Controllable Network Data Balancing with GANs

Abstract

The scarcity of network traffic datasets has become a major impediment to recent traffic analysis research. Data collection is often hampered by privacy concerns, leaving researchers with no choice but to capture limited amounts of highly unbalanced network traffic. Furthermore, traffic classes, particularly network attacks, represent the minority making many techniques such as Deep Learning prone to failure. We address this issue by proposing a Generative Adversarial Network for balancing minority classes and generating highly customizable attack traffic. The framework regulates the generation process with conditional input vectors by creating flows that inherit similar characteristics from the original classes while preserving the flexibility to change their properties. We validate the generated samples with four tests. Our results show that the artificially augmented data is indeed similar to the original set and that the customization mechanism aids in the generation of personalized attack samples while remaining close to the original feature distribution.

1 Introduction

Despite the fact that enormous amounts of network data are generated every day as a result of the Internet's pervasiveness in our daily lives, the amount of network traffic that can be used for research purposes is comparatively limited. In reality, the few publicly available datasets for intrusion detection have a number of flaws, including artifacts if the traffic is artificially generated [Khraisat et al., 2019]. Nonetheless, future network anomaly detection research is limited by the availability of large and diverse network traffic datasets. Furthermore, recent cyber-attacks on critical infrastructures, small/large/medium businesses, non-profit organizations, and even entire governments have once again highlighted the vulnerability of network-connected software systems. This confirms that cyber-security has emerged as a critical concern in our interconnected global society. Typically, Intrusion Detection Systems (IDSs) are created to detect such malicious traffic within a network and recently, there has been a trend towards Machine Learning (ML)-based IDSs that rely heavily on large-scale datasets that contain a mixture of network traffic [Pacheco et al., 2018].

In this paper, we develop a framework based on Generative Adversarial Networks (GANs) to generate and augment malicious network traffic in order to make anomaly detection methods more robust. Effectively, we do not only generate similar samples, but also samples with specific characteristics that necessitates the use of a condition-based generation process. To generate synthetic network flows, we use four established GAN architectures: Conditional GAN (CGAN) [Mirza and Osindero, 2014], Wasserstein GAN (WGAN) [Arjovsky et al., 2017], Auxiliary-Classifier GAN (AC-GAN) [Odena et al., 2017], and Wasserstein GAN with Auxiliary Classifier (AC-WGAN). We also experimented with Wasserstein GAN with Gradient Penalty (WGAN-GP) [Gulrajani et al., 2017], but the results with this architecture were inconclusive. All of the preceding architectures support a defined input vector that we call the condition vector for controlling the generation process.

We propose two ways of controlling attack generation. First, as is standard practice, we set conditions on the network attack type. This enables the generation of flows that resemble a specific attack type, such as Distributed Denial of Service (DDoS) attacks. However, this method only allows for coarse control. As a result, the second approach aims at a finer-grained control over the generation

NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications

process by setting conditions on a set of the most important and intuitive features (for example, '`Flow Duration`' or '`Mean Packet Length`') that the generated flow should or should not exhibit. This doesn't only allows for the generation of an attack of a specific type, but also of an attack that should, for example, have a short '`Flow Duration`' and a high '`Mean Packet Length`'. In general, this method allows us to diversify the generation process thus, increasing the robustness of the IDS.

We evaluate the validity of the generated network flows on four dimensions to ensure that the generated network attacks are as realistic as possible; otherwise, their effect could be unfavorable: **(1)** Attack validity by checking if specific feature values are within the allowed ranges (e.g., a valid protocol number, a non-negative packet length etc.), **(2)** Application of an external classifier, **(3)** Statistical tests on the distribution of generated features in order to compare them to the original distributions, and **(4)** Distance measures between generated and real attacks. It is noteworthy that recent similar work on GANs for network traffic use the same approaches [Ring et al., 2019a, Rigaki and Garcia, 2018, Charlier et al., 2019].

In short, we make the following **contributions** in this paper:

- We propose an architecture for balancing network traffic with GANs and compare it to a state-of-the-art method.
- We propose and evaluate two control techniques for augmenting network traffic.
- We propose an approach that allows for fine-grained control over the generation process.

The rest of this paper is organized as follows: In the next Section, we discuss related work in traffic generation, in particular with GANs. In Section 3 we describe the dataset we use for our experiments and the preprocessing steps we conduct. Furthermore, we explain the methodology and configuration of our approach, that allows us to control the generation process. In addition, we present the evaluation benchmark that we conduct to validate our approach. In Section 4 we present the results we obtain and evaluate the proposed framework. Finally, we conclude the paper in Section 5.

2 Background

Recently, ML has been widely applied to a variety of network traffic applications and has emerged as a competitive alternative to the commonly used rule- and pattern-based methods. GANs, in particular, are used in a variety of approaches where the task is to generate new data. In this section, we go over the preliminary foundations for this work in a nutshell. First, we discuss ML approaches for intrusion detection. Then, GAN architectures for network traffic generation are reviewed.

ML for Intrusion Detection. IDSs are a critical component of any network's cyber security arsenal. Their primary task, as the name implies, is to assist in the detection of unauthorized network intrusions [Buczak and Guven, 2015]. In general, there are two types of intrusion detection systems: (1) signature-based IDSs and (2) anomaly-based IDSs [Khraisat et al., 2019]. Signature-based IDSs rely on pattern matching to detect (known) attacks. There are a number of commercial and open-source signature-based IDSs available, the most well-known of which is SNORT [Roesch et al., 1999]. Although signature-based IDSs can provide strong protection against known threats, they have their limitations, most notably their inability to detect unknown attacks or known attacks with evolving patterns. Anomaly-based IDSs, on the other hand, are an appealing alternative because they can overcome the aforementioned limitations. Such methods employ ML techniques to learn a detection model from available network datasets and use it afterwards to detect outliers. As many of those do not rely on signatures, they have the ability to detect zero-day attacks [Khraisat et al., 2019]. They do, however, rely on the availability of large and diverse datasets to learn the detection model.

GANs for Cyber Security. A GAN is a type of neural network architecture that was originally developed for synthetic image generation [Goodfellow et al., 2014]. The quality of images generated with original architectures was low, and they were easily distinguishable from real images [Radford et al., 2015]. In recent years, however, GAN-architectures, such as StyleGAN v2, have become more sophisticated making it increasingly hard to tell if generated samples are real or fake [Karras et al., 2020]. Since their inception, GANs have been applied successfully to numerous other tasks, such as generating video, audio, text, tabular data, and even entire video games [Tulyakov et al., 2018, Donahue et al., 2018, Engel et al., 2019, Nie et al., 2018, Xu et al., 2019, Kim et al., 2020]. The success of GANs has also attracted the attention of the cyber security research community. GANs

were recently also used to generate network traffic on a flow level [Ring et al., 2019a] or on a packet level [Cheng, 2019] and to generate both normal traffic [Shahid et al., 2020] as well as synthetic malicious traffic [Charlier et al., 2019, Lin et al., 2018].

Network Traffic Generation and Data Balancing. The generation of network traffic, on the other hand, has been a long-standing issue due to a shortfall of labeled traffic for research purposes. Many works used techniques other than GANs to solve this problem, including [Vishwanath and Vahdat, 2009, Botta et al., 2007, Wu and Xia, 2016, Ghazanfar et al., 2020, Kuang et al., 2018, Zhang et al., 2015], and even proposed methods to control the generation process [Sommers and Barford, 2004].

Some proposed Deep Learning (DL)-based solutions that aid in balancing minority classes to deal with class imbalance; an issue that is frequently encountered in anomaly detection datasets [Hasibi et al., 2019, Yuan et al., 2020, Jan et al., 2020].

To the best of our knowledge, we are the first to propose fine-grained control over the generation process for network data. Our proposal provides **a)** generated samples with superior quality and **b)** the option to tune the generation process.

3 Methodology

In this section, we describe the methodology used in this paper. First, we provide a brief overview of the CIC-IDS207 dataset [Sharafaldin et al., 2018], including aggregate statistics and a few notable flaws. Then we describe the model architecture, as well as our two conditioning approaches that we use in our subsequent experiments. Finally, we describe our evaluation strategy for data augmentation as well as controlled traffic generation.

3.1 Dataset

The CIC-IDS2017 dataset is a well-known collection of network traffic containing 14 different types of attacks as well as a variety of normal traffic protocols and services extensively used in network traffic analysis [Vinayakumar et al., 2019, Ring et al., 2019b, Khraisat et al., 2019, Meghdouri et al., 2020]. The number of samples and proportion of each class in the dataset are shown in Table 1. The dataset is highly imbalanced: benign activity accounts for 80% of the traffic, the first three attack classes account for 18%, and the rest account for only 2% of all samples in the dataset. This is reflected in the results presented in the following sections, as the GANs perform well on over-represented classes but poorly on under-represented classes, such as *Heartbleed* or *Sql Injection*. Nonetheless, this is unsurprising and discussed in section 4.

Class	Samples	Proportion (%)
BENIGN	*2273097*	*80.30037*
DoS → Hulk	231073	8.16298
Port Scan	158930	5.61443
DDoS	128027	4.52273
DoS → GoldenEye	10293	0.36361
Patator → FTP	7938	0.28042
Patator → SSH	5897	0.20832
DoS → slowloris	5796	0.20475
DoS → Slowhttptest	5499	0.19426
Bot	1966	0.06945
Web Attack → Brute Force	1507	0.05324
Web Attack → XSS	652	0.02303
Infiltration	36	0.00127
Web Attack → Sql Injection	21	0.00074
Heartbleed	11	0.00039

Table 1: Number of samples per class type in CIC-IDS2017

Each sample in the dataset represents a communication flow with 85 statistical features defined in [Sharafaldin et al., 2018]. However, because `Flow ID`, `Source IP`, `Destination IP`, `Source Port`, `Destination Port`, `Protocol`, and `Timestamp` are traffic identifiers rather than statistical features, we drop them during the generation process. If the `Destination Port` or the `Protocol`

are required during analysis after the generation process, they can simply be appended based on the attack class since each class often exhibits the same value for these features.

3.2 GANs for Data Balancing: Proposed Architecture

We use GAN models for both attack network traffic generation and augmentation. We experiment with four established GAN architectures: CGAN, WGAN, AC-GAN and AC-WGAN discussed in Section 1.

Figure 1 illustrates our architecture on a conceptual level. Overall, there are two central components: the generator and the discriminator which are both fully connected neural networks. The generator produces increasingly more realistic traffic flow samples whereas the discriminator must decide whether the given samples are real or fake i.e., come from the same distribution as the original samples. Hence, the generator and discriminator are, in a way, competing against each other. As one component improves the other keeps up, and vice versa. This is the fundamental idea of GANs [Goodfellow et al., 2014]. Among the different architectures discussed above, we keep the same network size and we only alter the optimizers. Before feeding them into the network, we normalize all input features into the range [0,1].

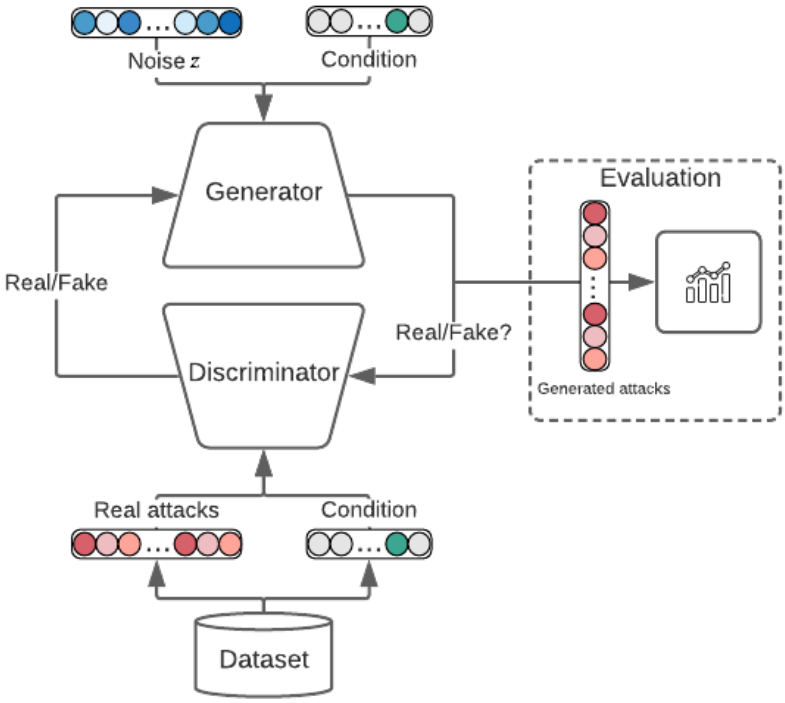

Figure 1: Proposed architecture for network attack generation.

Generator. In order to generate new (attack) samples, the generator is given two inputs: a noise vector **z** concatenated with an additional condition vector (discussed in what follows). The noise vector **z** is a 128-dimensional random vector sampled from a normal distribution ($\mu = 0$, $\sigma^2 = 1$) that is generated anew with each forward pass. The generator is composed of two 256 neuron layers and one 512 neuron layer. We use batch normalization [Ioffe and Szegedy, 2015] between layers to re-scale batches and make the network training more stable, as well as Leaky Relu activations [Xu et al., 2015], except at the final layer, where we use a linear activation.

Discriminator. During training, the discriminator determines whether the attacks generated by the generator are real or fake. To learn to distinguish real samples from fake ones, it also receives real attacks that are sampled from the dataset. Therefore, the discriminator obtains two inputs: the (attack) sample to be generated picked from the original dataset and an additional condition. Depending on the

GAN model used, there are a few differences. For example, with AC-GAN the discriminator doesn't only predict whether the input is real or fake but also what class it belongs to. The discriminator has a similar architecture to the generator, but we use no batch normalization.

Training. To compute the loss at the generator and the two losses (real and fake samples) at the discriminator, we use a sigmoid activation followed by a *Binary Cross Entropy (BCE)* loss. Furthermore, we use the *Adam* optimizer [Kingma and Ba, 2015] with a learning rate of 0.0002 or *RMSprop* [Igel and Hüsken, 2000] in the case of WGAN with the same lr.

3.3 Coarse control: Static Class Condition Vectors

In order to generate a sample of a specific attack, we represent the condition vector shown in in Figure 1 as a static one-hot vector that encodes the attack class. As there are 14 attack classes, the condition vector is of dimension 14. For CGAN and WGAN this condition vector is provided for both the generator and the discriminator. However, for models that additionally employ an auxiliary classifier within the discriminator, only the generated "real" sample is fed into the discriminator, otherwise, it would be simple to predict the right class and the additional cross entropy loss on the attack class would be useless. This setup makes it possible to not only generate attacks, but attacks of a particular type.

3.4 Fine-grained control: Dynamic Feature Condition Vectors

While the static class approach allows to generate attacks of a particular attack type, it only enables coarse control. Therefore, we propose a dynamic condition vector that aims for a more fine-grained control over the generation process, by setting conditions on a subset of eleven features present in our dataset and letting the GAN predict the rest therefore, generating samples using a subset of features that represent a condition. The features selected are shown in Table 2. These features are selected based on their relevancy from the IDS trained on the original data. Instead of constructing a condition vector using the plain feature values, we encode them in such a way that they are easily adjustable during the generation process as shown in the same table.

Feature	**Condition on Value**

`Flow Duration`
`Total Backward Packets`
`Total Forward Packets`
`Mean Packet Length`
`Bytes/s`
`Minimum IAT`
`Maximum IAT`
	Low/Mid/High

`PSH Flags`
`SYN Flags`
`RST Flags`
`ACK Flags`
	Flags/No Flags

Table 2: Dynamic features chosen for the condition vector.

Overall, this results in a 25-dimensional binary condition vector ($3 \times 7 + 4 = 25$ dimensions) for each training sample shown in Figure 2. This approach enables a more fine-grained control over the generation process. For example, we can specify that attack "X" with a short '`Flow Duration`' and high '`Mean Packet Length`' should be generated. However, if the goal is simply to augment a certain predefined class, we can compute the mean flow and then extract the respective dynamic condition vector to be used afterwards. In contrary to the static class condition vector, generating the training set in this case is not trivial. We pick at most 10k random samples from each class then construct the dynamic condition vector for each sample, as explained below. The resulting subset is then used as a training set. This allows the generator to also learn the relation between the output value and the selection technique explained above.

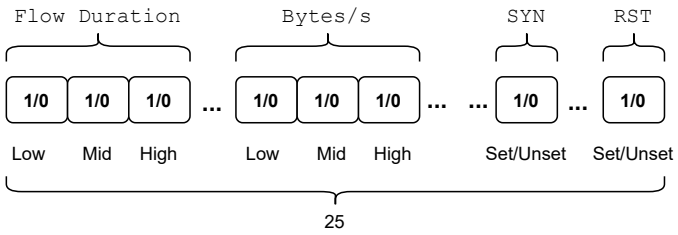

Figure 2: Dynamic Condition Vector.

All three-level features are continuous. In order to convert them into a three level binary vector, each feature value is binned into 3 buckets: low, mid and high. For each feature in the training dataset the 33rd and the 66th quantiles (Q_{33} and Q_{66}) are computed and the encoded vector ($\mathbf{E}_{x,f}$) of a feature f and sample $\mathbf{x}$ is chosen such that:

$$\mathbf{E}_{x,f} = \begin{cases} [1,0,0], (low) & \text{if} \quad \mathbf{x}_f < Q_{33} \\ [0,1,0], (mid) & \text{if} \quad Q_{33} \leq \mathbf{x}_f < Q_{66} \\ [0,0,1], (high) & \text{if} \quad Q_{66} \leq \mathbf{x}_f \end{cases} \tag{1}$$

The flag-features however, are binary and determine whether a flag is set in at least one of the flow packets or not. Therefore, we simply include them in the condition vector as a single binary dimension.

3.5 Evaluation

Often, GANs are used to generate 2D data (i.e. images) which makes the (visual) evaluation and inspection easier. However, when dealing with tabular data the evaluation is not as straightforward and generated samples cannot be visually validated thus, many challenges arise [Borji, 2019]. Therefore, our approach requires additional methods to evaluate the quality of the generated samples. Subsequently, we present and employ three techniques to quantify the quality of generated samples as well as the evaluation method for the coarse control of the generation process.

3.5.1 Attack Validity

One aspect of evaluating the process is ensuring that the generated samples represent genuine attack flows. Unfortunately, GANs are blind to the generated values and their meaning, so we set rules to filter out all samples with inconsistent feature value combinations, such as samples where the `Maximum IAT` is less than the `Minimum IAT`. The rules for each feature that exhibits multiple statistical operators are as follows:

- Maximum X $>$ Minimum X
- Maximum X $>$ Mean X $>$ Minimum X

After cleaning the generated data, we assess its quality using an external Random Forest (RF) classifier trained on the original dataset (excluding normal traffic). The idea behind this approach is that when the generated network flows are fed into an IDS, they should produce high detection ratios. We chose this ML-model because it trains quickly and produces the highest classification scores, as presented by the dataset authors [Sharafaldin et al., 2018]. To avoid over-fitting, we perform a 5-fold cross-validation and report the accuracy-, precision-, recall-, and F1-scores (marco and weighted) in the comparison.

3.5.2 Statistical Test

In addition, we run statistical tests to compare the generated feature distributions to the real feature distributions. The underlying assumption of this evaluation method is that for generated samples to be realistic, their feature distributions should be similar to real ones. As a result, we apply the two-sample t-test (with significance level $\alpha = 5\%$) [Snedecor and Cochran, 1989] to each feature in a batch of generated and real attacks. The test considers a feature to be "real" if its distribution is

similar to the original one and their means are fairly equal. In the best-case scenario, all distributions are not significantly different when compared pairwise, and thus all features are considered real.

3.5.3 Distance Measures

We compute the Euclidean distances between generated samples and the original class centroid. Smaller distances represent samples near the original class mean and vice versa.

3.5.4 Output Control

We also assess how well the generation process can be controlled under dynamic conditions. We accomplish this by varying one feature at a time and observing the effect on the related features at the output. However, because the input condition contains features of various types, we group them into two families: three-level features and binary features.

Three-level features. We manipulate the value of each feature three times, setting it to low (1,0,0), mid (0,1,0), or high (0,0,1) and then generate 1000 samples of each attack class to see if the averaged output over all samples μ meets a given condition by comparing it to the original means μ_{low}, μ_{mid} and μ_{high}. As an example:

- Set the `Mean Packet Length` to a low value (1,0,0),
- generate 1000 DDoS samples,
- compute the average value of the `Mean Packet Length` at the output,
- increment an initially zeroed quantity called the "success score" based on the following conditions:
  - ➔ +1 **if** *low* is selected **and** $\mu < \mu_{mid} < \mu_{high}$,
  - ➔ +1 **if** *mid* is selected **and** $\mu_{low} < \mu < \mu_{high}$,
  - ➔ +1 **if** *high* is selected **and** $\mu_{low} < \mu_{mid} < \mu$,
  - ➔ 0 otherwise.

To make the success score more understandable, we normalize it by the total number of conditions checked: number of attack classes $\times$ number of features $\times$ number of levels ($14 \times 7 \times 3 = 294$) which yields a success rate.

Binary features. For transport layer flags, we again generate 1000 samples per class and flag, then use the accuracy score since the input is a single bit indicating whether the flag is present at least once in the flow or not.

4 Results & Discussion

In this section, we present and discuss the obtained results. The source code as well as the used model weights are available[1] for reproducibility and future use of the framework. During training, we ensure that all architectures are trained for the same number of epochs (150) and that the weights of the last epoch are chosen to generate artificial attack samples.

4.1 Data Augmentation

As discussed in the previous section, we run a variety of tests and generate 10k samples of each class to assess the quality of generated samples when augmenting the training dataset or balancing the minority classes. Furthermore, we compare the scores obtained using the four proposed GAN architectures and Synthetic Minority Oversampling Technique (SMOTE) [Chawla et al., 2002], a widely used de-facto technique for balancing datasets. We show the classification scores obtained at the output of the RF classifier in Table 3. High detection scores represent samples that were correctly classified by the IDS trained on the original set. We observe that AC-GAN has the highest accuracy, whereas CGAN has the highest macro and weighted f1 score with its two condition vectors. This demonstrates that the CGAN architecture can generate distinguishable attack types with high detection scores. SMOTE, on the other hand, performed poorly but is still superior to WGAN and AC-WGAN.

[1] `<hidden-for-blind-review>`

Metric	AC-GAN	AC-WGAN	CGAN	WGAN	Dynamic-CGAN	SMOTE
Accuracy	**0.86±0.02**	0.72±0.03	0.84±0.01	0.55±0.10	0.83±0.01	0.67±0.00
Macro f1	0.19±0.00	0.27±0.02	0.48±0.13	0.15±0.05	**0.52±0.03**	0.38±0.04
Macro precision	0.19±0.00	0.31±0.01	0.45±0.09	0.14±0.04	**0.56±0.08**	0.39±0.06
Macro recall	0.22±0.00	0.30±0.02	**0.57±0.11**	0.18±0.08	0.52±0.06	0.42±0.09
Weighted f1	0.83±0.02	0.71±0.02	**0.85±0.02**	0.53±0.12	0.84±0.01	0.79±0.02
Weighted precision	0.78±0.01	0.71±0.00	**0.88±0.06**	0.57±0.11	0.82±0.02	0.78±0.02
Weighted recall	**0.87±0.01**	0.72±0.01	0.84±0.01	0.56±0.10	0.85±0.02	0.80±0.06

Table 3: Classification performance of generated samples over two trials.

Architecture	Distance
AC-GAN	0.40±0.08
AC-WGAN	0.41±0.08
CGAN	0.35±0.05
WGAN	0.49±0.21
Dynamic-CGAN	0.39±0.06
SMOTE	**0.24±0.10**

Table 4: Average distance measure to the class mean over two trials.

Architecture	Real Features
AC-GAN	19.7±1.5
AC-WGAN	28.1±0.4
CGAN	25.0±4.3
WGAN	25.6±0.1
Dynamic-CGAN	41.8±1.0
SMOTE	**45.7±0.1**

Table 5: Average number of "real" features over two trials.

We show the average distance to the original class mean across all classes in Table 4. Low numbers indicate a shorter distance. Unsurprisingly, SMOTE obtains the lowest (best) score because its generation technique generate samples approximately by computing the mean coordinates between each pair; thus, on average, all generated samples are relatively close to the centroid of a given cluster. However, CGAN obtains the next best scores with its two condition vectors, indicating a high similarity between artificial and real samples.

We show the estimated average number of "real" features computed from the generated sample distributions using the t-test discussed in Section 3.5.2 over all attack classes in Table 5. If all distributions match the real ones, the maximum number of features obtained is 78. We again see that SMOTE obtains the highest score because it does not generate samples arbitrarily but rather respects the original distribution. However, using dynamic conditioning with CGAN, we still achieve a similar score. This demonstrates that, on average, the generated samples have distributions similar to the original ones.

4.2 Inter/extra-polated Traffic

To generate customized samples, or more precisely, inter-/extrapolating samples, we employ the technique described in Section 3.5.4 and evaluate it using the success rate, which represents how accurate the control is (in $\%$). The performance for controlling each feature, level, and transport layer flag is shown in Tables 6, 7 and 8 respectively. In terms of features, we notice that not all features are equally controllable. For example, the total number of packets and the maximum inter-arrival time can be controlled with high precision, but the minimum inter-arrival time is difficult to control. TCP flags, on the other hand, achieve nearly equal scores with the exception of the RST flag, where it was found after a thorough investigation that a low score is caused by the fact that very few training samples with the respective condition (with RST set) exist, resulting in the network lacking good knowledge and being unable to build the relationship between the input condition and the desired output. Finally, we summarize the results in Table 7, which displays the success rate per level. All levels achieve roughly comparable scores, with the exception of the mid level, which has a slightly lower score. We believe this is because the mid level condition has two constraints: the predicted value should be lower than Q_{66} **and** higher than Q_{33}, whereas the other levels have only one constraint, i.e., either larger than Q_{66} or smaller than Q_{66}.

Feature	Success Rate
Flow Bytes/s	59.3%
Flow Duration	85.5%
Flow IAT Max	92.0%
Flow IAT Min	54.2%
Packet Length Mean	91.1%
Total Backward Packets	84.4%
Total Fwd Packets	96.5%

Table 6: Success rate by feature.

Level	Success Rate
Low	84.9%
Mid	74.2%
High	80.8%
Overall	80.0%

Table 7: Success rate by level.

Feature	Success Rate
ACK Flags	79.5%
PSH Flags	87.2%
RST Flags	50.0%
SYN Flags	73.7%

Table 8: Success rate for binary features.

5 Conclusion

In this paper, we use GANs to solve the problem of network dataset augmentation and balancing. We apply four well-known GAN architectures: CGAN, WGAN, AC-GAN, and AC-WGAN, to the CIC-IDS2017 dataset and explore two kinds of conditions for attack flow generation. The first approach uses a binary attack-type condition vector to control the generation process and allows for data balancing with only limited control. By conditioning the values of a set of eleven features, the second approach aims for more fine-grained control over the generation process therefore, allowing for a controlled dataset augmentation. Overall, we show that our approach achieves high accuracy scores across three evaluation methods, i.e., t-test, distance measure and detection accuracy with an external classifier; indicating that the proposed framework can be used to balance minority classes in various network traffic datasets as well as for augmenting a given dataset. Furthermore, we demonstrate how to control the attack generation process in order to create customized network flows by superficially altering a set of intuitive features. When compared to the state of the art, our technique produces better results in terms of the representativeness of the generated samples and demonstrates a better understanding of the distributions and feature values. Future work includes using a similar architecture for per-packet generation in which the framework is supposed to construct the entire sequence of packets of a given flow instead of aggregate statistics.

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

Alessio Botta, Alberto Dainotti, and Antonio Pescapè. Multi-protocol and multi-platform traffic generation and measurement. *INFOCOM 2007 Demo Session*, 2010, 2007.

Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

Jeremy Charlier, Aman Singh, Gaston Ormazabal, Radu State, and Henning Schulzrinne. Syngan: Towards generating synthetic network attacks using gans. *arXiv preprint arXiv:1908.09899*, 2019.

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

Adriel Cheng. Pac-gan: Packet generation of network traffic using generative adversarial networks. In *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0728–0734. IEEE, 2019.

Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208*, 2018.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

Syed Ghazanfar, Faisal Hussain, Atiq Ur Rehman, Ubaid U Fayyaz, Farrukh Shahzad, and Ghalib A Shah. Iot-flock: An open-source framework for iot traffic generation. In *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, pages 1–6. IEEE, 2020.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5769–5779, 2017.

Ramin Hasibi, Matin Shokri, and Mehdi Dehghan. Augmentation scheme for dealing with imbalanced network traffic classification using deep learning. *arXiv preprint arXiv:1901.00204*, 2019.

Christian Igel and Michael Hüsken. Improving the rprop learning algorithm. In *Proceedings of the second international ICSC symposium on neural computation (NC 2000)*, volume 2000, pages 115–121. Citeseer, 2000.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1190–1206. IEEE, 2020.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1):1–22, 2019.

Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1231–1240, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

Xiao-hui Kuang, Jin Li, and Fei Xu. Network traffic generator based on distributed agent for large-scale network emulation environment. In *International Conference on Intelligent Science and Big Data Engineering*, pages 68–79. Springer, 2018.

Zilong Lin, Yong Shi, and Zhi Xue. Idsgan: Generative adversarial networks for attack generation against intrusion detection. *arXiv preprint arXiv:1809.02077*, 2018.

Fares Meghdouri, Maximilian Bachl, and Tanja Zseby. Eagernet: Early predictions of neural networks for computationally efficient intrusion detection. In *2020 4th Cyber Security in Networking Conference (CSNet)*, pages 1–7. IEEE, 2020.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Weili Nie, Nina Narodytska, and Ankit Patel. Relgan: Relational generative adversarial networks for text generation. In *International conference on learning representations*, 2018.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*, pages 2642–2651. PMLR, 2017.

Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys & Tutorials*, 21(2):1988–2014, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Maria Rigaki and Sebastian Garcia. Bringing a gan to a knife-fight: Adapting malware communication to avoid detection. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 70–75. IEEE, 2018.

Markus Ring, Daniel Schlör, Dieter Landes, and Andreas Hotho. Flow-based network traffic generation using generative adversarial networks. *Computers & Security*, 82:156–172, 2019a.

Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019b.

Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

Mustafizur R Shahid, Gregory Blanc, Houda Jmila, Zonghua Zhang, and Hervé Debar. Generative deep learning for internet of things network traffic generation. In *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 70–79. IEEE, 2020.

Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

George W Snedecor and William G Cochran. Statistical methods, eight edition. *Iowa state University press, Ames, Iowa*, 1191, 1989.

Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 68–81, 2004.

Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

Ravi Vinayakumar, Mamoun Alazab, KP Soman, Prabaharan Poornachandran, Ameer Al-Nemrat, and Sitalakshmi Venkatraman. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, 7:41525–41550, 2019.

Kashi Venkatesh Vishwanath and Amin Vahdat. Swing: Realistic and responsive network traffic generation. *IEEE/ACM Transactions on Networking*, 17(3):712–725, 2009.

Jiajing Wu and Yongxiang Xia. Complex-network-inspired design of traffic generation patterns in communication networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(5): 590–594, 2016.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015. URL `http://arxiv.org/abs/1505.00853`.

Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. *arXiv preprint arXiv:1907.00503*, 2019.

Danni Yuan, Kaoru Ota, Mianxiong Dong, Xiaoyan Zhu, Tao Wu, Linjie Zhang, and Jianfeng Ma. Intrusion detection for smart home security based on data augmentation with edge computing. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

Junhui Zhang, Jiqiang Tang, Xu Zhang, Wen Ouyang, and Dongbin Wang. A survey of network traffic generation. 2015.