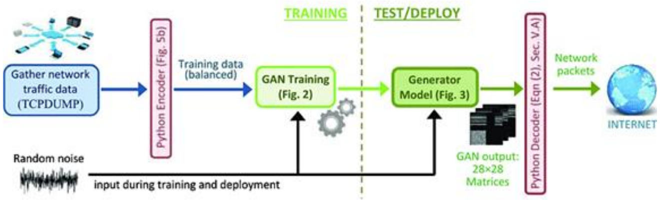

Fig. 3: PAC-GAN framework for network traffic generation and testing/deployment [[28]].

The conversion process is:

$$Y = f_c(X) \quad (15)$$

where $Y = (y_n, \ldots, y_1, y_0)$ is the tuple containing the converted string of byte value digits and $X = (x_n, \ldots, x_1, x_0)$ is the length n string of packet byte value digits. The reverse operation $f_c^{-1}(X)$ is performed on G's output to extract the actual packet byte values. G is further decoupled and deployed for generation of traffic to be transmitted through the Internet. Fig. 4 shows the PAC-GAN architecture [28].

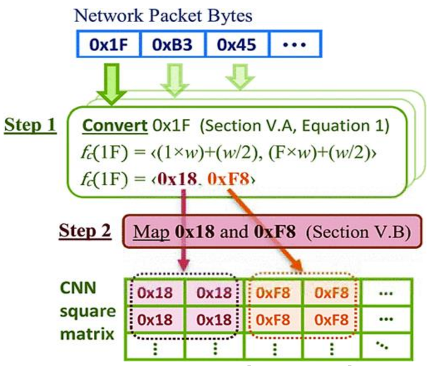

Fig. 4. PAC-GAN Conversion and Map encoding process [28].

3) *Flow-Based network traffic generation GAN [49]*

Ring *et al.* [49] proposed three approaches to generate and transform flow-based traffic into continuous attributes, pre-processed and regenerated into new flow-based network data using WGAN-GP with a Two Time-Scale Update Rule (TTUR). These accepted network attributes as numerical values, created binary attributes from categorical attributes, and used a new similarity measure (*IP2Vec*) to learn vector representations from categorical attributes as shown in Fig. 5 [49]. Flow-based network traffic features comprising *IP addresses*, *Destination Ports* and *Transport Protocols* were extracted and served as input vocabulary with each value representing a one-hot vector, i.e., a group of bits containing only one logical one with all other bits set to logic zero [50]. Input and output layer neurons were each assigned specific values of the vocabulary and these layers (having the same number of neurons) were equal to the vocabulary size. The hidden layer neurons were fewer in number than the input layer neurons. The output layer used a Softmax Classifier that normalized the sum of all output neurons ensuring that it was 1, thus predicting the probability for each value of the vocabulary shown in the same flow as the input value.

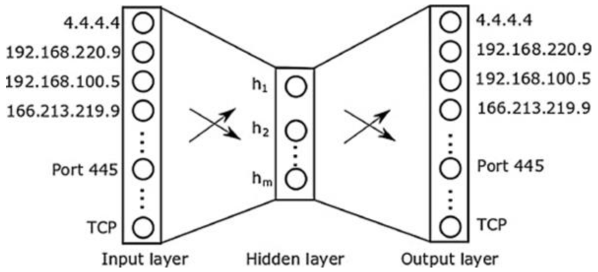

Fig. 5. IP2Vec neural network architecture [49].

4) *Zipper network (ZipNet-GAN) [51]*

ZipNet-GAN, proposed in [51], combined a new deep network, the Zipper Network, and GAN architectures tailored towards Mobile Traffic Super-Resolution (MTSR) to infer narrowly localized fine-grained mobile traffic patterns collected from aggregate coarse data measurements by a limited number of network probes with arbitrary granularity. G is constructed using a deep ZipNet architecture (see Fig. 6 [51]) and comprises *3D Upscaling Blocks* for extracting spatial and temporal features specific to the mobile traffic, *Zipper Convolutional Blocks* as the core and *Convolutional Blocks* that predict the decision after summarizing distilled features received from the core. The *3D upscaling blocks* are input and consist of a *3D deconvolutional* layer, three *3D convolutional* layers, a *batch normalization* layer and a *Leaky ReLU activation* layer. The core, which has 24 *convolutional* layers, a *batch normalization* layer and a *Leaky ReLU activation* layer, takes output from the *3D upscaling blocks*. The convolutional blocks consist of three convolutional layers, a *batch normalization* layer and a *Leaky ReLU* layer with no skip connections. D, which is based on a VGG-net neural network, consists of 6 *Convolutional Blocks* with the final layer employing a *Sigmoid activation* function that constrains the output to a probability range. The *Convolutional Blocks* include a convolutional layer, a *batch normalization* layer and a *Leaky ReLU activation* layer.

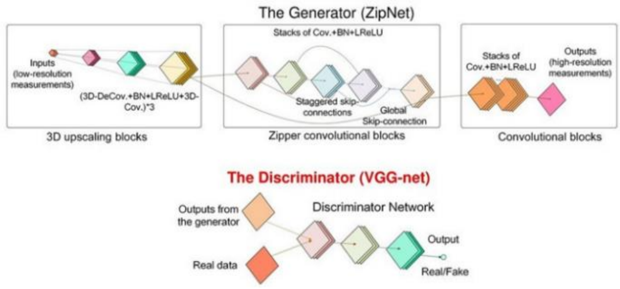

Fig. 6. Architecture of G and D in ZipNet-GAN [48] showing the D upscaling blocks and Convolutional blocks for G′s architecture, and D based on the VGG-net framework [51].

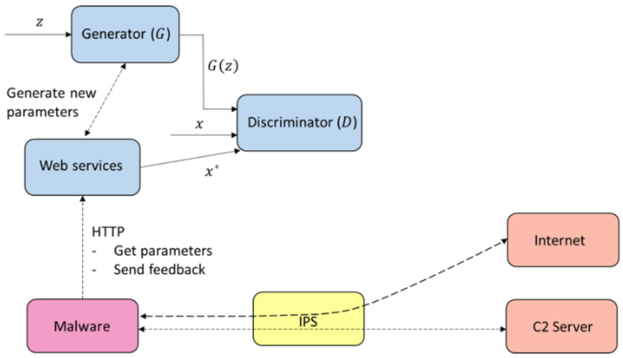

Fig. 7. Framework for facebook chat network traffic GAN model [52].

5) *Facebook chat network traffic GAN [52]*

Rigaki and Garcia [52] proposed a GAN to imitate Facebook chat network traffic and modify the network behavior of real malware by mimicking the traffic of legitimate users while evading detection. D and G for this model were unidirectional and Recurrent Neural Networks (RNNs) modelled using the Long Short-Term Memory (LSTM) architecture. These used a Web Service (HTTP) to communicate with malware by exposing two *API calls.* These were *get_params* (that loads the saved G model, produces new traffic parameters, and sends the same as a JavaScript Object Notation object to malware) and *feedback* (that loads the saved G and D models, adds the parameters of the previous time window to the current dataset based on feedback received and proceeds to another training round). The C2 channel is kept active and operational while HTTP facilitates communication over the channel to the $C2$ Server, and the Intrusion Prevention System (IPS) serves to secure the channel from non-Facebook chat traffic. The model framework is illustrated in Fig. 7 [52].

6) *Packet capture file generator style-based GAN (PcapGAN) [53]*

Proposed to generate and augment *Pcap data* (Packet Capture data for analysis), PcapGAN comprises an Encoder

(E) with four network data parts, G that generates new data for each part of E, and a Decoder that replaces D. Information from *Pcap data* is extracted by E and converted into features such as a *graph* (*IP source → IP destination*), an *image* (*time interval*), and a *layer sequence* structured from network data. *Style* (a vector value) is used to represent relationships between hosts (Server – Client and command and control Server – Botnet). Each data sample generated by E is labelled by the *edge style* (that is, the style value of the relationship between hosts) and used in designing G, which operates in a hybrid structured manner to generate new data which are combined with the reconstructed valid *Pcap file* by the Decoder. Figure 8 shows the PcapGAN architecture [53].

PcapGAN uses a version of (1) modified by the addition of parameters to represent its objective function to produce:

$$\min_{\theta_G} \max_{\theta_D} V(G, D) = \sum_{c=1}^{V} \left\{ E_{v \sim P_{true}(\cdot|v_c)}[\log D(v, v_c; \theta_D)] + E_{v \sim G(\cdot|v_c;\theta_G)}[\log\{1 D(v, v_c; \theta_D)\}] \right\} \quad (16)$$

$P_{true}(v|v_c)$ is the probability of connecting a given node (v_c) to another, where $c = \{1, \ldots, V\}$. $G(v, v_c; \theta_G)$ and $D(v, v_c; \theta_D)$ represent the value function $V(G, D)$ in the Vanilla GAN framework as originally given in (1) [53].

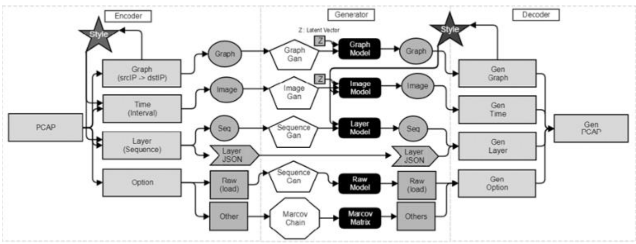

Fig. 8. PcapGAN Framework that implements a hybrid style-based Generator, replaces the Discriminator with a Decoder and incorporates a four-network-data-part Encoder [53].

B. Traffic Generation Results to Date

We now summarize the traffic generation results that have been obtained using various GAN implementations in the literature. In each case, we also summarize the structures and parameters that have been employed in the instances cited.

1) ITCGAN

Unlike previous GAN models, ITCGAN focused on solving the network traffic data imbalance problem. The Pre-training module trained for 300 epochs and used idea of focal loss, which is a method to place increased weight on rare samples. The Formal training module set the batch sizes for all models (G, D and Cl) to 512 and used 40000 training steps, where the ITCGAN parameters were updated twice within a batch for every training step. G and D had fully connected layers and a learning rate of 10^{-3} with a decay of 10^{-4} while Cl employed a learning rate of 3×10^{-4} and decay of 10^{-6}. ITCGAN used a ReLU activation function for hidden layers in both the Pre-training and Formal training modules, with optimization using the Adam optimizer [48].

To evaluate the results, baseline performance was established by training a classifier without addressing imbalance. Then, a range of metrics were considered to compare ITCGAN with established techniques, namely Random Over Sampling (ROS), Adaptive Synthetic Algorithm (ADASYNC), Synthetic Minority Oversampling Technique (SMOTE), SMOTE + Support Vector Machine (SMOTE-SVM), SMOTE + Tomek Links (SMOTE-TL) and a CGAN; the reader is referred to [48] and the references therein for full details of these methods. Here we summarize the global metric results for G-mean (GM) and Mean Area Under Precision-Recall Curve (MAUC-PR) that show the ICTGAN's performance.

ITCGAN outperformed the other methods on GM and MAUC-PR (Table I[48]). The authors also explored the effects of the Pre-Training module, the constraint provided by Cl to G and changing the fully connected G and D layers to convolutional layers. They found that the Pre-Training module enabled faster convergence, the Cl constraint was essential and convolutional layers increased training duration and difficulty.

2) PAC-GAN

This was the first model to successfully generate and manipulate network traffic data (that is, ICMP Pings, DNS queries and HTTP Get Requests) at individual IP packet byte level, which was also deployed to the Internet thereby eliciting responses. Previous GAN traffic generating models only produced traffic at metadata/flow-level. In the network, G consisted of six layers; two fully connected layers, a reshape layer, two deconvolution layers and an output convolutional layer. D had two 2D convolutional layers, a fully connected layer, and an output linear layer for classification. Both D and G used L_2 regularization (with a weight decay value of 2.5×10^{-5}), a *ReLU activation* function, Adam Optimization (with a learning rate of 10^{-4} and beta$_1$ exponential decay of 0.5), and the *W-Loss* function (with a gradient penalty of 1.0).

The success rate in generating individual traffic types is shown in Table II [[28]]. Although this was as high as 99% for some traffic types and 87.7% averaged over all tasks, the model could not achieve the same success rate for generating multi serial network packets from greater variety of network traffic types.

3) Flow-Based network traffic generation GAN

Five training samples were generated by *IP2Vec* (an input and an expected output value for each sample) from each of *Source IP Address*, *Destination IP Address*, *Destination Port* and *Transport Protocol* flows. The neural network was trained with captured flow-based network traffic, taking the value generated by *IP2Vec* as its input and producing the probability for each input vocabulary value, using backpropagation for learning. To reduce the backpropagation training time, *IP2Vec* used Negative Sampling to modify a small percentage of the weights. After training, *IP2Vec* ceased using the neural network and switched to employing the weights of the hidden layers as *m*-dimensional vector representations of the *IP Addresses*. The network attributes were dealt with in three ways to investigate which method produced the most realistic values.

First, network attributes were interpreted as numbers (even though they were in fact categorical). Each octet of *IP addresses* was transformed to continuous attributes within the interval [0, 1]. *Ports* were divided by the highest port number and transformed to continuous attributes while other attributes (*duration, bytes, and packets*) were normalized to the interval [0, 1]. This approach was termed the Numeric-based Improved WGAN (N-WGAN-GP).

TABLE I: ITCGAN OUTPERFORMS ALL METHODS IN THE GLOBAL METRICS EVALUATION WITH REMARKABLE GM AND MAUC-PR IMPROVEMENTS [48].

		Baseline	ROS	ADASYNC	SMOTE	SMOTE-SVM	SMOTE-TL	CGAN	ITCGAN
Global Metric	GM	86.89	90.06	90.82	89.75	90.08	86.24	86.84	**91.19**
	MAUC	91.90	91.00	91.31	93.17	93.08	91.80	91.52	**94.17**

TABLE II: RESULTS FROM PAC-GAN NETWORK TRAFFIC GENERATION [[28]]

	Ping	DNS	HTTP	Ping/DNS	Ping/HTTP	DNS/HTTP	Ping/DNS/HTTP
Success Rate	76%-90%	95%-99%	76%-79%	75% - 86%	71% - 85%	70% - 88%	66% - 88%
Byte Error	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

Second, each octet of an *IP address* was mapped to an 8-bit binary representation producing a 32-bit binary representation. Similarly, *ports* were transformed to 16-bit binary representations, while *bytes* and *packets* were transformed to binary representations limited to a length of 32-bits. The *duration* attribute remained normalized in [0, 1]. The technique was named the Binary-based Improved WGAN (B-WGAN-GP).

In the third approach, the Embedding-based Improved WGAN (E-WGAN-GP) involved the embedding of *IP addresses*, *ports*, *duration*, *bytes*, and *packets* into an *m*-dimensional continuous feature space $\mathbb{R}$. Here, each flow generated 13 training samples consisting of an input and an output value for each. Flows were then mapped to embeddings, which were re-transformed to the original space after generation. *IP2Vec* was used to replace values by their closest generated embeddings.

For training, Ring *et al.* [49] used the opensource unidirectional flow-based network traffic dataset (CIDDS-001) [54], G and D for all three methods (N-WGAN-GP, B_WGAN-GP and E-WGAN-GP) were configured to use feed-forward neural networks and trained for five Epochs. Euclidean distance was used to avoid calculation errors, especially where the probability of generated data is zero.

Results using N-WGAN-GP showed unwanted similarities between categorical values with significant errors (such as similarities in *IP addresses* that should be ranked as dissimilar) making it unsuitable for generating realistic flow-based network traffic. However, as shown in Table III [49], both B-WGAN-GP and E-WGAN-GP successfully generated high-quality flow-based network traffic with E-WGAN-GP achieving better evaluation results (an average of 99.83% over seven heuristic domain knowledge sanity checks) while B-WGAN-GP was able to generate previously unseen values (such as *IP addresses* or *ports*) which was not possible with E-WGAN-GP.

4) *ZipNet-GAN*

Here, the model was trained with Telecom Italia's Big Data Challenge publicly available real-world mobile traffic dataset, the SDG approach, and optimized using the Adam Optimizer for faster convergence, while the loss was calculated based on Euclidean distance. D and G progressed in training synchronously and the learning rate was 10^{-4}. ZipNet-GAN outperformed existing Super Resolution methods for all MTSR instances as shown in Fig. 9 [51] it was evaluated for Peak Signal-to-Noise Ratio (PSNR), Normalised Root Mean Squared Error (NRMSE) and Structural Similarity Index (SSIM) and achieved 40% higher PSNR, smaller NRMSE (up to 78%) and 36.4 times higher SSIM when compared with existing SR techniques.

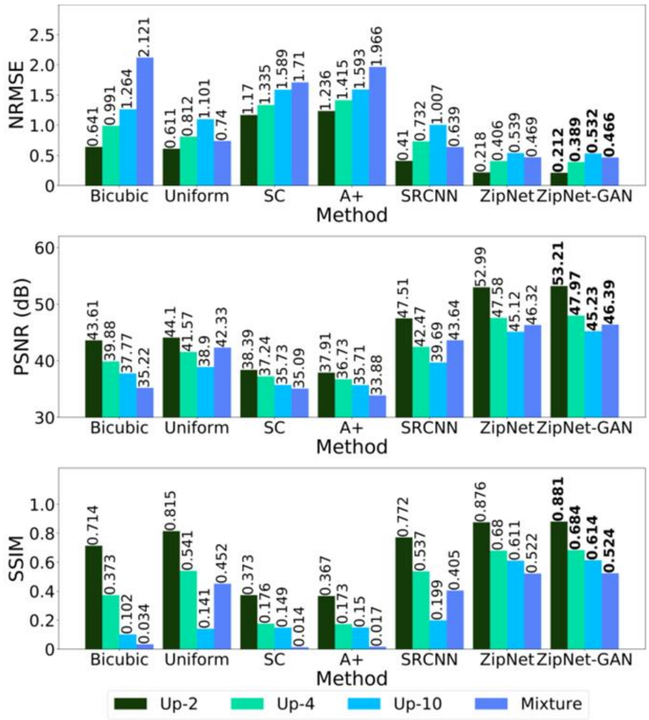

Fig. 9. ZipNet-GAN inference accuracy comparison with existing SR techniques [51].

5) *Facebook chat network traffic GAN*

This GAN was tested by G taking in Facebook chat flow parameters (z), which the GAN used to train for a predefined number of epochs and then sent output to malware via Web Services. Malware traffic remained continuously active in the network and adapted its nature based on detection status and data from additional GAN training. Both D and G had depths of one, 128 hidden units and a sequence length of 6. Model training was via Batch Gradient Descent and the Adam optimizer with a learning rate of 10^{-3}. D trained for three epochs for every one epoch of G. The dataset used for training were network captures (text, images, links, and documents) of Facebook chat between two users over 24 hours, converted to time series (features included *network flow duration*, *total number of bytes in flow*, *calculated inter-flow time from timestamp* of each flow) and used as the variable x.

TABLE III: RESULTS FROM HEURISTIC DOMAIN KNOWLEDGE CHECKS [49]

	BASELINE	N-WGAN-GP	B-WGAN-GP	E-WGAN-GP	*WEEK1*
TEST 1	14.08	96.46	97.88	**99.77**	*100.0*
TEST 2	81.26	0.61	98.90	**99.98**	*100.0*
TEST 3	86.90	95.45	**99.97**	**99.97**	*100.0*
TEST 4	15.08	7.14	**99.90**	99.84	*100.0*
TEST 5	**100.0**	25.79	47.13	99.80	*100.0*
TEST 6	0.07	0.00	40.19	**92.57**	*100.0*
TEST 7	71.26	**100.0**	85.32	99.49	*100.0*

The first objective of the model was to determine if a GAN could mimic the traffic profile of Facebook chat. The Detector was used to determine at the end of each time window if the traffic flow should be logged (fewer than three flows in the threshold), unblocked (due to no decision) or blocked (more than three flows in the threshold). As shown in Fig. 10 [52], increasing the number of epochs eventually led to no blocked flows.

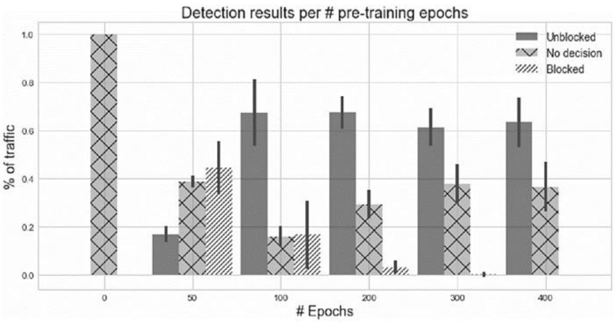

Fig. 10. Facebook GAN traffic detection results [52].

6) PcapGAN

Here, the style-based G took *IP graph* (a sparse matrix in the form $V \times V \times S$ – style vector's *batch size*) as its input and generated a synthetic version of this as network flow data. To generate the *time image*, G performed a mapping of a concatenation of *style vector* (instead of latent space) and the *intermediate vector* (ω). The *layer sequence* was encoded as *sequential data* using the *SeqGAN* model [55] which also customized the model to create the *sequential data* labelled with the *style vector* (for example, the input style vector). *Option data* (a sequence of identical numbers) was augmented to both *sequential data* (using *SeqGAN*) and *labelled sequential data* (using any simple model). The Decoder received the generated *IP graph* and *time image*, the *layer sequence*, and the *option data* and used them to create a *Pcap file* in three steps. Layer sequences were converted into combinations of protocols and then, packet data was created for each protocol using the *option data*. The final step was randomly setting the start time for the first packet of each *edge*, using the time interval information of the time image to set the reception time of the other packets, then chronologically sort the generated packets at each *edge* of the *IP graph* before transforming it into a *Pcap file*.

PcapGAN augmented a cyber-attack dataset (GTISC) [56] with a model pre-trained with a normal dataset (MACCDC 2012) [57], then converted the initial datasets (original GTISK and MACCDC 2012) and the generated (augmented) data into KDD format via the KDD_{99} extractor [58] for applying to an Intrusion Detection Algorithm (IDA). Converted MACCDC data, GTISC data and generated data were labelled data A, data B and data C, respectively. The datasets were experimented on by transforming string data into integers, normalizing them, and then using sklearn algorithms [59] to calculate accuracy, precision, recall and f_i score values (a weighted average of the precision and recall). The results showed consistent accuracy for similarity at 0.5 (showing that the IDA was not able to distinguish between original data and distinguished data). A further test using a classification model was conducted to distinguish between the original data and the generated data and showed that the performance of each IDA improved by 2% to 4% as shown in Fig. 11 [53].

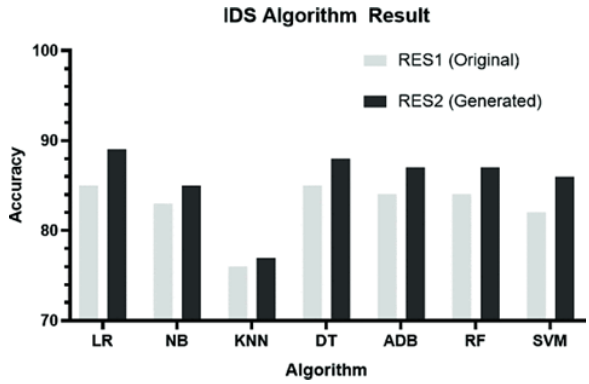

Fig. 11. Result of IDA on classification models RES1 (distinguishing data A and data B) and RES2 is result by IDA model 2 (distinguishing data A and data C). RES2 performance shows that the generated GTISK dataset is valid [53].

IV. DISCUSSION

Despite the progress recorded in other fields, GANs are only just entering the realm of traffic generation. As discussed in the previous sections and shown in Table IV, it can be said that this process has met with successes in some instances.

ZipNet-GAN was only tailored to mobile traffic inference and pattern analysis, and not to generating traffic flows. Although PcapGAN successfully generated high quality cyber data (particularly *pcap* files), this was only for analysis of network flow graph and timestamps. A rate of unblocking actions greater than 63% using the Facebook Chat Network Traffic GAN method showed that GANs could be successfully deployed to mimic Facebook traffic flows.

Unlike the other GAN models reviewed, only limited data are required for training the model, and it was successfully implemented using the stratosphere behavioural IPS in a router to block traffic that was not similar to Facebook chat traffic. However, the framework involved separate deployment of web services to facilitate communication and other types of network traffic were not tested.

The Flow-Based Network Traffic Generation GAN training was only implemented for single flow-based network traffic. However, the model showed sufficient potential to indicate that further studies could achieve training to generate sequences of traffic flows. The PAC-GAN model revealed the potential that GANs have for network traffic flow generation and the possibility of extending research to cover multi-serial network packets for multi-variant traffic flow types of generation especially for large scale traffic and when incorporating RNNs as a hybrid with GANs. Imbalanced traffic was addressed successfully by ITCGAN to emphasize the true potential of GANs for realistic network traffic generation.

We would thus contend that even though network traffic generation using GANs has achieved mixed and varying success levels as shown in Table IV, further research, improvements on the model architectures and training can produce results exceeding the successes recorded to date.

TABLE IV: A COMPARATIVE ANALYSIS OF THE VARIOUS GAN MODELS REVIEWED IN THIS SURVEY SHOWING THE TYPE OF TRAFFIC GENERATED, EXTENT AND LIMITATION OF GENERATION, AND POSSIBLE IMPROVEMENT

	ITCGAN	**PAC-GAN**	**PcapGAN**	**Facebook Chat Network Traffic GAN**	**Flow-Based Network Traffic Generation GAN**	**ZipNet-GAN**
Flow Parameters used during generation	Application layer traffic flows containing imbalanced data with only the first 784 bytes including IP addresses, Port numbers and transport layer protocol	Traffic flows at IP Packet byte level including ICMP, Pings, DNS queries and HTTP Get Requests.	Source IP to Destination IP graph, Time Interval image, and Layer Structure sequence, all extracted from *pcap* data.	Meta-data statistics of network captures converted in Time-Series features.	Meta-data statistics of IP addresses, Port numbers, flow duration, number of bytes and packets sent and received.	Mobile traffic super resolution graph patterns.
Extent of generation	Individual metrics showed average Precision and Recall scores of 93.84 and 91.47 respectively, and Global metrics of 91.19 (GM) and 94.17 (MAUC-PR	Average of 87.7% for all tasks, and up to 99% for DNS.	Generated *pcap* file was analysed by Wireshark, generated network flow graph data was similar to the real data, likewise the patterns in the Time Interval data	Determined by rate of blocked and unblocked actions; up to 63.4% unblocked actions and 0% blocked actions.	Average of 99.83% for E-WGAN-GP, and 81.33% for B-WGAN-GP.	78% NRMSE and 36.4 times higher SSIM
Limitation	Small noise dimension worsens generation diversity while large dimension increases model calculation thus affecting performance and convergence speed.	Was not able to successfully generate multi-serial Network Packets.	Limited to generating flow graph and for timestamp analysis.	Only targeted at mimicking Facebook chat traffic and not tested for other types of network traffic.	Generation achieved only at flow-level meta-data statistics and limited to single flow-based traffic.	Focused on network infrastructure and civil applications.
Possible Improvement	Can be improved to eliminate the Pre-Training module and G 's conditioning without compromising performance level.	Can be improved to generate multi-serial network packets for multi-variant traffic flow types.	Improve the model to be able to quantitatively evaluate the generated *pcap* packet data for total accuracy assessment	Improvement can only be tailored towards blocking and unblocking mechanism which is not relevant to our survey	Can be improved to generate sequence of traffic flows though still at the meta-data level.	Could be modified to extend applicability.

V. CONCLUSIONS

Network traffic generation methods, such as Poisson models, only worked well for simple network applications but were inconsistent with complex network traffic flows. Generation models utilizing self-similar traffic solved the consistency issues associated with Poisson models but were not able to reflect the true characteristics of network flows. Methods used to generate traffic based on characteristic analysis such as Harpoon, flow-level matrix, Multi thread simulation and interdomain traffic simulation were only able to generate traffic at the flow-level. This gave rise to *Plab* and *Swing* that achieved packet-level generation but could not define traffic characteristics according to the distributions that they should follow nor to the number of characteristics to be considered.

Application protocol-based traffic generation models were successfully implemented to generate and simulate network traffic that resembled the original network traffic. This was a significant achievement compared to previous generation levels, even though they could only produce traffic for particular application protocols. Efforts to produce more realistic synthetic traffic flows have led to the employment of GANs.

ITCGAN, PAC-GAN, Flow-based traffic generation GAN, Facebook Chat GAN, ZipNet GAN and PcapGAN are among the GAN models that have been used to generate traffic flows. ZipNet GAN, PcapGAN and Facebook Chat GAN have been implemented for different purposes. These are, respectively, inferring and analysing traffic patterns; generating Pcap files, and network flow graph and timestamp analysis; mimicking traffic flow capture. The flow-based traffic generation GAN achieved metadata level traffic generation for single flows only. Nevertheless, PAC-GAN successfully generated network traffic flows at the packet byte level thereby showing that GANs can generate traffic flows beyond the flow-based level. Further research is recommended into the generation of a variety of traffic flows at the packet byte level, as well as sequences of traffic flows. ITCGAN further introduced a new direction to show the ability of GANs to address the common data imbalance problem in network traffic flows while generating high quality network traffic data. Thus, when compared with previous methods, it is evident that GANs have exceeded existing state-of-the-art in network traffic flow generation hence inspiring further research in this area.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

T. J. A. conducted the literature search and drafted the paper; M. S. L. added material and edited the work to produce the final version; both authors approved the final version.

REFERENCES

[1] V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communications Magazine*, vol. 32, pp. 70-81, Mar. 1994.

[2] T. Bonald, "The Erlang model with non-poisson call arrivals," *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, pp. 276−286, Jun. 2006.

[3] F. Gebali, *Analysis of Computer and Communication Networks*, 1st ed. New York, NY: Springer, 2008, Ch. 11, pp. 383-428.

[4] A. Shawky, H. Bergheim, O. Ragnarsson, A. Wranty, and J. M. Pedersen, "Characterization and modelling of network traffic," in *Proc. International Computer Engineering Conference (ICENCO)*, 2010, pp. 72-76.

[5] J. Zhang, J. Tang, X. Zhang, W. Ouyang and D. Wang, "A survey of network traffic generation," in *Proc. Third International Conference on Cyberspace Technology (CCT 2015)*, 2015, pp. 1-6.

[6] K. Park and W. Willinger, "Self-Similar network traffic: An overview," in *Self-Similar Network Traffic and Performance Evaluation*, K. Park and W. Willinger Eds. Chichester, U. K.: Wiley, 2000, ch. 1, pp. 1-37.

[7] W. Willinger, M. S. Taqqu, R. M. Sherman, and D. V. Wilson, "Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 71-86, Feb. 1997.

[8] S. Wang and Z. Qiu, "A novel multifractal model of MPEG-4 video traffic," in *Proc. IEEE International Symposium on Communications and Information Technology, 2005 (ISCIT 2005), 2005*, pp. 97-100.

[9] V. Paxson, "Fast, approximate synthesis of fractional gaussian noise for generating self-similar network traffic," *ACM SIGCOMM Computer Communication Review*, vol. 27, pp. 5 -18, Oct. 1997.

[10] J. Sommers and P. Barford, "Self-configuring network traffic generation," in *Proc. Fourth ACM SIGCOMM Conference on Internet Measurement*, 2004, pp. 68 – 81.

[11] J. Sommers R. Bowden, B. Eriksson, P. Barford, M. Roughan and N. Duffield, "Efficient network-wide flow record generation," in *Proc. IEEE INFOCOM*, 2011, pp. 2363 – 2371.

[12] Y. Han, "Flow-level traffic matrix generation for various data center networks" in *Proc. IEEE Network Operations and Management Symposium (NOMS)*, 2014, pp. 1-6.

[13] A. Hafsaoui, N. Nikaein and L. Wang, "OpenAirInterface Traffic Generator (OTG): A realistic traffic generation tool for emerging application scenarios," in *Proc. IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012, pp. 492 – 494.

[14] K. V. Vishwanath and A. Vahdat, "Swing: Realistic and responsive network traffic generation," *IEEE/ACM Transactions on Networking*, vol. 17, pp. 712 – 725, Jun. 2009.

[15] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay and F. Donelson, "Tmix: A tool for generating realistic TCP application workloads in ns-2," *ACM SIGCOMM Computer Communication Review*, vol. 36, pp. 65 – 76, Jul. 2006.

[16] Y. Cheng U. Hölzle, N. Cardwell, S. Savage and G. M. Voelker, "Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying," in *Proc. USENIC Annual Technical Conference,* 2004, pp. 87–98.

[17] TCPDUMP. [Online]. Available: https://www.tcpdump.org/

[18] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, pp. 12 – 20, Apr. 2010.

[19] P. Barford and M. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proc. Joint International Conference on Measurement and Modelling of Computer Systems*, 1998, pp. 151 – 160.

[20] K. Kant, V. Tewari and R. Iyer, "Geist: A generator for ecommerce & internet server traffic," in *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, 2001, pp. 49 – 56.

[21] J. Kolbusz, S. Paszczyński, and B. M. Wilamowski, "Network traffic model for industrial environment," *IEEE Transactions on Industrial Informatics*, vol. 2, pp. 213- 220, Nov. 2006.

[22] S. Jin and A. Bestavros, "GISMO: Generator of streaming media objects and workloads" *Performance Evaluation Review*, vol. 29, pp. 2 – 10, Dec. 2001.

[23] M. K. J. Ang, E. Valla, N. S. Neggatu and A. W. Moore, "Network traffic classification via neural networks," University of Cambridge Computer Laboratory Technical Report, UCAM-CL-TR-912, 2017.

[24] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys and Tutorials*, vol. 21, pp. 1988-2014, Apr. 2018.

[25] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436-444, May 2015.

[26] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, pp. 139-144, Nov. 2020.

[27] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, pp. 53-65, Jan. 2018.

[28] A. Cheng, "PAC-GAN: Packet generation of network traffic using generative adversarial networks," in *Proc. IEEE Annual Information Technology, Electronics and Mobile Communication Conference*, 2019, pp. 728-734.

[29] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv Preprint*, arXiv:1411.1784v1, 2014.

[30] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv Preprint*, arXiv:1511.06434v2, 2016.

[31] A. Ajit, K. Acharya, and A. Samanta, "A review of convolutional neural networks," in *Proc. International Conference on Emerging Trends in Information Technology and Engineering*, 2020, pp. 1-5.

[32] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431-3440.

[33] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th International Conference on Machine Learning*, 2010, pp. 807–814.

[34] L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. 30th International Conference on Machine Learning*, 2013, vol. 1, pp. 3-9.

[35] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein Generative Adversarial Networks," in *Proc. 34th International Conference on Machine Learning*, 2017, pp. 214-223.

[36] C. Frogner, C. Zhang, H. Mobahi, M. Araya-Polo, and T. A. Poggio, "Learning with a Wasserstein loss," in *Proc. 27th Conference on Advances in Neural Information Processing Systems*, 2014, pp. 2053–2061.

[37] M. Hong and Y. Choe, "Wasserstein generative adversarial network based de-blurring using perceptual similarity," *Applied Sciences*, vol. 9, art. 2358, Jun. 2019.

[38] Y. Rubner, C. Tomasi, and L. J. Guibas, "A metric for distributions with applications to image databases," in *Proc. 6th IEEE International Conference on Computer Vision*, 1998, pp. 59-66.

[39] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved Training of Wasserstein GANs," in *Proc. 30th Conference on Advances in Neural Information Processing Systems*, 2017, pp. 5767-5777.

[40] C. Fabbri. Conditional wasserstein generative adversarial networks. *Github*. [Online]. Available: https://cameronfabbri.github.io/papers/conditionalWGAN.pdf

[41] D. P. Kingma and L. J. Ba, "Adam: A method for stochastic optimization," presented at the International Conference on Learning Representations (ICLR), San Diego, NM, May 7-9, 2015.

[42] J. Donahue, P. Krähenbühl and T. Darrell, "Adversarial Feature Learning," *arXiv Preprint*, arXiv:1605.09782v7, 2017.

[43] S H. Alqahtani, M. Kavakli-Thorne, and G. Kumar, "Applications of Generative Adversarial Networks (GANs): An Updated Review," *Archives of Computational Methods in Engineering*, vol. 28, pp. 525–552, Mar. 2021.

[44] C. Dewi, R.-C. Chen, Y.-T. Liu, X. Jiang, and K. D. Hartomo, "Yolo V4 for advanced traffic sign recognition with synthetic training data generated by various GAN," *IEEE Access*, vol. 9, pp. 97228-97242, Jul. 2021.

[45] C. Dewi, R.-C. Chen, Y.-T. Liu, and H. Yu, "Various generative adversarial networks model for synthetic prohibitory sign image generation," *Applied Sciences*, vol. 11, art. 2913, Apr. 2021.

[46] C. Dewi, R.-C. Chen, Y.-T. Liu, and S.–K. Tai, "Synthetic Data generation using DCGAN for improved traffic sign recognition," *Neural Computing and Applications*, vol. 33, pp. 1–15, Apr. 2021.

[47] M. Shahid, G. Blanc, H. Jmila, Z. Zhang, and H. Debar, "Generative Deep Learning for Internet of Things Network Traffic Generation," in *Proc. 25th IEEE Pacific Rim International Symposium on Dependable Computing*, 2020, pp. 70-79.

[48] Y. Guo, G. Xiong, Z. Li, J. Shi, M. Cui, and G. Gou, "Combating imbalance in network traffic classification using GAN based Oversampling," in *Proc. 2021 IFIP Networking Conference (IFIP Networking)*, 2021, pp. 1-9.

[49] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Computers and Security*, vol. 82, pp 156 – 172, May 2019.
[50] S. L. Harris and D. M. Harris, *Digital Design and Computer Architecture, ARM® Edition*, Waltham, MA: Morgan Kaufmann, 2016, pp. 129-130.
[51] C. Zhang, X. Ouyang, and P. Patras, "ZipNet-GAN: Inferring fine-grained mobile traffic patterns via a generative adversarial neural network," in *Proc. 13th International Conference on Emerging Networking Experiments and Technologies*, 2017, pp. 363–375.
[52] M. Rigaki and S. Garcia, "Bringing a GAN to a Knife-Fight: Adapting malware communication to avoid detection," in *Proc. IEEE Security and Privacy Workshops*, 2018, pp. 70 - 75.
[53] B. Dowoo, Y. Jung and C. Choi, "PcapGAN: Packet capture file generator by style-based generative adversarial networks," in *Proc. 8th IEEE International Conference on Machine Learning and Applications,* 2019, pp. 1149 – 1154.
[54] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, A. Hotho, "Flow-based benchmark data sets for intrusion detection," in *Proc. European Conf. on Cyber Warfare and Security (ECCWS),* 2017, pp. 361-369.
[55] L. Yu, W. Zhang, J. Wang and Y. Yu. "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient," in *Proc. 31st AAAI Conference on Artificial Intelligence*, 2017, pp. 2852-2858.
[56] G. Severi, T. Leek and B. Dolan-Gavitt, "Malrec: Compact full-trace malware recording for retrospective deep analysis," *Lecture Notes in Computer Science*, vol. 10885, pp. 3–23, Jun. 2018.
[57] Mid-Atlantic Collegiate Cyber Defense Competition. [Online] Available: https://maccdc.org/
[58] Third International Knowledge Discovery and Data Mining Tools Competition dataset. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
[59] scikit-learn. [Online]. Available: https://scikit-learn.org/stable/.

Tertsegha J. Anande was born in Bauchi, Nigeria on November 4. He obtained his undergraduate degree in information and communications technology at Federal University of Technology, Yola, Adamawa State, Nigeria in 2010, and a master of science degree in communication systems from Swansea University, Swansea, Wales, United Kingdom in 2014.

He is currently a PhD student researching the application of deep learning methods on network traffic against cyber and advanced persistent threats at the University of Warwick, Coventry, United Kingdom. He also works as a senior graduate teaching assistant in the Computer Science Department at the University of Warwick. He is seconded from his role as a lecturer at the Federal University of Agriculture, Makurdi, Nigeria, and previously worked as an executive director at Nobel Heights Global Services Limited, Abuja, Nigeria. He has several published works in the areas of cyber security, computer simulation and data mining.

Mr. Anande's research interests are deep learning, generative adversarial networks (GANs), network systems and security, intrusion detection systems (IDS), advanced and applied computing, cyber systems and threats, and advanced persistent threats (APTs).

Mark S. Leeson was born in Rugby, England, and received the degrees of BSc and BEng with first class honors in electrical and electronic engineering from the University of Nottingham, UK, in 1986. He then obtained a PhD in Engineering from the University of Cambridge, UK, in 1990.

He is a reader in the School of Engineering at the University of Warwick, UK, which he joined in 2000. From 1990-92 he worked for NatWest Bank in London then held academic posts at the University of East London from 1992-94, at Manchester Metropolitan University from 1994-1998 and at Brunel University from 1998-2000. His major research interests are optical communication systems and machine learning, with over 300 published works.

Dr Leeson has been on the supervisory team of over 40 successful research students. He is a senior member of the IEEE, and a fellow of both the UK Institute of Physics and the UK Higher Education Academy. He is a previous associate editor of IEEE Communications Letters and the IEEE Communications Magazine. He is a regular reviewer for many academic journals and has also examined over 40 PhDs to date.