IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 3 April 2024, accepted 9 May 2024, date of publication 16 May 2024, date of current version 24 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3402069

RESEARCH ARTICLE

An Unsupervised Generative Adversarial Network System to Detect DDoS Attacks in SDN

DANIEL M. BRANDÃO LENT[1], VITOR G. DA SILVA RUFFO[2], LUIZ F. CARVALHO[3], JAIME LLORET[4], (Senior Member, IEEE), JOEL J. P. C. RODRIGUES[5], (Fellow, IEEE), AND MARIO LEMES PROENÇA JR.[2]

[1]Electrical Engineering Department, State University of Londrina, Londrina 86057-970, Brazil
[2]Computer Science Department, State University of Londrina, Londrina 86057-970, Brazil
[3]Department of Computer Engineering, Federal Technology University of Paraná, Apucarana 86036-370, Brazil
[4]Integrated Management Coastal Research Institute, Polytechnic University of Valencia, 46022 Valencia, Spain
[5]Superior School of Technology, Amazonas State University, Manaus 69050-010, Brazil

Corresponding author: Jaime Lloret (jlloret@dcom.upv.es)

This work was supported in part by the National Council for Scientific and Technological Development (CNPq) of Brazil under Grant 306397/2022-6 and Grant 306607/2023-9; in part by the Coordination for the Improvement of Higher Education Personnel Foundation of Brazil; and in part by the Superintendency of Science, Technology and Higher Education (SETI) and the State University of Londrina [Pró-Reitoria de Pesquisa e Pós-Graduação (PROPPG)].

ABSTRACT Network management is a crucial task to maintain modern systems and applications running. Some applications have become vital for society and are expected to have zero downtime. Software-defined networks is a paradigm that collaborates with the scalability, modularity and manageability of systems by centralizing the network's controller. However, this creates a weak point for distributed denial of service attacks if unprepared. This study proposes an anomaly detection system to detect distributed denial of service attacks in software-defined networks using generative adversarial neural networks with gated recurrent units. The proposed system uses unsupervised learning to detect unknown attacks in an interval of 1 second. A mitigation algorithm is also proposed to stop distributed denial-of-service attacks from harming the network's operation. Two datasets were used to validate this model: the first developed by the computer networks study group Orion from the State University of Londrina. The second is a well-known dataset: CIC-DDoS2019, widely used by the anomaly detection community. Besides the gated recurrent units, other types of neurons are also tested in this work, they are: long short-term memory, convolutional and temporal convolutional. The detection module reached an F1-score of 99% in the first dataset and 98% in the second, while the mitigation module could drop 99% of malicious flows in both datasets.

INDEX TERMS Anomaly detection, deep learning, generative adversarial networks, software-defined networks.

I. INTRODUCTION

The Internet has been a fundamental tool for modern society for years. More and more people are dependent on its resources for various daily tasks. Cloud data storage, online banking transactions, navigation, media streaming, and communication are among them. Each of them is expected to work with near zero downtime, with the risk of causing severe injury to society in case of a halt. For this reason, backup, maintenance, and security procedures must be followed to avoid significant issues [1], [2].

This type of network structure tends to grow when new features or modules are developed, leading to more devices, such as servers or switches, connected to the network. The growth of these systems can lead to a more complex setup, laborious maintenance and may even hinder the

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

addition of new elements or features. For this reason, system administrators may opt for a different networking paradigm: Software Defined Networks (SDN) [3].

SDN has as its main characteristic the presence of a centralized controller that manages the networking appliances. The control plane is separated from the data forwarding plane since the switches rely on the controller to build their forwarding table [4]. The instruction forwarding between switches and the controller is made via the southbound interface using an open protocol (e.g., OpenFlow). In contrast, the northbound interface enables the communication between the controller and the high-level network-managing applications providing a top-down view of the whole network. With this setup, applications can be developed with ease, not only to manage resources but also to collect statistics and enable security systems [5].

Despite the advantages of SDN, the presence of a central controller responsible for managing the whole network is a significant vulnerability [6]. The network might collapse if the controller malfunctions or is attacked. For this reason, it is ultimately essential to keep the controller secure from attacks, especially the distributed denial of service (DDoS) type.

Denial of service is the most common type of network attack [7]. Its objective is to overload a vital module or device to make it unable to answer legitimate requests. There are various ways a system can be attacked to have its service denied, from sending enough packets to overload the hardware memory to opening requests in an application to reach maximum concurrent users. It is not trivial to overload a server with requests from a single computer since it could be easily detected and blocked, so this type of attack often is performed in its distributed form, using multiple infected computers to send requests simultaneously. This way, a greater volume of traffic is made while it is more difficult to tell legitimate and malicious apart.

For years anomaly detection systems have been studied to protect machines from this type of attack [8]. Usually, anomaly detection techniques can be classified into two main categories: signature-based and anomaly-based. The former has a collection of attack signatures and uses them to spot known attacks occurrences on the network. At the same time, the latter compares a traffic baseline from the expected behavior of the network to the actual traffic and points out anomalies when there is enough discrepancy. While signature-based systems tend to have better precision with known attacks, they are often unable to detect new, unknown attacks in time. For this reason, anomaly-based systems have been prioritized even with a higher false positive rate.

Anomaly-based systems usually have been developed using machine learning since it is necessary to generalize the regular network traffic. Different techniques have been applied, such as random forest, support vector machines, K-nearest neighbor, and others [9], [10], [11], [12]. However, in recent years, studies have shown how deep learning methods can overcome others' performance due to how well they can extract patterns and use larger volumes of data to improve. Commonly, deep learning is based on using deep neural networks to perform classification and prediction and may be implemented in a variety of ways, such as recurrent, convolutional, or generative adversarial networks.

Generative adversarial networks (GAN) have been used in recent years in anomaly detection, data generation, data reconstruction, classification, and other works. This method uses two competing networks to improve together in their respective tasks: the first is a generator with the objective of creating samples as close as possible to real ones, and the second is a discriminator that has to separate the fake data from the original. With enough training, the generator will be capable of creating nearly flawless data, while the discriminator will be able to detect anomalies from the dataset. In GAN's study field, there are different procedures to train and apply the networks; some of them are described in section II. GAN have emerged as a powerful tool to detect anomalies in computer networks and overcome the inherent class imbalance of the problem [13].

In this work, recurrent neurons were inserted in the discriminator to improve its capabilities to detect anomalies in time series data while forcing the generator to create time-consistent samples. Gated recurrent units are highlighted in this work, since they are an efficient variant of long short-term memory which are used to capture time-variant system dynamics [14]. In addition, to diversify our tests, other types of network neurons were also evaluated, such as convolutional and temporal convolutional.

Considering this study, we present as our main contributions:

- The use of Generative Adversarial Networks for anomaly detection in two datasets;
- An unsupervised network anomaly detection system;
- Compare the proposed method with other deep learning techniques;
- Present a mitigation algorithm to block the detected attacks automatically before major consequences.

The remaining of this work is organized as follows: section II presents fundamental concepts to understand the proposed system; section III presents similar studies that either use GAN or detect anomalies; section IV introduces the proposed system; section V presents the validation datasets and the model's performance in them; section VI explains the computational complexity of the system; and finally section VII presents conclusions.

II. BACKGROUND CONCEPTS AND METHODS

A. GENERATIVE ADVERSARIAL NEURAL NETWORKS

Neural networks have been studied for decades and were proven useful until nowadays, especially in reason of the advancements made in their neurons and architectures. From those improvements, recurrent and convolutional neurons were developed, as well as deep networks and autoencoders [15], [16], [17].

Generative adversarial networks are also a product of this evolution. This paradigm uses competition between two

neural networks to improve their performance [18]. The first, called generator has the purpose of creating entries similar to real data, be they images, the output of a function, or even a molecule [19], [20], [21], [22], [23]. The second is a discriminator, which has to separate the generator's fake data from real data. Both networks must be trained together but are independent once the training is complete.

The learning method of a traditional GAN model occurs in recurring iterations of the following process: first, random values are inserted in the generative network to create fake data samples, which are shuffled between regular samples. Each real sample is labeled with zero, while the counterfeit samples are labeled with one. The discriminator is then trained with this set to correctly classify the samples using an appropriate loss function, such as binary cross-entropy [24]. Next, the generator is trained to mislead the discriminator using the previously generated samples. It does so by relabeling them as zero and propagating the gradient from the discriminator to the generator to update its weights, which will work to maximize the discriminator's error when trying to detect fake samples. In other words, the difference between the discriminator's evaluation of an authentic and a counterfeit sample should decrease [25].

It is noticeable and fundamental that the generator does not receive regular samples directly in its training since this would break the parity of the min-max competition, precluding the learning process of the discriminator. Instead, the generator learns the distribution of the original set by minimizing the differences of classification between the types of data [26], [27]. The generator's input consists of random values, so it is able to produce different results every time. This is necessary since the output of a trained neural network is deterministic, so it will always return the same results given the same conditions and input. The random values should change between epochs during training to avoid overfitting to the same noise values.

The training process can be considered complete when Nash Equilibrium is reached, which means the discriminator is in a state where it has to ''guess'' the label of its input since the original and fake samples are too similar [28]. In this state, the generator also cannot change its samples considering there is no room for improvement. This state has proven challenging to reach due to different problems inherent to the training process [29].

Among the training problems, one the most commonly mentioned in the scientific community is the ''mode collapse''. Due to the necessity of deceiving the discriminator, the generator may start to specialize in a single sample class, ignoring other types [30]. This results in less diverse data produced by the generator, which leads to a bias from the discriminator towards the more frequent class. In this case, a local minimum can be reached, and both gradients may vanish, causing a halt in improvement [31].

This problem has been addressed in different studies by changing either the architecture of GAN and its loss functions or the training method itself. Arjovsky et al. [32] proposed Wasserstein GAN (WGAN) by changing the discriminator's loss function to act as a critic instead of a stratificator. This way, the discriminator evaluates how far the generator is from the original set, and its improvement is to maximize the grade difference between data types, thus resulting in a better gradient for the generator's learning and, consequently, more diverse outputs. The results presented in the study show that the proportion of samples generated is similar to the original set.

GAN networks can also be modified to improve their performance in specific tasks. A typical example is the deep convolutional network, which has kernels to process the data through the network layers. Studies have used this type of network to process and create images [33], [34]. This work used a recurrent neural network to improve the discriminator's ability to detect anomalies in a time series. The generator is forced to create samples not only similar to the original set, but also consistent with the sequence it is generating [35], [36].

B. GATED RECURRENT UNIT NEURAL NETWORK

Recurrent neural networks (RNN) have special neurons prepared to use context data to produce an output. Different types of RNNs receive and use context in their own way. The early generations used to receive the data and their last output, but this would create a greater focus on short-term context without the network having capabilities to store long-term information [37]. Thus, neurons with memory modules were created to store long-term context. Gated Recurrent Unit (GRU) has this context module and uses gates to update and apply this context during predictions [38], [39].

GRU neurons have two internal structures called gates to control the stored information. These structures are conventional neural network layers which calculate a non-linear combination of their input. They are the ''reset gate'', represented by r, and the ''update gate'', represented by z. The first determines how much information should be discarded, while the second regulates how much information will be combined with the input to the output. The gates also have their own set of trainable weights represented by W_r V_r in the reset gate and W_z V_z in the update gate, along with the traditional ones from a neuron represented by W and V. A bias value is also present, represented by b.

In equations 1 to 4, t represents the current iteration, while $t-1$ means the previous iteration; additionally, x represents the input data and h the output of the network at a given moment. For each equation, a brief explanation will be given.

First, the reset gate is calculated in 1. This equation receives the input x, multiplies it by the network weights W and combines with the reset gate's weights V_r multiplied by the previous output h_{t-1} added with the bias b. This equation outputs a value between 0 and 1 due to the sigmoid σ function. This value will be used in equation 2 to regulate how much information from previous iterations should be considered.

In equation 2, $\hbar_t$ is computed and it works as a preliminary output of the network. Its result comes from the combination of the previous iteration multiplied by the reset gate and the network input. Those values are multiplied respectively to V and W and inserted in a *tanh* function.

As mentioned before, the update gate is responsible to regulate how much information from the last iteration should be combined to the currently calculated output $\hbar_t$. To do so, it combines the network input to the previous iteration similarly to the reset gate in equation 3. It is important to highlight that the weights used in each gate are different from each other. The value of z_t is also between 0 and 1.

Finally, the current output is defined by equation 4. It will use the value defined by the update gate to combine the preliminary output to the previous iteration's output. In the equation, z_t multiplies the preliminary output while the previous output is multiplied by $1 - z_t$ [40].

$$r_t = \sigma(W_r x_t + V_r h_{t-1} + b_r) \tag{1}$$
$$\hbar_t = \tanh(W x_t + V(r_t \cdot h_{t-1}) + b_h) \tag{2}$$
$$z_t = \sigma(W_z x_t + V_z h_{t-1} + b_z) \tag{3}$$
$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \hbar_t \tag{4}$$

GRU networks have already been used for anomaly detection in computer network traffic. For example, Tang et al. [41] proposed a system to detect attacks using six raw features. The author compared GRU networks with SVM, DNN, and the traditional RNN in the NSL-KDD dataset. The results showed GRU's superiority over the other methods. Our work has a similar approach, using just a few features extracted from traffic. However, it differs by using GAN to make our system capable of unsupervised learning to detect unknown attacks.

C. SOFTWARE DEFINED NETWORKS AND ANOMALY DETECTION

As described in the introduction, it is not uncommon to have computer networks with enough switches and hosts to hinder adding new features or equipment due to the complexity of configuration and maintenance. Networking devices require setup between equipment, and compatibility is critical to a satisfactory orchestration of traffic data. However, manufacturers rarely provide support for each other's equipment or features, thus creating the necessity of a homogeneous ecosystem, which certainly would not be viable to replace if necessary [42].

Software Defined Networks is a networking paradigm that solves previously presented problems. Its main concept is the centralization of the control plane, which removes the autonomy from network switches that will instead be in constant communication with the controller. An open protocol (e.g., OpenFlow) enables this communication and is not bound to a manufacturer, thus allowing for a heterogeneous network with fewer restrictions [43].

The network controller works as an interface (called northbound interface) for management applications and security ones. The applications work as modules that can be installed at any time and do not require the configuration of switches since only the controller communicates with them [43]. The system proposed in this study works as one of these applications by consulting the controller for flows and sending hostile IP information. In the same way, it would not be required to update the application if a switch was added or replaced. This ease of configuration is one of the inviting aspects of SDN.

However, the strength of SDN is also its main vulnerability: since the controller is fundamental for the network's operation, any disruption may cause instabilities to the system [44]. Disruption may happen in different forms, but the main one addressed in this study is the distributed denial of service attack. It is possible to use the SDN resources to collect network flow data and analyze it to detect anomalies, protecting the controller and other hosts from attacks.

With the traffic data provided by the network controller, it is possible to create a behavior baseline from the regular traffic. Then, an anomaly threshold can be found to separate anomalous behavior from the normal one. Anomaly detection systems that work similarly in this way are classified as ''anomaly-based'' and have been studied due to their capacity to detect unknown threats since using attack data to train it is unnecessary. However, it is not uncommon for this system to present a higher false positive rate seeing that network behavior can have sudden changes due to external events. In addition, the traffic behavior may increase or decrease over time, requiring the baseline to be updated from time to time.

D. DISTRIBUTED DENIAL OF SERVICE ATTACKS

Despite common, distributed denial of service attacks have diverse categories of how they are executed and how they affect their victim. In this section, a brief overview of how other studies describe attacks is presented.

Wabi et al. [45] sunder attacks by which SDN plane they are affecting and describe their point of failure. For example, attacks that affect the data plane exploit the switch's limited memory. When an unknown packet is received, the controller has to be consulted to how it should be forwarded by having information about it sent through the network. While the controller makes its decision, the packet is stored in the switch's memory. With enough packets in a short period of time, a saturation of this buffer may happen, which leads to lost packets.

Another point of failure highlighted by Wabi et al. is the SDN controller. Similarly to the data plane example, the control plane also has limited resources to store forwarding tables, network information, and data for the applications installed. When the controller is overloaded, the whole network may suffer from the attack.

A third point of failure can be highlighted: the southbound interface. It is the communication channel between the data plane and the controller. A DDoS attack could saturate the bandwidth and interrupt the communication between

the switches and the control plane. As a consequence, similarly to an unavailable controller, the whole network could malfunction. Wabi et al. also present other SDN vulnerabilities. However the three presented are the most relevant to this work.

Chaudhary and Mishra [46] spotlight some types of DDoS attacks in their work. The first is UDP flood attack, that targets a server with UDP datagrams to random ports. When the server receives requests for ports that have no bound application, an error packet must be sent back. Enough requests could overwhelm the server and make legitimate ones get declined due to lack of resources.

TCP Syn is another flooding attack that exploits the TCP handshake protocol. When a user tries to setup a connection, it sends a Syn packet to start the handshake, which the server responds with an ACK and waits to receive it back. When a malicious agent sends a Syn, it usually has a spoofed IP, which means that the first ACK will be never be corresponded. The server resources will be consumed and legitimate connections will be refused.

Lastly, an NTP attack is a bandwidth depletion attack. It exploits the protocol to synchronize the clock of machines by making redundant requests. They return a list with previous queries to the server, which is significantly more costly than the query protocol itself. Again, with enough users requesting this list, the bandwidth will be depleted.

All these classes of attack may target a server but can also affect the software defined network that hosts it. For this reason, a combination of anomaly detection techniques with properly configured protocols and firewalls are fundamental for any network's integrity, availability and confidentiality [47], [48].

III. RELATED WORK

This section gleans similar studies about generative adversarial networks. It also contains brief descriptions of other deep learning methods that may be combined with GAN networks with some examples. In chapter V, some of those methods were applied as a comparison to GRU.

A. GAN-SUPPORTED ANOMALY DETECTION

Generative adversarial network is a versatile architecture due to the multiple possibilities of how they can be applied to solve problems. The first three showcased articles are an example of GAN being used as some form of data generation. There are different reasons to do so, at it is useful to address imbalance problems in a dataset and also to better improve other machine learning models by producing adversarial examples of data.

Ding et al. [49] used GAN to create samples to minimize the imbalance of intrusion detection datasets. The study proposes a tabular auxiliary classifier GAN (TACGAN), a more suitable method to generate tabular data. This algorithm is used only to create anomaly samples, while regular entries were undersampled to reduce even more the imbalance. The performance of their method was compared with several other oversampling methods, and some worthy mentions are SMOTE, WGAN, and ACGAN. Three well-known datasets were studied: KDDCUP99, UNSW-NB15, and CIC-IDS2017. The proposed algorithm effectively improved the classification from the deep multilayer perceptron, which is not the best detection method but served as an experiment. The authors propose to use a better classifier combined with the GAN upsample in future works.

Another work that addresses imbalance and data generation is Kumar et al.'s work [50]. The authors use GAN to generate samples from a minority class to improve the performance of a classifier. The method uses an autoencoder to learn the data distribution and create a reduced feature vector to help increase the performance of the tested classifiers. Next, the GAN model is trained to generate data, and the resulting samples feed the classifiers among Random Forest, Decision Tree, Support Vector Machine, and XGBoost. The latter was chosen as the best in performance and was used in the final testing. Three network traffic datasets were used: NSL-KDD, UNSW-NB15, and Bot-IoT. The results are promising and outperform other state-of-the-art systems.

An adversarial DDoS attack detection system was proposed by Mustapha et al. [51]. This work presented a recurrent neural network anomaly detection system, and GAN networks were used to generate adversarial attacks, reducing its detection effectiveness. To do so, an explainable machine learning technique called SHapley Additive exPlanations (SHAP) was applied. This method shows how each feature affects the analyzed model. This way, using both SHAP and GAN to generate adversarial attacks, the recurrent model can be enhanced to detect them. The CIC-DDoS2019 dataset was used to validate the system. The authors addressed its imbalance and used the benign samples from CIC-IDS2017 to even the amount of benign and anomalous flows. Despite both our work and Mustapha's aim to detect DDoS attacks, the GAN modules are completely different, since their GAN is used to improve the IDS by generating DDoS data, while our method uses GAN as the IDS itself. However, there are still development insights that could be used. For example, the authors mention how the generator becomes too powerful in the dispute between networks. While the reason for this to happen is not clear, the solution presented was to train the discriminator with four times more epochs. This prevents the vanishing gradient problem, but does not solve the mode collapse.

The former article used GAN to improve another machine learning model, Kim and Pak [52] did it as well but in a different way: they presented a method to detect intrusion in real time by analyzing packet data using a recurrent neural network. Upon receiving such a small amount of data in the first packets from a session, the model is expected to fail to classify every entry correctly. A GAN system is trained with the misclassified entries to predict when the recurrent network will misclassify that entry. With this, the process is halted until the next packet arrives. The system

was tested in three datasets: ISCX2012, CIC-IDS2017, and CSE2018. In the study, not only the system's detection rate was measured, but also how fast attacks could be blocked. Overall, the system had a better or equal detection rate than the compared methods but had a quicker detection speed, enabling attack mitigation before significant consequences.

B. GAN-ORIENTED ANOMALY DETECTION

The previous works show GAN as a support module, to solve an issue with the dataset or the anomaly detection system. However, it can also be used as the intrusion detection system itself, as the following works suggest. These studies may combine special neurons to the GAN architecture in order to improve its performance.

Wang et al. [53] addressed the deep neural network problem of test-time evasion attacks (TTEA) using GAN. TTEA, also known as adversarial attacks, happens when small changes to the input causes the neural network's decision to change. To do inhibit it, they used class conditional GAN: a type of network that allows the developer to select which class the generator will mimic each time. Although it is common for GAN to be used to address the adversarial attack problem, this work presents a different approach: Both the discriminator and the generator were applied to detect the anomalies. In this case, the generator works as an image reconstructor, and its error influences the system's decision. The datasets MNIST, CIFAR-10, and Tiny-ImageNet-200 were used to test the system. In all datasets, the mode collapse problem was present specially in CIFAR-10 since it has more diverse data, but the discriminator's detection module could still achieve state-of-the-art performance. The authors claim that mode collapse did not impact the model's performance due to how the discriminator was still able to distinguish real from fake, even with the reduced diversity of the generator's sample.

Hoh et al. [54] proposed a study that applies generative adversarial networks to time series anomaly detection. In this work, a conditional convolutional GAN is implemented. Convolutional neurons rely on kernels to extract information from data. They help the network to learn spacial relationships in their input since they were designed to work with computer vision, where relationship between pixels in the same area are far more important than those away from each other. In Hoh's work, the generator also has an autoencoder architecture to reconstruct data from its input, with transposed convolutional layers in the decoder module. An autoencoder will ''compress'' and ''decompress'' information using data patterns learned from data and will generate error based on how well the information could be reconstructed. When the error is too high, the probability of the entry being an anomaly increases since the patterns learned by the network were not accurate. In this system, the discriminator is only used in the training process as the generator's reconstruction error is used as the indicator of anomaly. The framework was tested using a public dataset of malfunctioning industrial machine investigation and inspection and a single stereo sound file of a song. The authors' proposed system could be adapted for processing network traffic data, as it is a time series. In this case, the different classes would be the variations of traffic throughout the day.

A GAN framework is showcased in the work by Li et al. [55]. It also uses an autoencoder as the generator and real data as its input instead of random noise. However, instead of being trained to output a copy of its input data, the generator receives incomplete samples and has to recreate the original ones. The missing features may vary from each input, so the generator is forced to specialize in all features. After the codification, the decoder will then rebuild the original entry from this ''compressed'' state by using previous knowledge to fill the missing parts. This is a great solution for mode collapse since the generator is required to recreate samples from every class, and it is possible to intensify the training in the classes that it has a poor performance. Additionally, a single data sample might be used several times to train the generator by swapping the missing features each time. By using real samples as the generator's input, it can create more realistic samples, thus becoming more representative of the original data distribution. The generated data can be divided into different time windows and combined with real data to feed the discriminator. This way, a single window from the original time series can become multiple fake samples. The tests presented show that the method is effective in anomaly detection.

Another special architecture is presented in Adiban's et al. work [56]. They presented an anomaly detection model that uses multiple generator networks to solve the mode collapse problem. In this study, each generator is trained to create samples from a specific class in order to force each mode to be represented. In addition to detecting anomalies in the data, the discriminator is also trained to specify the generator from which the anomaly was created. The proposed system was tested in two imbalanced datasets: the Industrial Control System Security dataset and the UNSW-NB15 dataset. The metrics accuracy and F1 score were applied to measure the system's performance, with F1 being the most important due to the datasets' imbalance. The results show how the model outperforms state-of-the-art systems and present how the generators can learn to recreate more data modes than other GAN networks.

Outside of network security, but still in the field of time series anomaly detection, is the work of Xu et al. [57]. The proposed system applies long short-term memory networks (LSTM) with a GAN architecture to detect anomalies in time series datasets. LSTM and GRU neurons have the same purpose: to select and store useful context information from the time series. However, LSTM has an extra gate when compared to GRU. This gate is an extra tool to select how context will be passed through the iterations and adds extra processing and trainable parameters. In addition to LSTM-GAN, the extreme gradient boosting (XGBOOST) algorithm is applied to the neural network to extract the most

important features from data. The system was tested using the ball-bearing time series dataset and achieved state-of-the-art performance.

There is another paradigm that can be combined with GAN: the transformers. A transformer is a type of architecture that has been used mainly in natural language processing, but works such as Ma et al. [58] employed it to detect anomalies in time series. This paradigm uses an attention mechanism to define which features and moments are valuable to the network's prediction. The transformer process can be done in parallel for the whole input, thus permitting for long sequences of data to be entered without significant impact to computational complexity or context loss. In this work, a model with similar characteristics was studied to compare with GRU: the temporal convolutional networks.

Temporal convolutional networks employ successive convolutions to extract information from its input. Those convolutions reduce the dimensionality of data while extracting from its spacial organization the temporal aspect. This type of network offers as an advantage over recurrent networks the ability of processing temporal data all at once, without the possibility of losing context over time due to long sequences. This allows for longer receptive fields since the information will not be lost. Yang et al. [59] proposed an intrusion detection system using TCN to detect anomalies in internet of things (IoT) data. Their method uses federated learning to combine training from different parties.

From the selected works presented in this section, it is possible to confirm how GAN has different use cases, from direct anomaly detection to data generation. Relevant work proves its efficiency and how there are various methods of avoiding inherent problems, including mode collapse and vanishing gradient. Since those problems are frequently present in other works, they will be addressed in this study in chapter IV; there we present our unsupervised method that combines GAN with GRU neurons to detect and mitigate anomalies in network traffic along with insights gathered during development. It is also worth noting how novel anomaly detection techniques are still studied and proposed due to the importance of the theme and its room for improvement.

IV. PROPOSED SYSTEM

In this section, the proposed system is described in two modules: detection and mitigation.

A. DETECTION MODULE

In this article, an unsupervised, anomaly-based anomaly detection system is proposed. The system adapts the traditional generative adversarial networks concept to use gated recurrent units. The generator network is trained to mimic normal traffic behavior from the monitored network to deceive the discriminator using latent noise from a uniform distribution, as shown in Fig. 1. It is expected that, after the training process, the discriminator will be able to detect

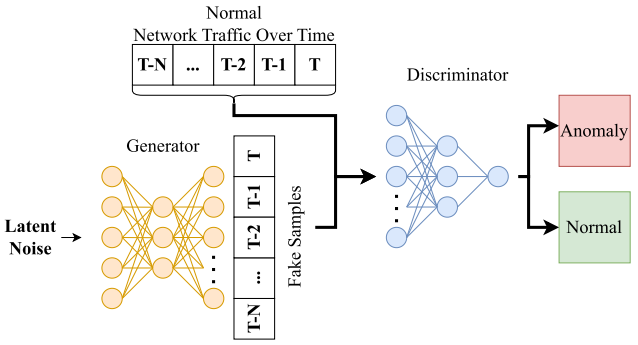


FIGURE 1. System training overview.

anomalies from real network traffic without needing previous contact with it.

This system treats network traffic as a time series. The GRU neurons store context information through iterations to better classify them. For this reason, the discriminator network's input uses the ten last seconds of traffic to evaluate if there is an anomaly in the last second. The GRU neurons run through each second before reaching a final value, which will be received by the next layers of the neural network to make its estimation.

To summarize a whole second of network traffic, some flow dimensions are collected from the SDN controller: total bits, total packets, source and destination IP entropies, and source and destination port entropies. It is essential to extract the entropies of IP and port due to the nature of the neural network's learning. Information such as raw IP addresses tends not to be helpful for a neural network since an address being numerically greater than another has no meaning. Thus, this qualitative data is transformed into a quantitative one: the entropy.

In this work, entropy works as a measure of randomness in a variable. For example, to calculate the destination IP entropy, the frequency of each IP is used. When multiple different IPs are evenly accessed during the analyzed period, entropy increases. Inversely, if fewer IPs become significantly more frequent than others, the entropy tends to decrease. This way, it is possible to transform the IP address information into a quantitative value.

Equation 5 shows how Shannon's entropy is calculated [60], where i represents an individual event, or an IP in the previous example; p_i the probability of the event to occur, or the frequency of i among all IP occurrences on that day; and N the number of possible events, or the number of different IPs that appeared in the last second. The resulting entropy H is then calculated for that dimension in that second.

$$H = -\sum_{i=1}^{N} p_i \log_2(p_i) \tag{5}$$

Algorithm 1 briefly explains how each second of flow is treated. The system uses a sliding window composed of seconds as an input to decide for an anomaly. It works as

a first in, first out queue so every second the oldest set of dimensions is removed to make place for the most recent one. Thus, each second is utilized 10 times for anomaly detection. It is important to understand that the system works in discrete intervals of one second. Therefore, the anomaly detection happens not in real time, but with a one second delay in its worst case scenario.

In this algorithm a threshold of 0.5 is applied to separate the network's answer (Algorithm 1 line 3). This threshold was chosen as a neutral approach for the output, as lowering it would make the system more sensible and more tolerant if increased. As an unsupervised system, the neutral approach is more adequate as the discriminator's output can be seen as a probability of the observed second has an anomaly in it.

Algorithm 1 Anomaly Detection

Require: Flows ← Last 10 seconds of flows; Trained Discriminator
Ensure: Is the current second normal or anomalous
1: flow_dimensions ← Extract Dimensions(Flows)
2: y ← Discriminator(flow_dimensions)
3: **if** y > 0.5 **then**
4: return Anomalous
5: **else**
6: return Normal

1) SYSTEM'S HYPERPARAMETERS

Just as with every machine learning model, this system required hyperparameter tuning. In this section the most important system's hyperparameters will be described:

- Neurons and layers in discriminator and generator. This parameter defines how powerful the network is at extracting patterns from data. If there are not enough layers and neurons, the network will not be capable to fit to the data. When an excessive amount is given to a network, a longer training will be required, as well as a larger dataset, otherwise the model will not be able to generalize the problem and overfit to the training data.
- Number of input seconds. This is the amount of seconds the discriminator will receive from data to determine if there is an anomaly in the last second. Consequently, the generator will also create that many seconds during training. Too few seconds make harder for the network to separate outliers from real attacks, while too many will hinder the generator's training, consequently causing the discriminator to underperform.
- Number of training epochs. It represents how many times the whole dataset will be used to train the neural networks. A sufficient amount is necessary for both networks to converge. However, it is possible for the model to overfit to the training data and loose it's generalization capacity. In GAN networks, this hyperparameter can be divided in three, since the discriminator and generator are trained alternately. The discriminator and generator can each have their own value of epochs. Additionally, the loop for the alternated training is also considered an amount of epochs.
- Batch Size. This parameter controls how often the network weights are updated, since the number represents the amount of entries of data are considered in each weight update. A smaller value represents a more detailed adjustment, which slows the training and is more susceptible to overfitting.

TABLE 1. Considered hyperparameters.

Hyperparameter	Tested Values
Total Layers in Generator	3, **4**, 5
Neurons in Generator	16, **32**, 64
Total Layers in Discriminator	3, **4**, 5
Neurons in Discriminator	**16**, 32, 64
Total Epochs (Training cycles)	**32**, 64, 96, 128
Generator Epochs	5, **10**, 15
Discriminator Epochs	5, **10**, 15
Input Seconds	5, **10**, 15, 20
Batch Size	**60**, 120, 180, 240

The system's hyperparameters were tuned using the second dataset that will be presented in a later section. Table 1 presents the considered hyperparameters and points out the chosen ones in bold. It is important to highlight how some hyperparameters have different values for each neural network, since they are different and their assignments have different difficulty levels. It was observed that when one of the networks become too dominant over the other, the tendency is to stagnate the training process due to lack of improvement from the other. To avoid this, the number of neurons and individual epochs should be tuned.

The networks architecture used in this work is similar to a funnel, since the discriminator's input is bigger than the output. For this reason, in Table 1 "Neurons in Generator" refers to the number of neurons in the first layer, which gets lower after each layer. The same is valid for the discriminator since its input is a matrix representing the last seconds of traffic and its output a single number.

The final structure of each neural network is as follows: the generator has an input layer followed by three dense layers with 32, 16 and 60 neurons, respectively. Between those layers there is a dropout of 0.3 to avoid an overfit during training and increase the chances of the network discovering new "classes" of traffic. For example, the generator could learn to reproduce samples of peak traffic but never generate samples of lower movement moments. The 60 neurons in the last layer are used to match to the sample size from the original dataset, that used 10 seconds of 6 different dimensions. This vector is reshaped to work as an input for the discriminator.

The discriminator's final structure has a first layer of 16 GRU neurons followed by three dense layers with 8, 4, and 1, respectively. The last layer outputs the probability of the sample being an anomaly. No dropouts were used in the discriminator since there were no classes to overfit for:

the training had equal amounts of normal data and fake data. In addition, the discriminator was frequently the network that was "losing" the zero sum game, so adding difficulties to it was avoided. As shown in Table 1, there was no success of adding extra epochs to the discriminator.

During the system's development, there was an issue with how both networks behaved. During the training cycle of alternating the training of each network, both networks could overcome the other after its fit. For example, during the discriminator fit, it would start with a low score but in few epochs it would be able to differ the fake data from the original. The same would happen with the generator: it would start with a low error and it would increase during training (remember that the generator's objective is to maximize the discriminator's error). This is expected since the objective of each training was being fulfilled. However, when the discriminator was tested with data from another day, the performance would vary from terrible to close to perfection. In some cases, even in consecutive epochs, the difference could be greater than 30% of F1-Score. For this reason, a validation algorithm was developed to "stop" the training in a better spot.

Since Nash equilibrium is a rare occurrence and the performance of the discriminator was unstable as the epochs passed, a validation algorithm was applied to the discriminator to select the best one during training. To do so, the discriminator is tested after each training cycle with a subset of the training day that is never used in the fit process. At the end of the last epoch, the network from the step with the best result is chosen as the final model. Algorithm 2 presents this logic.

Algorithm 2 Model Selection During Training

Require: Val_Flows ← Validation set of flows;
Require: Generator; Discriminator; Current_Best_F_score; Current_Best_Discriminator;
Ensure: Best_Discriminator

```
Fake_Samples ← Generator(Random_Numbers)
Val_Flows ← Val_Flows + Fake_Samples
F-Score ← f_score(Discriminator(Val_Flows))
if F-Score > Current_Best_F_Score then
    return Discriminator
else
    return Current_Best_Discriminator
```

B. MITIGATION MODULE

The mitigation module is an algorithm that decides which flows should be blocked by the controller. This system must be working at all times along with the detection module, even when no threats are detected. In seconds when there are no anomalies, the active IP's are added to a safe list which temporarily prevent them to be blocked when an attack is detected.

When an anomaly is detected, the mitigation module checks the IP that receives most flows and creates a suspect list. This ensures that only traffic towards the victim gets blocked and, if there are multiple targets, when the traffic towards one diminishes, the mitigation algorithm will change the victim accordingly. The IPs on the suspect list that are not on the safe list are blocked for 20 seconds. The safe list prevents regular users to be blocked in case they become suspicious due to other users' activity, which avoids false positives. The safe list also has a timer for each IP but for 5 minutes, so once the timer runs out the IP will be susceptible to be blocked again. Both lists use a dictionary to store the IPs thus lowering the search time and avoiding to hinder the system during high traffic load moments. Algorithm 3 summarizes the mitigation process. The blocked IPs are sent to the SDN controller which will distribute block instructions to the whole network automatically.

It is possible for a legitimate user to try to have its first access to a host at the same time as an attack starts against it. In this case, the user would be considered an attacker and be blocked by the mitigation system. However, the block list would only stop those flows for 20 seconds and some attackers would be blocked some seconds later or earlier. This way, while attackers get blocked and maybe change targets, the legitimate users will eventually be unblocked and not misclassified as an attacker again.

Algorithm 3 Anomaly Mitigation

Require: Flows ← All flows from the last second
Require: Is_Anomalous ← Detection Module Veredict
Ensure: List of blocked IPs

```
if Is_Anomalous then
    Attacked IP ← IP that received greatest number of flows
    Suspect IPs ← IPs that sent flows to the attacked IP
    add suspect IPs in block list if not in safe list for 20 seconds
else
    Add flows to safe list for 5 minutes
Block flows from ips in block list
Reduce 1 second from IPs in Block list
Reduce 1 second from IPs in Safe list
```

V. PERFORMANCE EVALUATION AND RESULTS ANALYSIS

This section presents the evaluation method as well as the performance of the proposed system in two scenarios to represent different networks and attack types. All tests were made in a Windows 10 machine with a Ryzen 7 1700x, 32GB of RAM and Python version 3.10.

To evaluate the detection method, two main metrics were measured: F1-score and Matthew's Correlation Coefficient (MCC). Those metrics were chosen because the datasets studied suffer from severe imbalance between classes, which is comprehensible since both aim to be similar to real network traffic. This imbalance will be addressed for each scenario individually.

To calculate both metrics it is only required to have the true positive (TP), true negative (TN), false positive (FP) and false negative (FN). F1-score, depicted in 8 represents the harmonic mean of precision and recall, represented in 6 and 7 respectively. Despite being a good indicative of balance between precision and recall, the F1-score has a flaw that may be significant to this work: it does not consider the true negatives (the benign class). This way, this metric might return a high value even if the majority of the normal seconds are classified as attacks. In the F1-score equation, if the classes are inverted, a worse result would be reached, showcasing the true performance of the model. For this reason, another metric was added to the evaluation: the Matthews' correlation coefficient.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{6}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{7}$$

$$F1 - score = 2\frac{Precision \times Recall}{Precision + Recall} \tag{8}$$

The Matthews' correlation coefficient is also a metric used to evaluate models when there is imbalance in the dataset. Represented in 9, this metric is applicable to binary problems. The output of this metric can vary between -1 and 1, with 1 characterizing a perfect score, 0 a random prediction and -1 an inverse prediction. In this equation, it is possible to invert the positives and negatives (in this study's case anomaly and normal) and the result would still be the same since there is not a preferred class, even when an imbalanced dataset is used.

$$MCC = \frac{TN \times TP - FN \times FP}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{9}$$

A. FIRST SCENARIO

The first dataset was generated by the computer networks study group from State University of Londrina. The *Mininet* SDN emulator was used to create a network with six switches structures in a tree topology with one switch as a root and five as leaves. Each of the leaf switches are connected to twelve hosts to form a total of 60 connected users. The *Scapy* Python library generated artificial traffic to be sent over the network and the software *Hping3* generated distributed denial of service attacks.

This dataset contains three 24-hour days of traffic in form of labeled flows. One of the days does not have any attacks, while the other two have each a portscan and a DDoS in different time intervals. To fit in the proposed system, the flows were aggregated into groups of one second with the dimensions described in section IV and then normalized to the interval $[0, 1]$. Fig. 2 presents the testing day of this scenario.

As mentioned at the start of this section, this dataset suffers from imbalance caused by a discrepancy between the amount of normal seconds and anomaly seconds. Each attack day contains one hour and fifteen minutes of DDoS and an hour and five minutes of portscan attacks. This is not a problem for the system's training since only normal traffic is used to do so, but it is possible to misevaluate the performance when metrics such as accuracy are employed.

B. SECOND SCENARIO

The second dataset named CIC-DDoS2019 was created by the Canadian institute for Cyber security from the University of New Brunswick. The dataset has two days of 8 hours each, with a variety of distributed denial of service attacks. The first day contains the following DDoS types: Portmap, NetBIOs, LDAP, MSSQL, UDP, UDP-Lag, and SYN while the second has NTP, DNS, LDAP, MSSQL, NetBIOS, SNMP SSDP, UDP, UDP-Lag, WebDDoS, SYN, and TFTP.

The background traffic was emulated by a profiling system that imitate human interactions of 25 users in different operational systems and collected by a network traffic analysis tool called CICFlowMeter-V3. The data has invalid numbers such as NaN and inf in some fields. They were treated by calculating their true values, if possible, and set to zero otherwise. For example, if a flow has NaN in the bytes per second field, it is first checked if it is possible to use the total bits and flow duration fields to replace the invalid entry. This way, the data inaccuracies are minimized.

This dataset also has imbalance that requires addressing. It has more than 26 thousand seconds without anomalies, and only 8 thousand with. Most of the imbalance is caused by moments when there is no traffic in the network at all. It is plausible to assume that there are no attacks when there is no traffic and no need for an action of an anomaly detection system. Therefore, the empty seconds were disregarded and dropped in this experiment.

To use the dataset, one of the days had all attacks removed before training the networks. Since the dataset is organized in labeled flows, the seconds with anomalies were not discarded, rather they had the anomalous flows removed and the legitimate flows could be availed. This is advantageous since it is important to supply as much data as possible to deep learning models.

In the testing day, it was noticed that some attacks had moments with close to zero packets of anomalous flows. As a distributed denial of service attack, a small amount of flows is ineffective, since the resources of the attacked system would not be exhausted. For this reason, weak DDoS attempts were ignored in this experiment, and the following process as performed to remove them from the dataset: for each attack type, every second with less flows than 1% of Q has its flows removed, where Q is the maximum flows in a single second of that attack type. Fig. 3 and 4 present the training and testing day respectively. Since the anomalies studied are distributed denial of service attacks, the moments without traffic were removed since it is not possible for a DDoS to be occurring. This makes the network training faster and removes obvious true negatives that would otherwise inflate the metrics. Consequently, the time in Figs. 3 and 4 is shorter than the actual dataset.

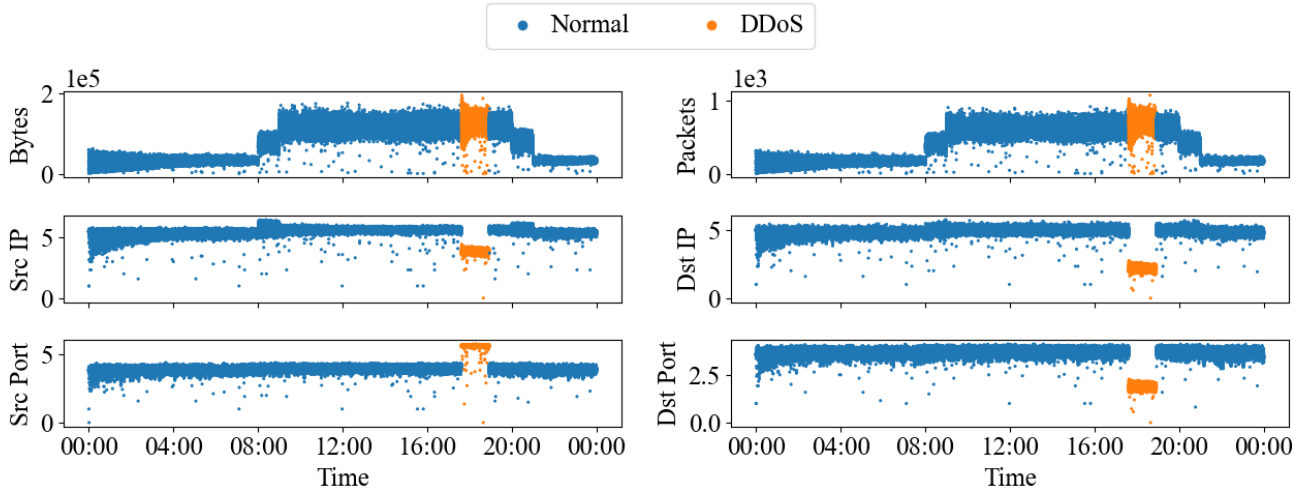


FIGURE 2. Orion dataset testing day.

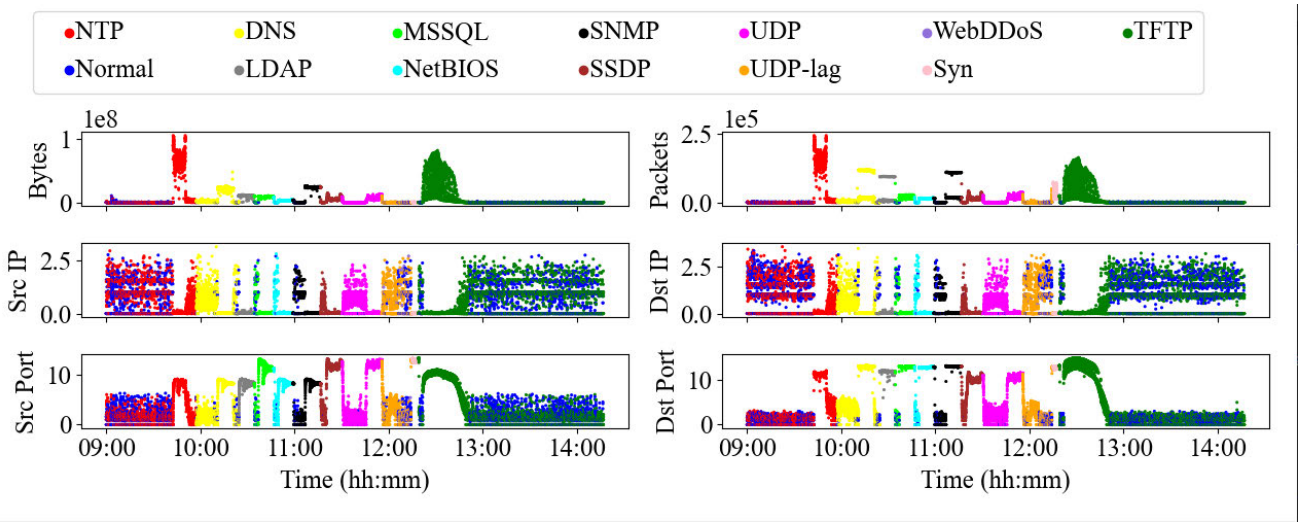


FIGURE 3. CIC-DDoS2019 dataset training day.

TABLE 2. Anomaly detection confusion matrix in first scenario.

	Prediction	
	Normal	Attack
Normal (Label)	81883	8
Attack (Label)	5	4495

TABLE 3. Anomaly detection metrics in the first scenario.

Metric	Result
Precision	0.998
Recall	0.998
F1-Score	0.998
MCC	0.998

C. RESULTS AND ANALYSIS

The confusion matrix from the first scenario is represented in Table 2, which yields the metrics presented in Table 3. The results are adequate since a minimal portion of the dataset was misclassified. Despite the greater amount of normal traffic in this testing day, the MCC is still close to one as most of the attacks were detected. Due to the model being unsupervised, the imbalance of the dataset did not affect it's performance.

For the second scenario, Table 4 presents the confusion matrix from the testing day. From the metrics in Table 5, it is noticeable how the model had a worst performance when compared with the first scenario yet still a satisfactory one. This probably happens due to the more variable traffic behavior of the emulated network, which causes more false positives, in addition to some attack moments with little impact to the monitored flow dimensions.

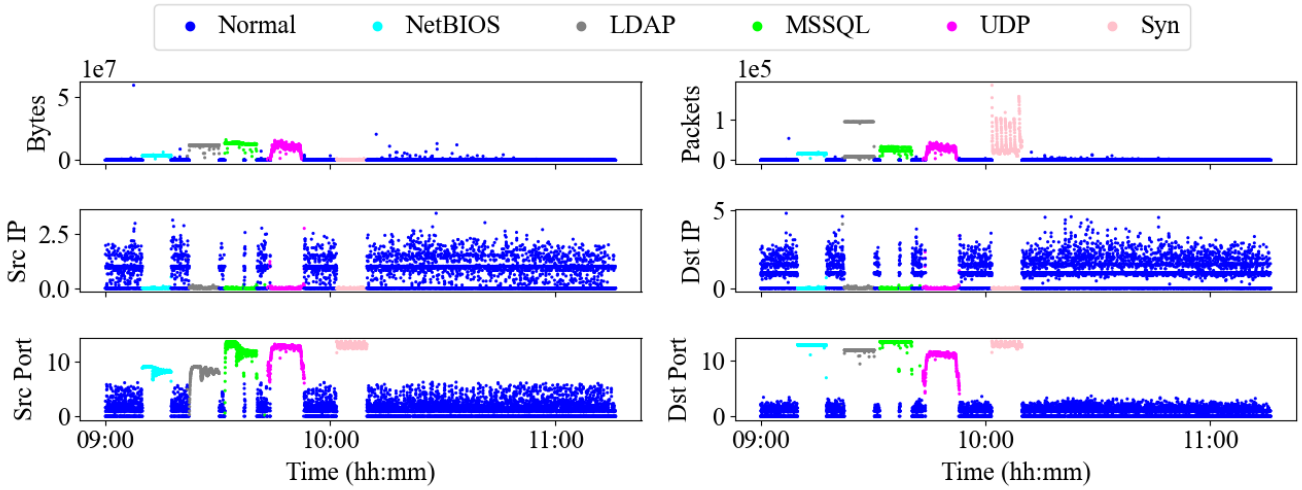


FIGURE 4. CIC-DDoS2019 dataset testing day.

TABLE 4. Anomaly detection confusion matrix in the second scenario.

	Prediction	
	Normal	Attack
Normal (Label)	5642	36
Attack (Label)	11	2473

TABLE 5. Anomaly detection metrics in the second scenario.

Metric	Result
Precision	0.985
Recall	0.995
F1-Score	0.990
MCC	0.986

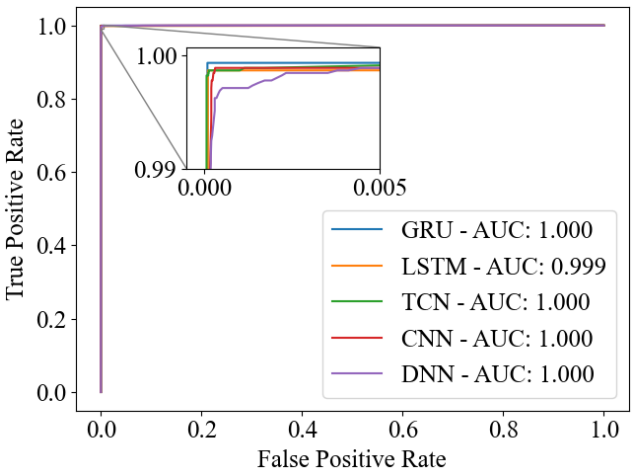


FIGURE 5. Roc curve and AUC from first scenario.

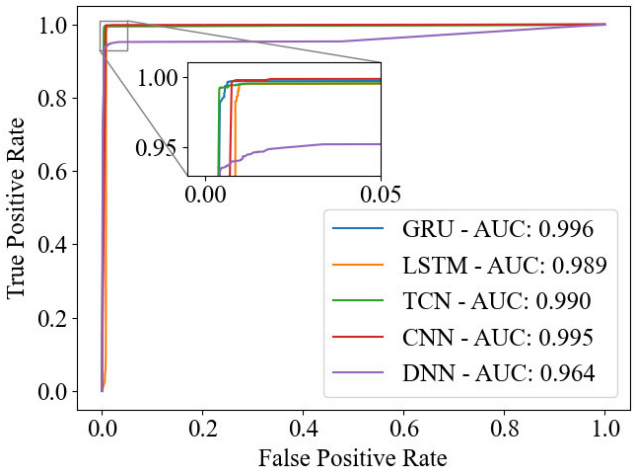


FIGURE 6. Roc curve and AUC from second scenario.

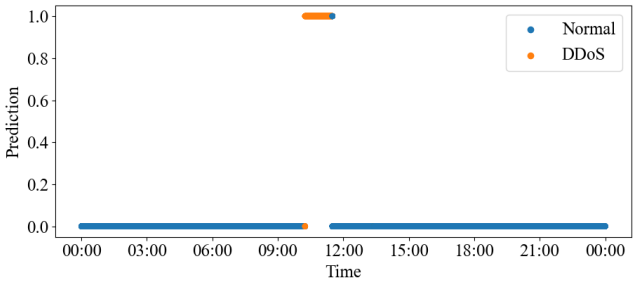


FIGURE 7. Model predictions for the first scenario test day.

Figs. 5 and 6 present a ROC curve with their corresponding area under curve from both datasets respectively. This system is unsupervised so it does not use data to find a threshold. Rather, the neutral threshold of 0.5 is applied to avoid inputting any sensibility bias to the network's decision. In this study, the purpose of the ROC curve is to present how certain the neural network is about its answers. In a perfect classifier, the distance between classes in its answer is maximized (normal class receives answers as close as possible to zero and anomaly to one), which means that the threshold would

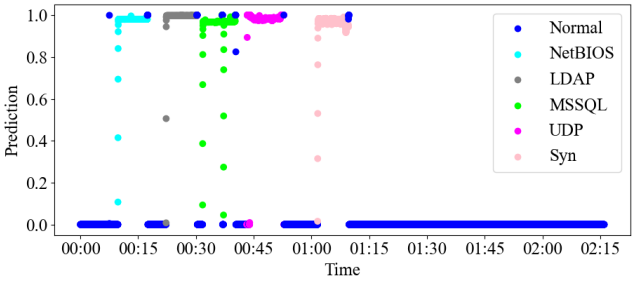


FIGURE 8. Model predictions for the second scenario test day.

not matter for it. The ROC curves in both datasets have high area under curve, which means that the network can separate each class with confidence and with high success rate. It is noticeable that in Fig. 5 the AUC is 1 for the networks, even with their result not being perfect. This happens due to an automatic rounding since the error is too small. In this case, LSTM's AUC should also be 1.

Figs. 7 and 8 are the discriminator's answers for the testing day of each dataset. It is noticeable how separated the normal and anomaly classes are, which directly impacts the ROC curve. The more separated the classes are, the higher is the model's confidence degree, which avoids false positives and negatives in less defined traffic moments. It is also possible to see that there are more false positives in the second scenario, which is caused probably by the less regular traffic present and smaller training data.

D. COMPARISON WITH OTHER METHODS

The proposed system was also implemented with four other types of neurons: long short-term memory, temporal convolutional neurons (TCN), convolutional neurons (CNN), and traditional neurons (DNN). The three neuron types GRU, LSTM and TCN were chosen due to being adapted to receive sequential data such as time series. LSTM does store context information to use in later predictions and TCN has the advantage to be able to receive large sequences of data and make use of the whole information during its prediction without the need of a context module.

Convolutional networks can be adapted to process time series, despite its main utility being image processing. In this work, two dimensional convolutions were used, thus the data is organized in a matrix where one axis represents the features and the other the time. Since the kernels extract spacial information of data, organizing in such way allows for this type of information to be exploited.

As a way of demonstrating how important the special neurons are for the performance of the system, the DNN is also present for comparison. In this work, the conventional deep network received all ten seconds of data as a single vector.

Tables 6 and 7 present the results of each type of neural network in scenarios one and two respectively. All five models present great results in both datasets, with GRU

TABLE 6. Anomaly detection comparison in the first scenario.

Network	F1-Score	MCC	AUC
GRU	0.998	0.998	1.000
LSTM	0.998	0.998	0.999
TCN	0.998	0.998	1.000
CNN	0.998	0.998	1.000
DNN	0.980	0.979	1.000

TABLE 7. Anomaly detection comparison in the second scenario.

Network	F1-Score	MCC	AUC
GRU	0.990	0.986	0.996
LSTM	0.990	0.986	0.989
TCN	0.988	0.984	0.990
CNN	0.989	0.984	0.995
DNN	0.960	0.944	0.964

achieving a slightly higher score in the ROC curves. It is safe to say that GRU, LSTM, TCN and CNN methods are tied and one may overcome the other in different training iterations. This happens due to the random nature of weight initializing and the randomization of the training data that may cause modest variation. During tests, the results from each method remained consistent to the presented results. The DNN had a good performance, but not enough to reach the other network's.

Since the four methods are close to equivalent in resulting metrics, it is fair to observe the resources required for each implementation. In this case, GRU consumes less memory and is less complex than LSTM, TCN and CNN. When compared to LSTM, GRU is close to identical to it but with only a two gates per neuron, whilst LSTM has three. Despite being a slight difference, it adds up when training with large datasets. The TCN is significantly different than GRU, it uses chained causal and dilated convolutions to process sequenced data, which increases the number of trainable parameters, training time, execution time and required memory. Convolutional neurons are not as costly as TCN ones, but do also consume more memory compared to GRU since each neuron produces a new batch of data. Each neuron represents a kernel and outputs data with the same dimension as the input minus one. Considering the complexity and memory, GRU is superior than the other network types.

In Table 8 the time in seconds to train each network in each scenario is presented. There is a significant difference between the duration of each dataset due to their size: CICDDoS2019 has only a few hours of traffic, while the Orion dataset has an entire day. The amount of traffic in each scenario does not influence significantly in the training time due to how the system works: each entry in the network is a "compilation" of a whole second of traffic. It is evident how TCN is significantly slower than the other methods due to the chained convolutions, while DNN is the fastest due to its simplicity. GRU and LSTM have similar training time with a slight advantage to GRU, as expected. The time difference

TABLE 8. Training time comparison in seconds.

Network	First Scenario	Second Scenario
GRU	98.224	82.190
LSTM	107.978	79.156
TCN	331.620	92.620
CNN	148.715	87.604
DNN	56.959	26.134

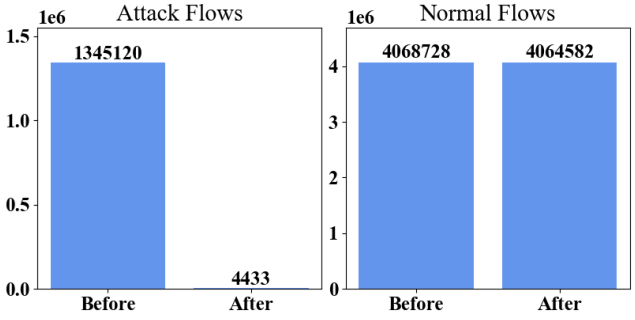


FIGURE 9. Mitigation in the first scenario.

between them would probably be emphasized by a larger dataset or more training epochs, similarly to how the time difference between them decreases in the smaller dataset.

E. MITIGATION RESULTS

After the detection module exports its result, the mitigation algorithm was tested in both datasets.

In the first scenario, it is noticeable how effective the algorithm was on blocking malicious flows without affecting a significant portion of the legitimate flows as shown in Fig. 9. Only 0.32% of attacking flows were not mitigated, and only 0.10% of the normal flows were wrongly blocked. In moments where the detection algorithm miss-classified anomalous behavior, the block list is able to exclude flows from the malicious IPs while the safe list avoids to stop normal traffic.

In the second scenario, the same effectiveness is observed as presented in Fig. 10. Although there is a significant difference in the number of flows between both datasets, the mitigation was successful in blocking most anomalous flows. It is important to notice how the normal traffic in the first dataset is significantly higher than in the second due to the length of the day. The first scenario has a whole day and the other only a few hours. Meanwhile, the amount of anomalous traffic is significantly higher in the second scenario due to the amount of attack instances and types present in the dataset.

VI. COMPLEXITY ANALYSIS

During the design of this system, the complexity and scalability were considered. This analysis is divided in three modules: collection, detection and mitigation. In this case, the $O(n)$ notation will be used with n being the number of flows in each second.

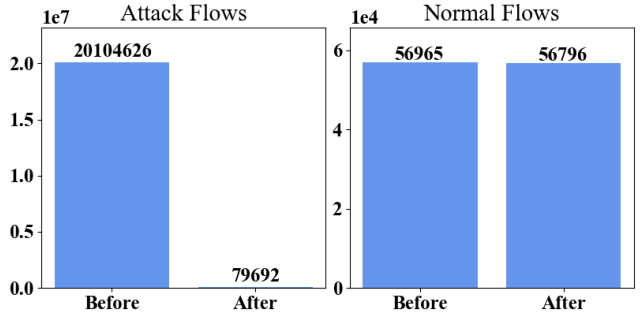


FIGURE 10. Mitigation in the second scenario. Notice the different scale in each graphic.

In the collection step, the flows are received from the controller and the flow dimensions are extracted and organized. The calculation of bits and packets per second is just a sum, so it is $O(n)$. The entropy calculation goes through the data 3 times for each, so $O(3n)$. Thus, the data processing step is considered $O(n)$.

The detection phase uses the neural network's calculation to define if an anomaly is present in that second. Since the data is already processed, the network is not required to go through the data, so the complexity is always the same: $O(1)$.

Finally, the mitigation step will use the list of flows to check the most frequent ones, consult the safe list and block them. This process requires to go through the data three times. The safe list is a dictionary, so the consultation time does not grow with the amount of flows. It is possible to conclude that the complexity of this step is $O(n)$.

VII. CONCLUSION

This work presented an unsupervised anomaly detection system to detect distributed denial of service attacks on software-defined networks using generative adversarial networks. When trained only with regular traffic, the GAN discriminator is able to identify network attacks as anomalies without previous knowledge. The GAN discriminator has a gated recurrent unit layer to store context from previous traffic seconds and help with anomaly detection.

This model was tested with two datasets: the Orion dataset, generated by the computer networks study group from the State University of Londrina, and CIC-DDoS2019, a widely used dataset for benchmarking anomaly detection systems. In both datasets, six flow dimensions were extracted to feed the neural network with only regular traffic during training.

The first scenario results show that the neural network was able to separate regular traffic from anomalous as shown in Figure 7. The detection resulted in an f1-score of 0.998 and a Matthew's correlation coefficient of the same value. In the second scenario, the model was also able to detect most of the attacks and achieved an f1-score of 0.985 and an MCC of 0.979. It is noticeable how the imbalance of the datasets with the minority of the attack class did not significantly affect the detection's performance since the training data does not require anomalies.

In the mitigation phase, the detection output determines how the incoming flows will be treated using a block list and a safe list. In this phase, the imbalance changes in both datasets. The first is quite balanced, while the second now has a majority of attack flows. In the two, 99% of the malicious flows become blocked, and less than 1% of legitimate flows get discarded.

The proposed model was compared with other deep-learning techniques. They are long short-term memory, temporal convolutional networks and convolutional neural networks, two models designed to extract characteristics from time series and one from images that could be alternatives in the proposed model. All three networks achieve results within 1% from GRU's. Due to the random nature of neural network training, one may achieve a better result than the others, but they would still be technically tied. Deep Neural Network with Traditional neurons were also tested as some sort of ''control group'' to show how necessary the special neurons are. The network still had good results but was clearly worse than the others. The time each network takes to evaluate the whole dataset was also measured. The GRU and LSTM models were tied. Both CNN and TCN were slower and DNN was faster than all the others.

For future work, our group plans to study and test smaller intervals of time to collect metrics and apply the detection to reduce the system's reaction time. Other flow dimensions are also being studied to represent the network's baseline of regular traffic, such as average packet size and flow duration. Additionally, other datasets are also being studied to benchmark anomaly detection systems. For example, UGR'16 has real traffic collected by a Spanish internet service provider with many weeks of data. It has as an advantage the massive amount of data and a completely different network size, which can be great to evaluate the system's scalability. At last, we plan to study transformer networks by developing a system that explores their self attention capability as well as diffusion models in order to explore their capabilities on generating accurate data.

REFERENCES

[1] I. T. Abdel-Halim and H. M. A. Fahmy, ''Toward efficient vehicular-based virtual network infrastructure for smart cities,'' *Eng. Sci. Technol., Int. J.*, vol. 44, Aug. 2023, Art. no. 101456.

[2] Q. Liu and T. Zhang, ''Deep learning technology of computer network security detection based on artificial intelligence,'' *Comput. Electr. Eng.*, vol. 110, Sep. 2023, Art. no. 108813.

[3] R. Kumar, U. Venkanna, and V. Tiwari, ''Optimized traffic engineering in software defined wireless network based IoT (SDWN-IoT): State-of-the-art, research opportunities and challenges,'' *Comput. Sci. Rev.*, vol. 49, Aug. 2023, Art. no. 100572.

[4] X. Etxezarreta, I. Garitano, M. Iturbe, and U. Zurutuza, ''Software-defined networking approaches for intrusion response in industrial control systems: A survey,'' *Int. J. Crit. Infrastruct. Protection*, vol. 42, Sep. 2023, Art. no. 100615.

[5] A. Narwaria and A. P. Mazumdar, ''Software-defined wireless sensor network: A comprehensive survey,'' *J. Netw. Comput. Appl.*, vol. 215, Jun. 2023, Art. no. 103636.

[6] N. S. Shaji, T. Jain, R. Muthalagu, and P. M. Pawar, ''Deep-discovery: Anomaly discovery in software-defined networks using artificial neural networks,'' *Comput. Secur.*, vol. 132, Sep. 2023, Art. no. 103320.

[7] A. B. de Neira, B. Kantarci, and M. Nogueira, ''Distributed denial of service attack prediction: Challenges, open issues and opportunities,'' *Comput. Netw.*, vol. 222, Feb. 2023, Art. no. 109553.

[8] M. L. Proenca, B. B. Zarpelao, and L. S. Mendes, ''Anomaly detection for network servers using digital signature of network segment,'' in *Proc. Adv. Ind. Conf. Telecommun./Service Assurance Partial Intermittent Resour. Conf./E-Learning Telecommun. Workshop (AICT/SAPIR/ELETE)*, 2005, pp. 290–295.

[9] S. Alem, D. Espes, L. Nana, E. Martin, and F. De Lamotte, ''A novel bi-anomaly-based intrusion detection system approach for Industry 4.0,'' *Future Gener. Comput. Syst.*, vol. 145, pp. 267–283, Aug. 2023.

[10] M. V. O. de Assis, L. F. Carvalho, J. J. P. C. Rodrigues, and M. L. Proença, ''Holt-winters statistical forecasting and ACO metaheuristic for traffic characterization,'' in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2013, pp. 2524–2528.

[11] J. I. I. Araya and H. Rifa-Pous, ''Anomaly-based cyberattacks detection for smart homes: A systematic literature review,'' *Internet Things*, vol. 22, Jul. 2023, Art. no. 100792.

[12] G. F. Scaranti, L. F. Carvalho, S. Barbon, and M. L. Proença, ''Artificial immune systems and fuzzy logic to detect flooding attacks in software-defined networks,'' *IEEE Access*, vol. 8, pp. 100172–100184, 2020.

[13] W. Lim, K. S. C. Yong, B. T. Lau, and C. C. L. Tan, ''Future of generative adversarial networks (GAN) for anomaly detection in network security: A review,'' *Comput. Secur.*, vol. 139, Apr. 2024, Art. no. 103733.

[14] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich, ''A survey on anomaly detection for technical systems using LSTM networks,'' *Comput. Ind.*, vol. 131, Oct. 2021, Art. no. 103498.

[15] A. González-Muñiz, I. Díaz, A. A. Cuadrado, D. García-Pérez, and D. Pérez, ''Two-step residual-error based approach for anomaly detection in engineering systems using variational autoencoders,'' *Comput. Electr. Eng.*, vol. 101, Jul. 2022, Art. no. 108065.

[16] T. A. Naidu and S. Kumar, ''Impact of deep learning models on hate speech detection,'' in *Proc. 12th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Jul. 2021, pp. 1–5.

[17] L. Duan, Y. Ren, and F. Duan, ''Adaptive stochastic resonance based convolutional neural network for image classification,'' *Chaos, Solitons Fractals*, vol. 162, Sep. 2022, Art. no. 112429.

[18] M. A. Contreras-Cruz, F. E. Correa-Tome, R. Lopez-Padilla, and J. P. Ramirez-Paredes, ''Generative adversarial networks for anomaly detection in aerial images,'' *Comput. Electr. Eng.*, vol. 106, Mar. 2022, Art. no. 108470.

[19] S. Shahriar, ''GAN computers generate arts? A survey on visual arts, music, and literary text generation using generative adversarial network,'' *Displays*, vol. 73, Jul. 2022, Art. no. 102237.

[20] T. Song, Y. Ren, S. Wang, P. Han, L. Wang, X. Li, and A. Rodriguez-Patón, ''DNMG: Deep molecular generative model by fusion of 3D information for de novo drug design,'' *Methods*, vol. 211, pp. 10–22, Mar. 2023.

[21] S. Tripathi, A. I. Augustin, A. Dunlop, R. Sukumaran, S. Dheer, A. Zavalny, O. Haslam, T. Austin, J. Donchez, P. K. Tripathi, and E. Kim, ''Recent advances and application of generative adversarial networks in drug discovery, development, and targeting,'' *Artif. Intell. Life Sci.*, vol. 2, Dec. 2022, Art. no. 100045.

[22] M. Hashimoto, Y. Ide, and M. Aritsugi, ''Anomaly detection for sensor data of semiconductor manufacturing equipment using a GAN,'' *Proc. Comput. Sci.*, vol. 192, pp. 873–882, Jan. 2021.

[23] H. Lu, V. Barzegar, V. P. Nemani, C. Hu, S. Laflamme, and A. T. Zimmerman, ''Joint training of a predictor network and a generative adversarial network for time series forecasting: A case study of bearing prognostics,'' *Exp. Syst. Appl.*, vol. 203, Oct. 2022, Art. no. 117415.

[24] T. Zhou, Q. Li, H. Lu, Q. Cheng, and X. Zhang, ''GAN review: Models and medical image fusion applications,'' *Inf. Fusion*, vol. 91, pp. 134–148, Mar. 2023.

[25] H. Navidan, P. F. Moshiri, M. Nabati, R. Shahbazian, S. A. Ghorashi, V. Shah-Mansouri, and D. Windridge, ''Generative adversarial networks (GANs) in networking: A comprehensive survey & evaluation,'' *Comput. Netw.*, vol. 194, Jul. 2021, Art. no. 108149.

[26] P.-Y. Tseng, P.-C. Lin, and E. Kristianto, ''Vehicle theft detection by generative adversarial networks on driving behavior,'' *Eng. Appl. Artif. Intell.*, vol. 117, Jan. 2023, Art. no. 105571.

[27] Z. Li, H. Liu, C. Zhang, and G. Fu, ''Generative adversarial networks for detecting contamination events in water distribution systems using multi-parameter, multi-site water quality monitoring,'' *Environ. Sci. Ecotechnol.*, vol. 14, Apr. 2023, Art. no. 100231.

[28] X. Xia, X. Pan, N. Li, X. He, L. Ma, X. Zhang, and N. Ding, "GAN-based anomaly detection: A review," *Neurocomputing*, vol. 493, pp. 497–535, Jul. 2022.

[29] A. Sajeeda and B. M. M. Hossain, "Exploring generative adversarial networks and adversarial training," *Int. J. Cognit. Comput. Eng.*, vol. 3, pp. 78–89, Jun. 2022.

[30] M. Allahyani, R. Alsulami, T. Alwafi, T. Alafif, H. Ammar, S. Sabban, and X. Chen, "DivGAN: A diversity enforcing generative adversarial network for mode collapse reduction," *Artif. Intell.*, vol. 317, Apr. 2023, Art. no. 103863.

[31] P. T. Duy, N. H. Khoa, D. T. T. Hien, H. D. Hoang, and V.-H. Pham, "Investigating on the robustness of flow-based intrusion detection system against adversarial samples using generative adversarial networks," *J. Inf. Secur. Appl.*, vol. 74, May 2023, Art. no. 103472.

[32] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, D. Precup and Y. W. Teh, Eds. Aug. 2017, pp. 214–223.

[33] S. Devakumar and G. Sarath, "Forensic sketch to real image using DCGAN," *Proc. Comput. Sci.*, vol. 218, pp. 1612–1620, Jan. 2023.

[34] X. Yu, M. Li, C. Ge, P. P. Shum, J. Chen, and L. Liu, "A generative adversarial network with multi-scale convolution and dilated convolution res-network for OCT retinal image despeckling," *Biomed. Signal Process. Control*, vol. 80, Feb. 2023, Art. no. 104231.

[35] N. Qiang, Q. Dong, H. Liang, J. Li, S. Zhang, C. Zhang, B. Ge, Y. Sun, J. Gao, T. Liu, H. Yue, and S. Zhao, "Learning brain representation using recurrent Wasserstein generative adversarial net," *Comput. Methods Programs Biomed.*, vol. 223, Aug. 2022, Art. no. 106979.

[36] Q. Ni and X. Cao, "MBGAN: An improved generative adversarial network with multi-head self-attention and bidirectional RNN for time series imputation," *Eng. Appl. Artif. Intell.*, vol. 115, Oct. 2022, Art. no. 105232.

[37] H. Tang, X. Ling, L. Li, L. Xiong, Y. Yao, and X. Huang, "One-shot pruning of gated recurrent unit neural network by sensitivity for time-series prediction," *Neurocomputing*, vol. 512, pp. 15–24, Nov. 2022.

[38] D. M. Brandão Lent, M. P. Novaes, L. F. Carvalho, J. Lloret, J. J. P. C. Rodrigues, and M. L. Proença, "A gated recurrent unit deep learning model to detect and mitigate distributed denial of service and portscan attacks," *IEEE Access*, vol. 10, pp. 73229–73242, 2022.

[39] C. Tang, L. Xu, B. Yang, Y. Tang, and D. Zhao, "GRU-based interpretable multivariate time series anomaly detection in industrial control system," *Comput. Secur.*, vol. 127, Apr. 2023, Art. no. 103094.

[40] X. Pu, H. Xiao, J. Wang, W. Pei, J. Yang, and J. Zhang, "A novel GRU-TCN network based interactive behavior learning of multi-energy microgrid under incomplete information," *Energy Rep.*, vol. 9, pp. 608–616, Sep. 2023.

[41] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep recurrent neural network for intrusion detection in SDN-based networks," in *Proc. 4th IEEE Conf. Netw. Softwarization Workshops (NetSoft)*, Jun. 2018, pp. 202–206.

[42] J. Kim, Y. Kim, V. Yegneswaran, P. Porras, S. Shin, and T. Park, "Extended data plane architecture for in-network security services in software-defined networks," *Comput. Secur.*, vol. 124, Jan. 2023, Art. no. 102976.

[43] A. Bhardwaj, R. Tyagi, N. Sharma, A. Khare, M. S. Punia, and V. K. Garg, "Network intrusion detection in software defined networking with self-organized constraint-based intelligent learning framework," *Meas., Sensors*, vol. 24, Dec. 2022, Art. no. 100580.

[44] S. Wang, J. F. Balarezo, K. G. Chavez, A. Al-Hourani, S. Kandeepan, M. R. Asghar, and G. Russello, "Detecting flooding DDoS attacks in software defined networks using supervised learning techniques," *Eng. Sci. Technol., Int. J.*, vol. 35, Nov. 2022, Art. no. 101176.

[45] A. A. Wabi, I. Idris, O. M. Olaniyi, and J. A. Ojeniyi, "DDOS attack detection in SDN: Method of attacks, detection techniques, challenges and research gaps," *Comput. Secur.*, vol. 139, Apr. 2024, Art. no. 103652.

[46] S. Chaudhary and P. K. Mishra, "DDoS attacks in industrial IoT: A survey," *Comput. Netw.*, vol. 236, Nov. 2023, Art. no. 110015.

[47] Z. Zhao, Z. Li, Z. Zhou, J. Yu, Z. Song, X. Xie, F. Zhang, and R. Zhang, "DDoS family: A novel perspective for massive types of DDoS attacks," *Comput. Secur.*, vol. 138, Mar. 2024, Art. no. 103663.

[48] S. Kaur, K. Kumar, N. Aggarwal, and G. Singh, "A comprehensive survey of DDoS defense solutions in SDN: Taxonomy, research challenges, and future directions," *Comput. Secur.*, vol. 110, Nov. 2021, Art. no. 102423.

[49] H. Ding, L. Chen, L. Dong, Z. Fu, and X. Cui, "Imbalanced data classification: A KNN and generative adversarial networks-based hybrid approach for intrusion detection," *Future Gener. Comput. Syst.*, vol. 131, pp. 240–254, Jun. 2022.

[50] V. Kumar and D. Sinha, "Synthetic attack data generation model applying generative adversarial network for intrusion detection," *Comput. Secur.*, vol. 125, Feb. 2023, Art. no. 103054.

[51] A. Mustapha, R. Khatoun, S. Zeadally, F. Chbib, A. Fadlallah, W. Fahs, and A. El Attar, "Detecting DDoS attacks using adversarial neural network," *Comput. Secur.*, vol. 127, Apr. 2023, Art. no. 103117.

[52] T. Kim and W. Pak, "Early detection of network intrusions using a GAN-based one-class classifier," *IEEE Access*, vol. 10, pp. 119357–119367, 2022.

[53] H. Wang, D. J. Miller, and G. Kesidis, "Anomaly detection of adversarial examples using class-conditional generative adversarial networks," *Comput. Secur.*, vol. 124, Jan. 2023, Art. no. 102956.

[54] M. Hoh, A. Schöttl, H. Schaub, and F. Wenninger, "A generative model for anomaly detection in time series data," *Proc. Comput. Sci.*, vol. 200, pp. 629–637, Jan. 2022.

[55] Y. Li, X. Peng, Z. Wu, F. Yang, X. He, and Z. Li, "M3GAN: A masking strategy with a mutable filter for multidimensional anomaly detection," *Knowl.-Based Syst.*, vol. 271, Jul. 2023, Art. no. 110585.

[56] M. Adiban, S. M. Siniscalchi, and G. Salvi, "A step-by-step training method for multi generator GANs with application to anomaly detection and cybersecurity," *Neurocomputing*, vol. 537, pp. 296–308, Jun. 2023.

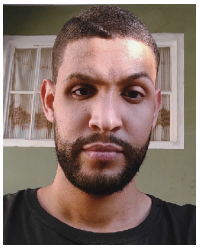
[57] X. Xu, H. Zhao, H. Liu, and H. Sun, "LSTM-GAN-XGBOOST based anomaly detection algorithm for time series data," in *Proc. 11th Int. Conf. Prognostics Syst. Health Manag.*, Oct. 2020, pp. 334–339.

[58] M. Ma, L. Han, and C. Zhou, "BTAD: A binary transformer deep neural network model for anomaly detection in multivariate time series data," *Adv. Eng. Informat.*, vol. 56, Apr. 2023, Art. no. 101949.

[59] R. Yang, H. He, Y. Xu, B. Xin, Y. Wang, Y. Qu, and W. Zhang, "Efficient intrusion detection toward IoT networks using cloud–edge collaboration," *Comput. Netw.*, vol. 228, Jun. 2023, Art. no. 109724.

[60] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

DANIEL M. BRANDÃO LENT received the bachelor's degree from the State University of Londrina (UEL), in 2021, and the master's degree, in 2023. He is currently pursuing the Ph.D. degree in electrical engineering studying computer network security and deep learning. He is a member of the Orion Research Group that studies computer networks and data communication with the Computer Science Department, UEL.



VITOR G. DA SILVA RUFFO received the B.Sc. degree (summa cum laude) in computer science from the State University of Londrina, Londrina, Brazil, in 2023, where he is currently pursuing the M.Sc. degree in computer science. Since 2021, he has been a member of the ORION Research Group, Computer Science Department, State University of Londrina. His research interests include software-defined networking, network anomaly detection, and deep learning.

LUIZ F. CARVALHO received the master's degree in computer science from the State University of Londrina, in 2014, and the Ph.D. degree in electrical engineering and telecommunications from the State University of Campinas, in 2018. He has experience in computer science with emphasis in computer networks, is part of the research group on Computer Networks and Data Communication and a Professor at Federal Technology University of Paraná. His main research interests are management and security of computer networks and Software-defined.

JAIME LLORET (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in physics, in 1997, the B.Sc. and M.Sc. degrees in electronic engineering, in 2003, and the Ph.D. (Dr.Ing.) degree in telecommunication engineering, in 2006. Since January 2017, he has been the Chair of the Integrated Management Coastal Research Institute (IGIC). He is currently a Full Professor with the Polytechnic University of Valencia. He is the Head of the ''Active and collaborative techniques and use of technologic resources in the education (EITACURTE)'' Innovation Group. He has led many local, regional, national, and European projects. Since 2016, he has been the Spanish Researcher with the highest H-index in the telecommunications journal list according to Clarivate Analytics ranking. He has authored 14 books and has more than 650 research papers published in national and international conferences, and international journals (more than 375 with Clarivate Analytics JCR). He is a Senior Member of ACM. He is a fellow of IARIA. He was an Internet Technical Committee Chair of the IEEE Communications Society and Internet Society, from 2013 to 2015. He was the Chair of the Working Group of the Standard IEEE 1907.1, from 2013 to 2018. He has been the general chair (or the co-chair) of 75 international workshops and conferences. He has been the co-editor of 54 conference proceedings and the guest editor of several international books and journals. He is the Editor-in-Chief of the *Ad Hoc and Sensor Wireless Networks* (with Clarivate Analytics JCR) and the international journal *Networks Protocols and Algorithms*. Moreover, he has been included in the world's top 2% scientists according to the Stanford University List, since 2020.

JOEL J. P. C. RODRIGUES (Fellow, IEEE) is currently with Amazonas State University, Manaus, Brazil, the Leader of the Center for Intelligence, Fecomcio/CE, Brazil, and a Full Professor with Lusfona University, Lisbon, Portugal. He is the Leader of the Next Generation Networks and Applications (NetGNA) Research Group (CNPq). He has authored or coauthored about 1250 papers in refereed international journals and conferences, three books, two patents, and one ITU-T Recommendation. He is a Member Representative of the IEEE Communications Society on the IEEE Biometrics Council. He is a member of the Internet Society, a Senior Member of ACM, and a fellow of AAIA. He has been awarded several outstanding leadership and outstanding service awards by the IEEE Communications Society and several best papers awards. He was the Technical Activities Committee Chair of the IEEE ComSoc Latin America Region Board, the Past-Chair of the IEEE ComSoc Technical Committee (TC) on eHealth and the TC on Communications Software, a Steering Committee Member of the IEEE Life Sciences Technical Community, and the Publications Co-Chair. He has been the General Chair and the TPC Chair of many international conferences, including IEEE ICC, IEEE GLOBECOM, IEEE HEALTHCOM, and IEEE LatinCom. He is the President of the Scientific Council, ParkUrbis Covilh Science and Technology Park. He was the Director for Conference Development of the IEEE ComSoc Board of Governors. He is the Editor-in-Chief of the *International Journal of E-Health and Medical Communications*. He is an editorial board member of several high-reputed journals (mainly, from IEEE). He was an IEEE Distinguished Lecturer. He is a Highly Cited Researcher (Clarivate), No. 1 of the top scientists in computer science in Brazil (Research.com).

MARIO LEMES PROENÇA JR. received the M.Sc. degree in computer science from the Informatics Institute, Federal University of Rio Grande do Sul (UFRGS), in 1998, and the Ph.D. degree in electrical engineering and telecommunications from the State University of Campinas (UNICAMP), in 2005. He is currently an Associate Professor and the Leader of the Orion Research Group that studies computer networks with the Computer Science Department, State University of Londrina (UEL), Brazil. He is a Master's Supervisor in computer science with the Computer Science Department, UEL. He is also a Ph.D. Supervisor with the Department of Electrical Engineering, UEL. He has supervised more than 150 Ph.D. and M.Sc. students and graduate and undergraduate students in computer science. He has authored or coauthored over 120 papers in refereed international journals and conferences, book chapters, and one software register patent. His research interests include computer networks, network operations, management and security, and IT governance.