Received 16 July 2022, accepted 29 July 2022, date of publication 5 August 2022, date of current version 11 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3196642

RESEARCH ARTICLE

A Deep Learning-Based Intrusion Detection and Preventation System for Detecting and Preventing Denial-of-Service Attacks

JUAN FERNANDO CAÑOLA GARCIA[1] AND GABRIEL ENRIQUE TABORDA BLANDON[2]
[1]Grupo Éxito S.A., Envigado 055428, Colombia
[2]Research Group in Automation, Electronics and Computer Science, Instituto Tecnológico Metropolitano, Medellín 050036, Colombia
Corresponding author: Juan Fernando Cañola Garcia (juancanola116639@correo.itm.edu.co)

ABSTRACT This document classifies, selects and trains a deep learning algorithm to create an IDS/IPS (Intrusion Prevention/Detection System) called Dique, which can detect and prevent denial of service (DoS) attacks. To mitigate DoS attacks, the IDS/IPS system, using the proposed deep learning model, classifies incoming packets to the web server into two classes: benign (which are normal traffic packets) and malicious (which the system considers to contain possible DoS attacks). Dique has a Graphical User Interface (GUI) where ''in real time'' you can display graphically and textually the information of captured and classified packets, and allows you to switch between the IDS mode and the IPS mode of the system operation. The proposed DoS attack classification model uses a multi-layered Deep Feed Forward neural network, the CICDDoS2019 Dataset was used for training and an accuracy of 0.994 was achieved. In addition, an offensive system called Diluvio was developed to verify the functioning of the Dique system. In Diluvio seven different types of DoS attacks were implemented (five contents in the training Datset and two that are not in said dataset) that users can selectively launch against a web server.

INDEX TERMS Denial of service attack, deep learning, intrusion detection system, intrusion prevention system, neural networks.

I. INTRODUCTION

In the field of computer security, intrusion detection is the ability to detect unauthorized access to a computer network. Such unauthorized access seriously threatens the confidentiality, integrity, and availability of the computer system and the data it stores. Generally, experts in this field use different tools and techniques that analyze network traffic to detect unusual behaviors, thus, protect data, and avoid harmful consequences.

Due to the ever-increasing size of the internet and despite efforts to protect computer systems, cybercrime continues to grow exponentially. In light of this situation, experts have felt the need to develop more sophisticated tools to detect intruders and act reactively rather than preemptively. Broadly, there are two strategies for intrusion detection from detection perspective: (1) signature-based detection, which detects intruders based on a signature database created using data from previous attacks, and (2) anomaly-based detection, which detects intruders by monitoring, collecting, and analyzing network packets that are further classified as benign or malicious.

Machine Learning (ML), which enables systems to learn without being explicitly programmed, is one of the techniques employed to classify anomalies. It uses a massive set of data (dataset) to train an algorithm to detect intruders, thus allowing the network to stay alert to possible threats. Thanks to the new data it processes, this algorithm is constantly trained to improve its classification ability [1].

Although ML seems to be a promising approach to address various cybersecurity issues, it has several drawbacks. One of them is that it could misclassify a malicious data packet as benign and accept it for learning, which would completely corrupt the algorithm, as well as to treat anomalous packets as

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

benign packets. A second drawback is that the large number of packets traveling over the network requires a high level of processing, which makes it difficult to analyze packets in real time and may affect the performance of the computer system. Given the need to improve ML techniques for intrusion detection, Artificial Intelligence (AI) experts are investigating the use of Deep Learning (DL)—a subarea of ML—methods to solve these problems. As observed in Figure 1, the performance of DL algorithms is proportional to the amount of data processed, while that of ML algorithms tends to stabilize over time [1].

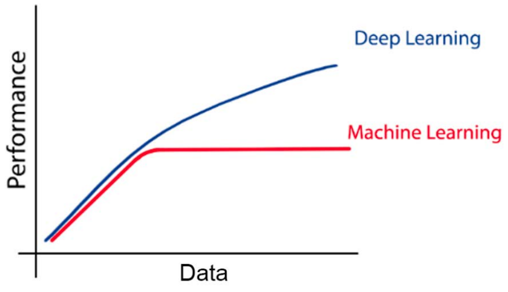


FIGURE 1. **Machine learning vs. deep learning. Adapted from the study by Xing et al [1].**

DL offers many benefits for intrusion detection when compared to ML [1]:

• *Data size*: DL algorithms perform much better with large amounts of data (millions), while ML algorithms work best with small datasets.

• *Time*: Although DL algorithms require more time for training, this additional time is compensated during the real-time production and operation stage.

• *Dedication*: DL algorithms select the features (inputs) on their own, and security experts are the ones who interpret the results based on their approach. Conversely, ML algorithms require their features and labels (outputs) to be defined.

In recent years, DL methods have been used to identify Denial-of-Service (DoS) attacks and have managed to correctly classify them. Table 1 presents the results of the best DL trained models employed to detect DoS attacks [2]. The training data set was created by the authors and classifies the packages between benign and malicious. The F1-score, precision, and recall are statistical measures used to evaluate the quality of a trained model. A value of one indicates an optimal fit.

Although training data are still reported in the scientific literature, in which different models are compared using various DL algorithms for DoS attacks and emerging datasets, these studies have only covered the detection stage. This means that implementing controls and rules to mitigate DoS attacks always requires human intervention. The purpose of this study is, thus, to go beyond the detection of DoS attacks using DL methods and propose a preventive system that, in addition to analyzing and detecting DoS attacks, allows controls to be implemented in order to autonomously mitigate

TABLE 1. **Classification results of the best deep learning trained models. Taken from the study by Apruzzese, Colajanni, Ferretti, Guido and Marchetti [2].**

Attack Name	F1-score	Precision	Recall
DOS attempt	0.9953	0.9938	0.9969
Overflow attempt	0.9939	0.9933	0.9946
SSH Brute Force login	0.9916	0.9941	0.9892
Suspicious DNS query	0.9753	0.9953	0.9586
Cache Poisoning attempt	0.9676	0.9872	0.9506
Possible Malware infection	0.9587	0.9939	0.9337
General approach (baseline)	0.7985	0.8727	0.7360

these attacks. Such system could also be used as an alternative to validate the implemented DL model.

The rest of this paper is structured as follows. Section 2 presents various key concepts, including deep learning and DoS attacks. Section 3 provides some background information on the subject and analyzes previous studies, particularly the DL algorithms and datasets used, as well as their classification performance. Section 4 describes the methodology and phases implemented to fulfill the objectives of this research. Section 5 presents the results obtained in each phase of the methodology. Finally, Section 6 draws the conclusions and makes some recommendations for future DL classification systems.

II. CONCEPTS

The most relevant concepts that support the development of this research are described below.

A. DENIAL OF SERVICE

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are well-known cyberattacks that attempt to consume the computing resources of a host or network, thus making them inaccessible to legitimate users or significantly affecting the operation of their computer system. DoS attacks can be classified into software exploits and flooding attacks. In software exploits, the attacker exploits vulnerabilities in a victim's server to disable its services or substantially decrease its performance. In flooding attacks, the attacker depletes system resources by sending a large number of false requests, thus causing the aforementioned problems [3]. The most recent version of DDoS uses the power of the network components to increase threats by distributing attacks among slave machines or zombies.

B. DENIAL-OF-SERVICE ATTACK DETECTION TECHNIQUES

The methods used to detect DoS attacks can be classified into three categories: (1) anomaly-based detection techniques, which detect unknown attacks; (2) signature-based detection techniques, which detect known attacks based on signatures; and (3) a hybrid detection technique, which combines the two aforementioned methods. Figure 2 shows the classification of DoS attack detection methods.

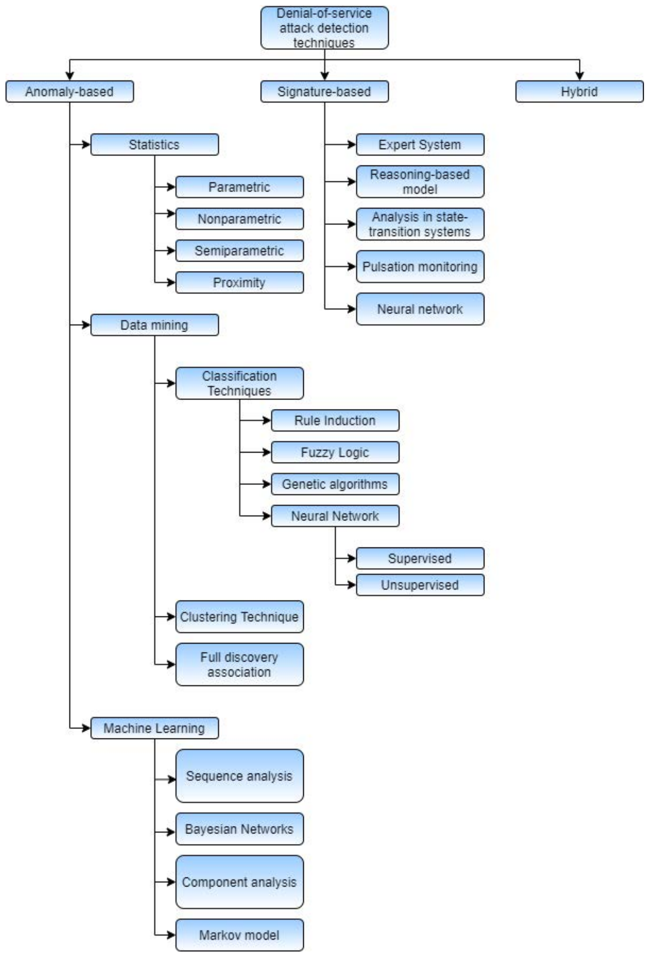


FIGURE 2. Denial-of-service attack detection techniques. Source: Adapted from the study by Zargar, Joshi, and Tipper [5].

C. MACHINE LEARNING

The concept of ML is not new. It has been perfected for years and has recently gained momentum because the technology necessary to implement it is now available.

ML has been used in different fields, such as machine vision, medical analysis, video games, and social media marketing, where it has proven to outperform algorithms based on traditional rules. ML is also being incorporated into cyberattacks detection systems to support or even replace security analysts [2].

D. DEEP LEARNING

DL algorithms perform their operations using multiple consecutive layers. These layers are interconnected, and each layer uses the output of the previous layer as input. A great advantage of DL methods is that, instead of doing it manually, they use efficient algorithms to extract hierarchical features that best represent the data. DL has been employed in a wide range of applications such as image processing, natural language processing, biomedical imaging, customer relationship management automation, and autonomous vehicle systems.

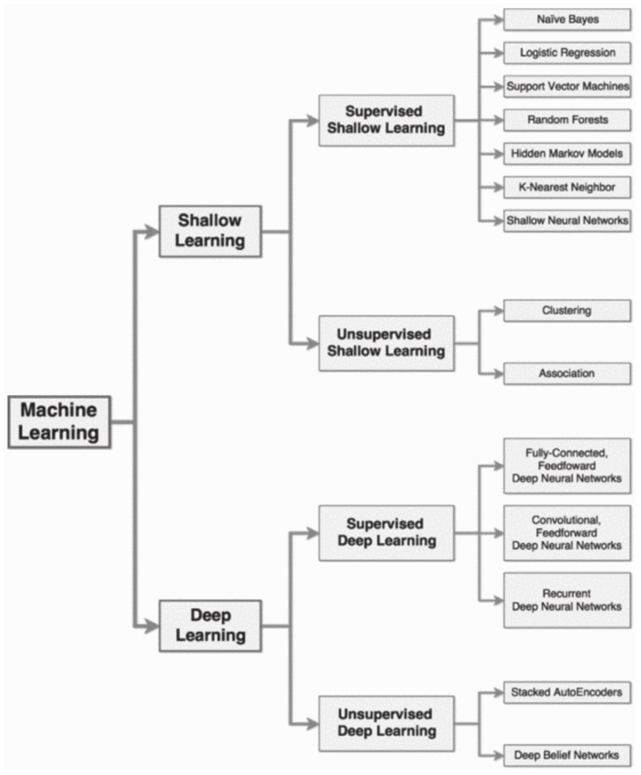


FIGURE 3. Classification of machine learning methods for cybersecurity [2].

E. CLASSIFICATION OF MACHINE LEARNING ALGORITHMS FOR CYBERSECURITY

ML can be classified into ***shallow learning*** and ***deep learning***. Shallow learning, or commonly referred to as machine learning, requires an expert in extracting features from a large amount of relevant data. In contrast, deep learning uses a multilayered model that can extract features automatically. Shallow learning and deep learning algorithms are further divided into supervised and unsupervised algorithms. The main difference between both is that, in supervised algorithms, their outputs (labels) are known, while, in unsupervised algorithms, outputs are not required [2]. Figure 3 presents the classification of the ML methods used in cybersecurity.

F. DATASET

Datasets, as the name implies, are sets of related data used to train a DL model. The most challenging step in evaluating the performance of DoS defense systems is finding the appropriate data. One way is to observe and collect information from the network. However, since collecting information from the network is expensive, datasets available on the internet can be employed.

The dataset used in this study is the CICDDoS2019 (DDoS Evaluation Dataset) [6]. This dataset contains 11 types of DoS attacks (see Figure 4), which were created in a controlled environment, and 88 features extracted using CICFlowMeter, a tool that extracts the features of pcap files. In addition, the creators of this dataset (experts from the Canadian Institute

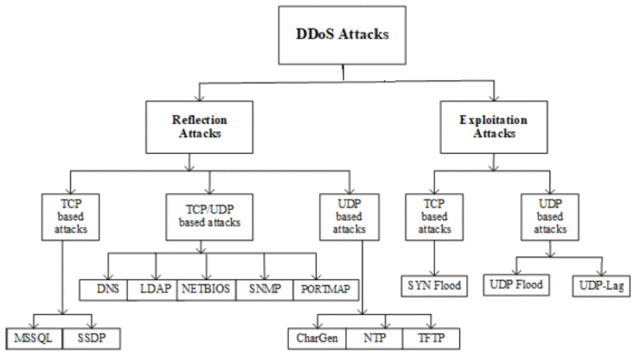


FIGURE 4. Types of DDoS attacks on the CICDoS2019 dataset [6].

for Cybersecurity) developed a process to identify the most relevant features of each DoS attack and obtained 22 features.

III. LITERATURE REVIEW

Below, we provide a brief overview of some of the studies that have been developed in the field, with an emphasis on the methods, algorithms, and datasets they used, as well as their findings. The method used to carry out the systematic review of the literature is detailed in the session Appendix.

Xing *et al.* [1], studied various ML and DL methods used for network intrusion detection. They compared different algorithms and their accuracy. However, after their implementation and comparison, they could not establish which the most effective method is because each one has its advantages and disadvantages. According to the authors, ML and DL methods do not perform well without proper data. In addition, they stress that existing public datasets (KDD99, DARPA and NSL-KDD) are outdated and uneven and creating new ones is time-consuming. Another important difficulty for these methods is that, since network information is constantly updating, models require to be retrained fast and over time.

Chockwanich and Visoottiviseth [17], presented a technique for DoS attacks using a DL model that can classify different types of attacks. The authors used supervised DL algorithms: RNN, Stacked Keras, and CNN to classify five types of attacks, using the Keras library in TensorFlow. This technique only requires the packet header information and not the payload. To verify the performance, a Dataset called MAWI was used, which are pcap files, and the result is compared with IDS Snort. The RNN algorithm obtained the best results in terms of Accuracy.

Barik and Priyadarshini [4] trained a model using the long-term memory (LSTM) algorithm. The Hogzilla and ISCX 2012 datasets were used for training. The LSTM network has been configured with 3 hidden layers, a dense layer, 128 input nodes and a drop of 0.2 for all hidden layers, which provides a good indicator performance in terms of higher accuracy and lower error rate. The final model yielded an accuracy of 98.88.

Selvakumar [8] proposed a DoS attack detection strategy known as Deep Radial Intelligence (DeeRal) with a Cumulative Incarnation Weight (CuI) optimizer and the Radial Basis Function (RBF) as the NN activation function. The datasets used to train the model are NSL KDD and UNSW NB15. The NN with the CuI optimizer had the highest performance (99.69%) in detecting DoS attacks. The article compares the result with other DoS attack detection strategies, but does not demonstrate its application in specific cases, nor does it show future work to be done with this strategy.

Chiba *et al.* [9], implemented a Network Intrusion Detection System (NIDS) using DL based on genetic algorithms and a simulated cooling algorithm. The system showed an accuracy of 99.92%. The proposed IDS was installed in a cloud datacenter, but the article does not evidence this implementation.

Kasongo and Sun [14], designed an Intrusion Detection System (IDS) based on an algorithm of recurrent NNs known as Deep LSTM that has 90 Hidden Units distributed in 3 Hidden Layers. The model yielded an accuracy of 99.51%. The DFFL structure has 29 neurons with sigmoid function in the first layer and 5 soft Max in the last layer. The article compares the results with other ML methods, showing a significant increase in performance. As future work, they plan to study the performance of each attack found in the NSL-KDD Dataset.

Siracusano *et al.* [23], trained different monitored ML algorithms to detect low-rate DoS attacks using their own external dataset. The algorithms used to train the model included Logistic Regression, K-NN, SVM, Decision Trees, Random Forest, and Deep NN. According to the researchers, the main objective of the work was to demonstrate whether the parameters of the TCP packets could be used as features to detect LDDoS attacks, which was achieved. The main limitations of the investigation were: the hardware resources that prevented increasing the number of epochs to achieve better performance and the Datasets.

Amma and Subramanian [20], developed a technique called Vector Convolutional Deep Feature Learning (VCDeepFL) to detect DoS attacks. This technique combines two DL algorithms: Vector Convolutional Neural Network (VCNN) and Fully Connected Neural Network (FCNN). The accuracy of the technique was 99.3%. The study does not demonstrate the implementation of the detection system in a test environment. As future work, the researchers propose the extraction of rules from the VCDeepFL network for each class of the Dataset and facilitate the interpretation.

Khuphiran *et al.* [18], trained a model with a DL algorithm known as Deep Feedforward using the DARPA Intrusion Detection Evaluation Dataset. The model managed to detect DoS attacks with an accuracy of 99.63%. The Dataset only contained SYN Flood attacks. The performance of the models in a test environment is not demonstrated, so the researchers propose as future work to use the two algorithms in a real-time data network.

Xu *et al.* [15], designed an IDS using DL with automatic feature extraction. The NN contained Gated Recurrent Units (GRU), Multilayer Perceptron (MLP), and softmax module.

TABLE 2. Relationship between algorithms vs dataset of DL for DoS attacks.

Dataset / Algorithms	MawiLab 2017	ISCX 2012	NSLKDD	UNSW NB15	CICIDS2017	CNTC 2017	CSIC 2010	DARPA 1998	Netflow	KDD CUP 99	Darpa 2009 DDoS	Yahoo S5 Web	Dataset propio SSE
Recurrent Neural Network	100 [17]		98.8 [1]							99.49 [13]			
Convolutional Neural Network	100 [17]					94,11 [7]		99,36 [7]	99.41 [1]				
Stacked Recurrent Neural Network	100 [17]												
Long short Term Memory		98,88 [4]	99,51 [14]			95,12 [7]	96,13 [7]	99,98 [7]		99.8 [1]			
Deep Radial Intelligence			99,69 [8]	96,15 [8]									
Genetic Algorithm(Simulated Annealing Algorithm (SAA)					99.92 [9]								
VCNN/FNN			99.2 [20]										
Deep Belief Network			97.5 [1]						97.6 [1]	93.49 [1]			
Gradient Recurrent Unit									84.15 [1]				
Deep Feed Forward			93.78 [19]								99.63 [24]		
(Bidirectional Gradient Recurrent Unit)BGRU + MLP			99.24 [15]							99.84 [15]			
Restricted Boltzmann Machine			73.23 [16]							92,12 [10]			
CNN + LSTM												98.6 [12]	
AutoEncoders										91,86 [10]			98,99 [10]
Deep Defense(CNN, RNN, LSTM, GRU)		97,60 [10]											

The NSL-KDD and KDD 99 datasets (with an accuracy of 99.42%) were used to train and test the model, which managed to detect DoS attacks with a rate of 99.98% and 99.55%, respectively. Also, the results revealed that GRU is more appropriate as a memory unit than LSTM. According to the article, the proposed system was only theoretical and requires engineering work to put it into practice, so they propose that the next step would be to optimize the system so that it can be applied in real-world environments.

Amarasinghe *et al.* [19], developed a framework to detect anomalies using DL. They trained the network with the NSL KDD dataset and used the Deep Feedforward algorithm. The accuracy of the proposed framework was 98.6217%. The researchers mention that the detection system is in the process of being implemented and they propose, as future work, to provide information on the predictions made by the framework.

Unal [11], employed the NSL-KDD dataset to evaluate the performance of a DL-based DDoS attack detection model. For this purpose, they carried out two different experiments. In the first experiment, the proposed NN detected DDoS attacks with a classification accuracy of 0.988 using all features in the dataset. In the second experiment, in which the number of features was reduced to 24, the NN classified such cyberattacks with an accuracy of 0.984. As future work, the authors propose to increase the diversity of DDoS attacks and system configurations to test the model in different environments and, of course, in a real-world environment. They also suggest building a system based on the proposed model to make the scientific community aware of the new challenges in the detection of DDoS attacks.

Yuan *et al.* [21], employed the Random Forest and LSTM methods to identify DoS attacks. According to their results, DL algorithms perform better than ML algorithms. The accuracy obtained with the LSTM algorithm was 97.606%. As future work they propose: increase the diversity of DDoS attacks, the configuration of the system to test the model in different environments and the creation of a new Dataset to identify DDoS attacks.

The intersection between the column (Dataset) and the row (DL Algorithms) contain the accuracy and the bibliography reference that reports the study.

Finally, Kim and Cho [12], trained a DL algorithm to detect anomalies using a combination of convolutional NNs and LSTM. The resulting model was compared to other DL methods and proved to have a significant performance, with an accuracy of 98.6%. The study demonstrates how the proposed model extracts features that could not be extracted in previous anomaly detection studies using conventional ML methods. Additionally, the method presents a delay to detect anomalies

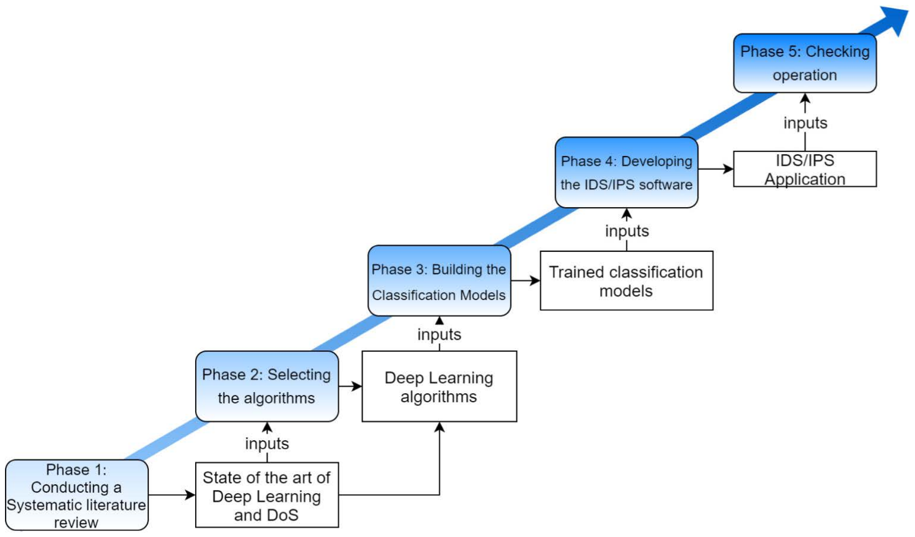

Phase 1: Conducting a Systematic literature review
State of the art of Deep Learning and DoS
inputs
Phase 2: Selecting the algorithms
Deep Learning algorithms
inputs
Phase 3: Building the Classification Models
Trained classification models
inputs
Phase 4: Developing the IDS/IPS software
IDS/IPS Application
inputs
Phase 5: Checking operation

FIGURE 5. Phases of the methodology implemented in this study.

with real data, due to the preprocessing that is required with the new data, which represents a challenge for future works.

IV. METHODOLOGY

This section describes the procedures, scientific techniques, activities, and other strategies we employed to conduct this research. Figure 5 illustrates the methodology implemented in this study.

A. PHASE 1: CONDUCTING A SYSTEMATIC LITERATURE REVIEW (SLR)

In order to characterize the algorithms and DoS datasets that have been used in studies in the field, papers on DL methods and DoS attacks published between 2009 and 2020 were retrieved from different scientific databases. This phase included the following steps:

1) Defining the research questions for the SLR
2) Specifying the search terms to be entered into the database search box
3) Selecting the scientific databases
4) Defining the inclusion and exclusion criteria to select relevant studies
5) Assessing the quality of the studies selected in the previous step
6) Analyzing and synthesizing the studies that meet all quality criteria and contribute to the proposed research.

The results of the SLR are presented in Sections Concepts and Literature Review. In addition, as a synthesis, the Table 2 was built to characterize the various DL algorithms employed to detect DoS attacks, as well as the datasets used to train models.

B. PHASE 2: SELECTING THE ALGORITHMS

This phase explains how the most appropriate algorithms for detecting DoS attacks were selected.

TABLE 3. Model proposed by sungur unal and hacibeyoglu [11].

Seed		7	
Epochs		1000	
Batch size		25192	
Activation function		tan	
Weights		Xavier	
Updater		Adam 0.01	
Layer 1	InputLayer	Input	41
		Output	41
Layer 2	DenseLayer	Input	41
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	2
		Loss function	MCXENT

1) ALGORITHM SELECTION CRITERIA

From the *Algorithms vs. Datasets* Table 2, we selected the most suitable algorithms based on four criteria that we defined. These criteria (see Table 9) made it possible to develop this research and fulfill the proposed objective.

2) ALGORITHM IMPLEMENTATION

Once the algorithms were selected, the training of the model by Unal [11], was replicated using the Java Deep Learning 4J library and the NSLKDD dataset in order to compare the behavior of the DL model with the results reported. Of the three algorithms that met the selection criteria, we chose the Deep Feedforward algorithm for the reasons

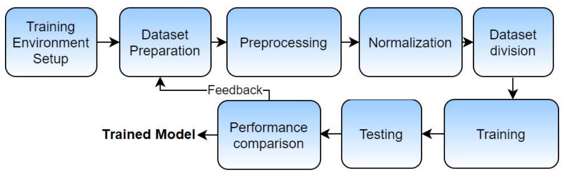


FIGURE 6. Steps of building the classification models.

TABLE 4. Training server characteristics.

Component	Specification
Processor	Intel Xeon 1.90 GHz 64-bit, 8 cores
RAM	32 GB DDR4 2666
Hard disk	Magnetic, 100 GB
Operating system	Debian 10
IDE	Eclipse
Libraries	Java 1.8, DL4J

TABLE 5. Files from the CICDDoS2019 dataset.

Name	Size	Number of records
DrDoS_DNSFile	2.0 GiB	5074414
DrDoS_LDAPFile	874.8 MiB	218153
DrDoS_MSSQLFile	1.8 GiB	4524499
DrDoS_NetBIOSFile	1.6 GiB	4094987
DrDoS_NTPFile	615.1 MiB	1217008
DrDoS_SNMPFile	2.0 GiB	5161378
DrDoS_SSDPFile	1.2 GiB	2611375
DrDoS_UDPFile	1.4 GiB	3136803
SynFile	607.8 MiB	1582682
TFTPFile	8.7 GiB	20107828
UDPLagFile	150.7 MiB	370606

explained in Section 5 and because it is a reference configuration for training the various models implemented with the CICDDoS2019 (the most recent dataset when this research was conducted). Table 3 presents the model configuration proposed by Sungur Unal and Hacibeyoglu.

C. PHASE 3: BUILDING THE CLASSIFICATION MODELS

Figure 6 describes how the classification models were built using DL.

Each step followed for building the classification models will be explained below.

1) TRAINING ENVIRONMENT SETUP

The Debian operating system was employed to train the model because it consumes few resources. Table 4 shows the server specifications used for model training.

2) DATASET PREPARATION

The CICDDoS2019 dataset (DDoS Evaluation Dataset) was selected for model training because, besides being the most recent one, it contains millions of records (48,099,733) of exclusively DoS attacks. As shown in Table 5, this dataset is divided into 11 files.

TABLE 6. List of features.

Feature	Description
Destination Port	Destination port
Protocol	Protocol number in the IP header
Flow Duration	Flow duration in microseconds
Fwd Packet Length Max	Maximum packet size in source-to-destination direction
Fwd Packet Length Min	Minimum packet size in source-to-destination direction
Fwd Packet Length Std	Standard deviation of packet size in source-to-destination direction
Flow IAT Mean	Mean time between two packets sent in the flow
Flow IAT Max	Maximum time between two packets sent in the flow
Fwd IAT Mean	Mean time between two packets sent to the source-to-destination direction
Fwd IAT Max	Maximum time between two packets sent to the source-to-destination direction
Fwd IAT Min	Minimum time between two packets sent source-to-destination direction
Fwd Header Length	Total bytes used by headers at the source-to-destination direction
Fwd Packets/s	Number of packets sent per second from source to destination
Min Packet Length	Minimum length of a packet
Max Packet Length	Maximum length of a packet
Packet Length Std	Standard deviation of the length of a packet
ACK Flag Count	Number of packets with ACK
Average Packet Size	Average packet size
Fwd Header Length.1	Packet header length in source-to-destination direction
Subflow Fwd Bytes	Average number of bytes in a subflow in the source-to-destination direction
Init_Win_bytes_forward	Total bytes sent in an initial window at the source-to-destination direction
min_seg_size_forward	Minimum size of the segment observed in the source-to-destination direction

Each record represents information in an 88-variable network packet that contains data such as source IP address, destination IP address, destination port, source port, and protocol. This packet also includes the flow information that was extracted with the CICFlowMeter tool used for feature extraction. The last column in each file shows the LABEL variable, which indicates the type of DoS attack to which each record belongs. If a record is not part of a DoS attack, it is listed as BENIGN.

According to the study carried out by the Canadian Institute of Cybersecurity, there are 22 most relevant variables, of the 88 contained in the Dataset, in terms of the importance of each characteristic for each class. For example, according to their study, the Packet_lenght_std variable is one of the most relevant for BENIGN packets. The Table 6 lists and describes these 22 features, which will be used to train the models.

3) DATASET PREPROCESSING

The training algorithm requires label values (NN output values) to be numerical. Therefore, since they are given in the form of a string in the dataset, output labels were changed to 1 for malicious and 0 for benign in the temporary file.

4) DATASET NORMALIZATION

NNs work best when input data are normalized, that is, restricted, for instance, to a range between -1 and 1. This is explained by the fact that NNs are trained using an off-center gradient and their activation functions generally have an active range between -1 and 1, which greatly improves training performance. In this case, our goal was to restrict all 22 features to a given range in order to enhance training performance.

5) DATASET DIVISION

Although in the literature, it is recommended to divide the Dataset into 70% and 30% for training and testing respectively, depending on the conditions different distributions can be applied [25]. In this research, training was prioritized, therefore, the Dataset was randomly divided into a training set (90% of the data) and a test set (10% of the data), using a program made in Java.

6) TRAINING

Several NN configurations were used for training in order to identify the model with the best classification performance. In total, 20 classification models were created. Table 7 presents the configuration of the model with the best accuracy in the training phase.

7) TRAINED MODEL TESTING

Once the training phase was completed, the trained model was stored in a.bin file. For the testing phase, we loaded the model and then evaluated 10% of the dataset records reserved for testing. The performance metrics used to evaluate the trained DL model were accuracy, precision, recall, and F1-score.

8) MODEL PERFORMANCE COMPARISON

After testing each trained model, the results were tabulated and compared based on the variables used as reference. This comparison allowed us to assess the efficiency of the proposed model and make the necessary adjustments in order to find the most efficient one. Table 11 shows the results of each trained model.

TABLE 7. Final configuration for training.

Proposed model			
Seed		7	
Epochs		200.000	
Data size		44.000	
Batch size		44.000	
Activation function		tan	
Initial weight values		Xavier	
Updater		Adam 0.001	
Features		73	
Layer 1	InputLayer	Input	73
		Output	73
Layer 2	DenseLayer	Input	73
		Output	128
Layer 3	DropoutLayer	0.5	
Layer 4	DenseLayer	Input	128
		Output	256
Layer 5	DenseLayer	Input	256
		Output	512
Layer 6	DenseLayer	Input	512
		Output	256
Layer 7	DenseLayer	Input	256
		Output	128
Layer 8	DropoutLayer	0.5	
Layer 9	OutputLayer	Input	128
		Output	2

D. PHASE 4: DEVELOPING THE IDS/IPS SOFTWARE

This section explains how the software to detect and prevent DoS attacks was developed by integrating the proposed DL model with the best performance.

1) REQUIREMENT ANALYSIS

The following were the most relevant functional requirements proposed to develop *Dique*:

1. The system must capture packets from a network interface.
2. The system must be a web application so that it can be viewed from any device.
3. The system must have a login so that only authorized users can access it.
4. The system must classify network packets into two types: malicious and benign.
5. The system must perform real-time packet classification.
6. The system must display the time, source IP address, destination IP address, source port, destination port, and protocol of the captured packets.
7. The system must include a button to start and stop packet capture.
8. The system must have a button to restart packet capture (reboot).
9. The system must store all network traffic in a pcap file for further analysis (log type).
10. A database must be created to store data on authorized users and classified packets for further validation.

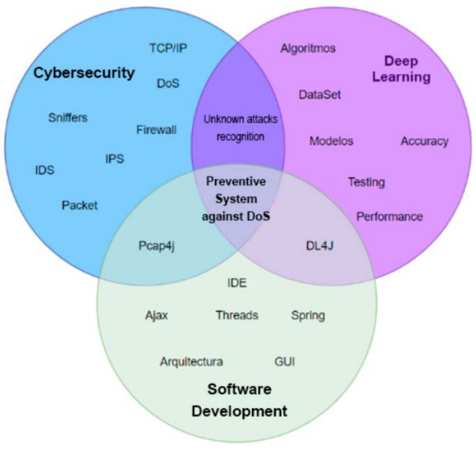


FIGURE 7. Technologies and libraries used to create the preventive system.

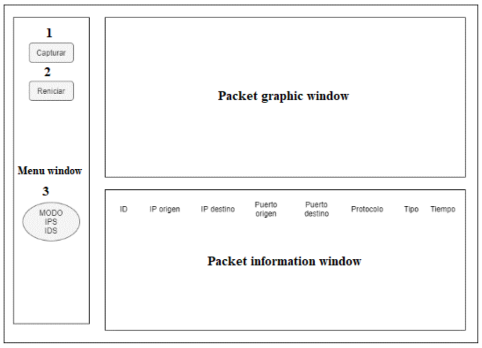


FIGURE 8. Scheme of the graphical user interface designed for the proposed preventive system.

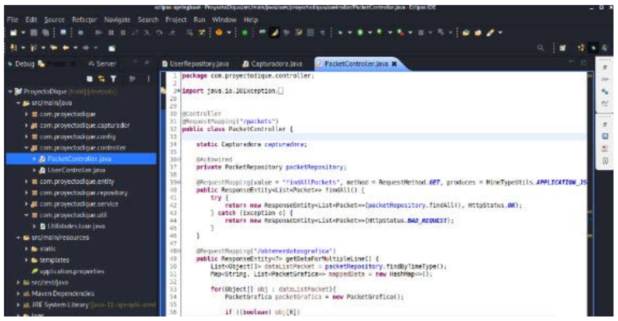
We used the Eclipse IDE with the spring boot framework to develop the preventive system. The spring boot framework imports the following Java libraries needed to create a website (including database connection pooling):

- **Maven**: Tool used to manage libraries. It allows developers to import libraries.
- **DL4J**: Java library used to train models with DL algorithms. It provides a whole set of methods that allow developers to build training networks.
- **PCAP4J**: Java library used to capture, analyze, save, and create packets from a network interface.
- **CICFlow**: Library created by the Canadian Institute for Cybersecurity to extract packet information from a pcap file.
- **Bootstrap**: CSS library used to create more user-friendly Graphical User Interfaces (GUI).
- **JQuery**: Library created using JavaScript to make calls to web services using Ajax and to execute custom events.

Figure 7 illustrates how different technologies were integrated to develop the preventive system.

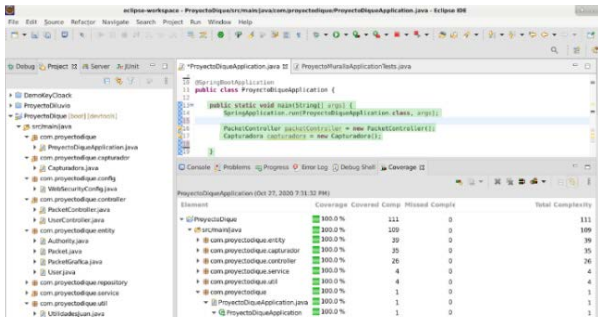
2) SYSTEM DESIGN

Figure 8 displays the designed GUI of the proposed preventive system (*Dique)*, which served as a guide for its development.


File Edit Source Refactor Navigate Search Project Run Window Help
UserRepository.java
Capturadora.java
PacketController.java

```
package com.proyectodique.controller;

import java.io.IOException;


@Controller
@RequestMapping("/packets")
public class PacketController {

    static Capturadora capturadora;

    @Autowired
    private PacketRepository packetRepository;

    @RequestMapping(value = "findAllPackets", method = RequestMethod.GET, produces = MimeTypeUtils.APPLICATION_JS
    public ResponseEntity<List<Packet>> findAll() {
        try {
            return new ResponseEntity<List<Packet>>(packetRepository.findAll(), HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<List<Packet>>(HttpStatus.BAD_REQUEST);
        }
    }

    @RequestMapping("/obtenerdatosgrafica")
    public ResponseEntity<?> getDataForMultipleLine() {
        List<Object[]> dataListPacket = packetRepository.findByTimeType();
        Map<String, List<PacketGrafica>> mappedData = new HashMap<>();

        for(Object[] obj : dataListPacket){
            PacketGrafica packetGrafica = new PacketGrafica();

            if ((boolean) obj[0])
```

FIGURE 9. Preventive system encoding in the eclipse IDE.


File Edit Source Refactor Navigate Search Project Run Window Help
*ProyectoDiqueApplication.java
ProyectoMurallaApplicationTests.java
Console
Problems
Progress
Error Log
Debug Shell
Coverage

```
@SpringBootApplication
public class ProyectoDiqueApplication {

    public static void main(String[] args) {
        SpringApplication.run(ProyectoDiqueApplication.class, args);

        PacketController packetController = new PacketController();
        Capturadora capturadora = new Capturadora();
    }
```

ProyectoDiqueApplication (Oct 27, 2020 7:31:32 PM)

Element	Coverage	Covered Comp	Missed Comple	Total Complexity
ProyectoDique	100.0 %	111	0	111
src/main/java	100.0 %	109	0	109
com.proyectodique.entity	100.0 %	39	0	39
com.proyectodique.capturador	100.0 %	35	0	35
com.proyectodique.controller	100.0 %	26	0	26
com.proyectodique.service	100.0 %	4	0	4
com.proyectodique.util	100.0 %	4	0	4
com.proyectodique	100.0 %	1	0	1
ProyectoDiqueApplication.java	100.0 %	1	0	1
ProyectoDiqueApplication	100.0 %	1	0	1

FIGURE 10. Results of the unit testing.

The designed GUI includes three components:

- **Menu window:** It contains the following three buttons that allow users to control the application: (1) the **Capture** button, which starts the analysis or classification process. (2) The **Restart** button, which has two functions: stop or restart (its label changes depending on its current status). And (3) the **IDS/IPS mode** button, which switches the execution mode between detect and prevent.
- **Packet graphic window**: It contains a two-dimensional graph showing, in real time, the number of packets that the system classifies. The horizontal axis indicates the time in seconds; and the vertical axis indicates the number of packets classified.
- **Packet information window:** It contains a table detailing information about packets, such as source IP address, destination IP address, source port, destination port, protocol, type (malicious/benign), and time when they were captured.

3) ENCODING

After defining the requirements, architecture, and design and integrating the libraries, we developed the system with Java programming language using the Eclipse IDE. Figure 9 shows a screenshot of some of the project codes in the Eclipse IDE.

4) SOFTWARE TESTING

Once the system was encoded, we validated that each requirement had been implemented and was working properly and performed unit testing. The figure 10 shows the coverage of

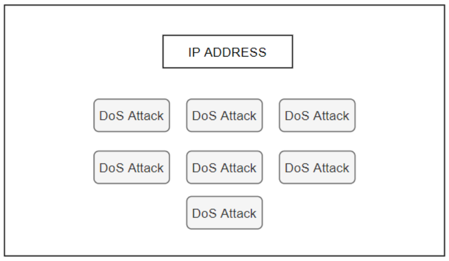


FIGURE 11. Schematic of the GUI designed for the Diluvio offensive system.

the code performed in unit tests with the eclipse eclEmma plugin and were successful; although initially, the plugin reported some structural problems in some modules of the Dique code, which were fixed to improve its operation.

E. PHASE 5: CHECKING DIQUE's OPERATION

To verify and validate the operation of the proposed preventive system (*Dique*) and the trained model, we created another application called *dique*. This application performs selective attacks on the preventive system to check if it correctly classifies each packet coming from the network. To design *Diluvio,* we followed the same steps (described below) used to develop *Dique*.

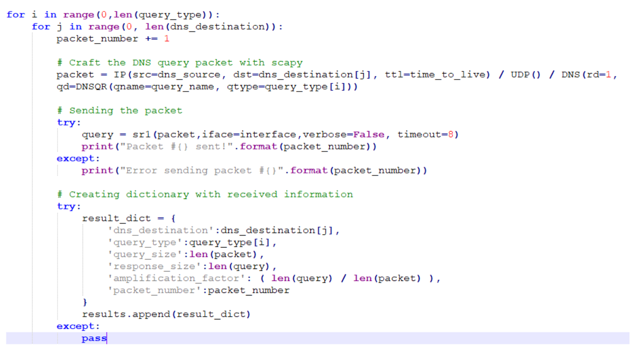
1) REQUIREMENT ANALYSIS

The requirements defined to create *Diluvio* are based on different DoS attacks falling under the five categories specified in the CICDDoS2019 dataset. Moreover, *Diluvio* was assumed to possess other DoS attacks not included in the dataset because the aim is for the proposed system to classify and prevent unknown DoS attacks. Therefore, *Diluvio* consists of seven types of DoS attacks. The following were the most relevant requirements proposed to develop *Diluvio*:

1. The system must have multiple DoS attack options so that users can select a specific attack to be executed.
2. The system must allow users to enter the IP address to which attacks will be directed.
3. The system must perform reflection DoS attacks with TCP/UDP packets.
4. The system must run exploit DoS attacks with TCP packets.
5. The system must execute exploit DoS attacks with UDP packets.
6. The system must perform reflection DoS attacks with UDP packets.
7. The system must run reflection DoS attacks with TCP packets.
8. The system must execute two types of DoS attacks that the model was not trained to detect.
9. The system must send benign packets.

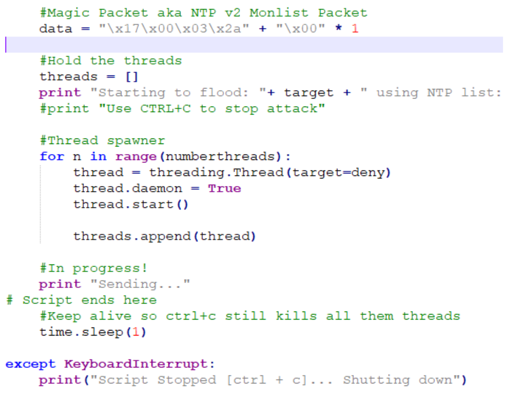
2) SYSTEM DESIGN

Figure 11 displays the system design of *Diluvio*, which served as a guide for its development.



```
for i in range(0,len(query_type)):
    for j in range(0, len(dns_destination)):
        packet_number += 1

        # Craft the DNS query packet with scapy
        packet = IP(src=dns_source, dst=dns_destination[j], ttl=time_to_live) / UDP() / DNS(rd=1,
        qd=DNSQR(qname=query_name, qtype=query_type[i]))

        # Sending the packet
        try:
            query = sr1(packet,iface=interface,verbose=False, timeout=8)
            print("Packet #{} sent!".format(packet_number))
        except:
            print("Error sending packet #{}".format(packet_number))

        # Creating dictionary with received information
        try:
            result_dict = {
                'dns_destination':dns_destination[j],
                'query_type':query_type[i],
                'query_size':len(packet),
                'response_size':len(query),
                'amplification_factor': ( len(query) / len(packet) ),
                'packet_number':packet_number
            }
            results.append(result_dict)
        except:
            pass
```

FIGURE 12. Reflection attack script with UDP packets.



```
    #Magic Packet aka NTP v2 Monlist Packet
    data = "\x17\x00\x03\x2a" + "\x00" * 1

    #Hold the threads
    threads = []
    print "Starting to flood: "+ target + " using NTP list:
    #print "Use CTRL+C to stop attack"

    #Thread spawner
    for n in range(numberthreads):
        thread = threading.Thread(target=deny)
        thread.daemon = True
        thread.start()

        threads.append(thread)

    #In progress!
    print "Sending..."
# Script ends here
    #Keep alive so ctrl+c still kills all them threads
    time.sleep(1)

except KeyboardInterrupt:
    print("Script Stopped [ctrl + c]... Shutting down")
```

FIGURE 13. NTP attack script.

3) ENCODING

After defining the requirements, architecture and design, and integrating the libraries, we continue with the coding of the *Diluvio* offensive system in the Eclipse IDE.

The execution of each of the seven DoS attacks that can be performed using *Diluvio* is described below.

- **Reflection DoS attack with UDP packets:** To perform this attack, the Python scapy library is used, which makes it possible to manipulate and create network packets. In this attack, a DNS server is queried with a spoofed IP address send to response to the victim's machine. Figure 12 shows the Python code stored in a dns_amplification.py file, which is then called from Java.
- **SYN flood DoS attack:** To perform this attack, hping3 is employed. Unlike Ping (which only sends ICMP packets), this Linux tool sends TCP and UDP packets, which is ideal to conduct cybersecurity tests like DoS tests. Hping3 makes it possible to set different parameters such as the number of packets to be sent, the port, and the flag, which, in this case, activates the parameter with the SYN flag. The implemented command (*hping3 -p 80 -c 100 -S –fast*) is then called from Java using the exec() method of the Runtime library.
- **Flood DoS attack with UDP packets**: Like the SYN flood attack, this attack can be performed using hping3. In this case, the number 2 is set as a parameter, which

TABLE 8. Selection criteria.

No.	Selection criteria
1	The algorithm is available in the Java Deep Learning library.
2	The model's training configuration used with the DL algorithm is available.
3	The dataset employed with the DL algorithm to train the model contains DoS attack packets.
4	The accuracy of the model trained with the DL algorithm is above 90%.

indicates that the packets sent are UDP packets. The command implemented with the exec() method of the Runtime library is *hping3 -c 100 -2 –faster.*

- **Reflection DoS attack with NTP packets:** To perform this attack, the Python scapy library is used. In this attack, an NTP server is queried with an impersonated IP address to send a responde to the victim's machine. Figure 13 displays the Python code stored in a ntpattack.py file, which is then called from Java.
- **Flood DoS attack with TCP packets:** To perform this attack, nping is used. Like hping3, this tool makes it possible to generate network packets with different protocols, which is ideal to conduct cybersecurity tests like DoS tests. With nping, parameters such as the type of packet, the number of packets to be sent, and the sending rate per second can be set. The implemented command (*nping –tcp-connect -rate -10 -c 100*) is then called from Java using the exec() method of the Runtime library.
- **Flood DoS attack with ICMP packets**: This attack can be performed using hping3. In this case, the number 1 is set as a parameter, which indicates that the packets sent are ICMP packets. The command implemented with the exec() method of the Runtime library is *hping3 -c 10 -1 -C 17.*
- **Slow HTTP DoS attack:** This attack can be executed with the Linux slowhttppost tool. Since this tool simulates DoS attacks at the application layer, packets can be sent at a low transfer rate. The command implemented with the exec() method of the Java Runtime library is *slowhttptest -c 20 -H -i 1 -r 2 —u*, where c denotes the number of packets; H, the Slowloris attack; i, the interval of seconds; r, the connections per second; and u, the server ip address.

4) SOFTWARE TESTING

As with *Dique*, once the system was encoded, we validated that each requirement had been developed and was working properly and performed unit testing.

V. RESULTS

This section describes the results obtained in each phase of the methodology implemented here, with the exception of Phase 1, which is described in the Appendix section.

A. PHASE 2: SELECTING THE ALGORITHMS

Of all the DL algorithms for DoS attack detection that were identified in the SLR, we selected the most suitable for this research based on four criteria defined by the researchers (listed in Table 8). Algorithms meeting the four selection criteria were considered appropriate to train the proposed models.

TABLE 9. Algorithms vs. selection criteria.

Algorithms \ Criteria	1	2	3	4	Adequate
Recurrent Neural Network (RNN)	✓	✓	✓	✓	✓
Convolutional Neural Network (CNN)	✓	X	✓	✓	
Stacked RNN	X	X	X	X	
Long Short-Term Memory (LSTM)	✓	✓	✓	✓	✓
Deep Radial Intelligence	X	X	✓	✓	
Improved Genetic Algorithm (IGA) and Simulated Annealing Algorithm (SAA)	X	X	✓	✓	
VCNN/FNN	X	X	✓	✓	
Deep Belief Network	✓	X	✓	✓	
Gated Recurrent Unit (GRU)	X	X	X	X	
Deep Feedforward	✓	✓	✓	✓	✓
Bidirectional Gated Recurrent Unit (BGRU) + Multi-Layer Perceptron (MLP)	X	X	✓	✓	
Restricted Boltzman Machine	X	X	✓	X	
CNN + LSTM	X	X	X	✓	
Autoencoders	X	X	✓	✓	
Deep Defense (CNN, RNN, LSTM, GRU)	X	X	✓	✓	

TABLE 10. Algorithms' performance.

Algorithm	Accuracy	Precision	Recall	F1-Score
Deep Feedforward	0.9	0.9353	0.9579	0.9464
RNN and LSTM	0.9555	0.9824	0.9222	0.9513

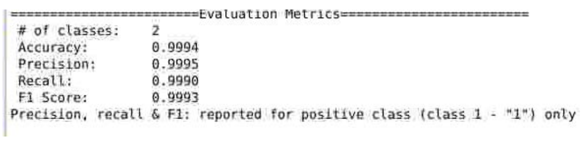
Table 9 shows the correlation between the DL algorithms for DoS attack detection reported in the literature and the four selection criteria defined in this study.

Table 10 presents the performance values obtained after testing the training of the Deep Feedforward and RNN–LSTM models with the selected algorithms using the NSL-KDD dataset, as reported in the literature.

Although the RNN–LSTM model training was replicated with the NSSL-KDD Dataset, as shown in Table 10, the algorithms using RNN and LSTM methods could not be employed to train the models. This is explained by the fact that the DL4J library required the dataset to be already organized and need no preprocessing. In other words, and technically speaking, the Sequence Record Reader DataSetIterator Java class, which iterates by means of sequential data necessary to use the RNN and LSTM methods, did not include the method to receive preprocessed data but required the dataset to already have the corresponding features and labels and to be standardized and balanced. This procedure was brazen because the initial purpose of the investigation is over-passed and would remain as a future job. On the contrary, with the algorithms using the DFNN method, all data preprocessing operations could be performed smoothly by Java code.

TABLE 11. Performance of the proposed models.

Model No.	Features	Labels	Accuracy	Precision	Recall	F1-Score
1	22	2	0.9582	0.9711	0.9835	0.9772
2	22	2	0.96	1	0.9321	0.9649
3	22	2	0.9785	0.9889	0.9765	0.9836
4	22	2	0.9282	0.9997	0.9215	0.959
5	22	2	0.9266	0.9831	0.9348	0.9583
6	73	2	0.9823	0.9997	0.9807	0.9901
7	73	2	0.9847	0.9996	0.9841	0.9918
8	73	2	0.9064	0.9064	1	0.9509
9	73	2	0.9818	0.9982	0.9817	0.9899
10	73	2	0.9859	0.9972	0.9872	0.9922
11	73	2	0.9093	0.9944	0.9038	0.947
12	73	2	0.9889	0.9913	0.9965	0.9939
13	73	2	0.9814	0.9982	0.9812	0.9896
14	73	2	0.9842	0.9966	0.9859	0.9912
15	73	2	0.9722	0.9893	0.9798	0.9845
16	73	2	0.9022	0.9022	1	0.9486
17	73	2	0.9952	0.998	0.9914	0.9947
18	73	2	0.997	0.9975	0.996	0.9967
19	73	2	0.998	0.998	0.997	0.9975
20	73	2	0.9994	0.9995	0.999	0.9993



```
========================Evaluation Metrics========================
 # of classes:    2
 Accuracy:        0.9994
 Precision:       0.9995
 Recall:          0.9990
 F1 Score:        0.9993
Precision, recall & F1: reported for positive class (class 1 - "1") only
```

FIGURE 14. Performance results of model No. 20.

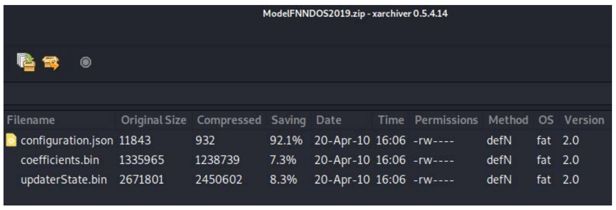
Based on the results of this phase, the most suitable DL algorithms for training models to detect DoS attacks are the DFNN, RNN, and LSTM. They all showed a good accuracy when the models were replicated using the dataset reported by different authors and met the selection criteria defined here. Due to technical specifications, of these three algorithms, we selected the DFNN algorithm, which, when trained with the CICDDOS2019 dataset, exhibited an appropriate behavior for DoS attack detection.

B. PHASE 3: BUILDING THE CLASSIFICATION MODELS

Table 11 shows the performance values obtained by the different models trained with the selected algorithm DFNN and dataset (CICDDoS2019). In said models, we modified the hyperparameters outlined in Phase 3 of the Methodology section.

As observed in Table 11, the trained model with the best accuracy was model No. 20 with two classification labels (malicious and benign). We started to train the models with the 22 most relevant features and then used the 73 features in order to improve their performance. Figure 14 presents the performance of model No. 20.

After training, each model was stored in a.zip file. This file contained three files: (a) two.bin files (one with the model's configuration, and the other with the model's weights, which can be imported or loaded into and used in the preventive



```
ModelFNNDOS2019.zip - xarchiver 0.5.4.14

Filename            Original Size  Compressed  Saving  Date       Time   Permissions  Method  OS   Version
configuration.json  11843          932         92.1%   20-Apr-10  16:06  -rw----      defN    fat  2.0
coefficients.bin    1335965        1238739     7.3%    20-Apr-10  16:06  -rw----      defN    fat  2.0
updaterState.bin    2671801        2450602     8.3%    20-Apr-10  16:06  -rw----      defN    fat  2.0
```

FIGURE 15. Binary file of a trained model.

system) and (b) a json file with the NN configuration. Figure 15 shows a screenshot of the files obtained after training a model. Only the.zip file in Java code is required for the preventive system.

The trained model with the best performance achieved an accuracy of 0.9994, a precision of 0.9995, a recall of 0.999, and a F1-score of 0.9993 for the two-class (malicious and benign) classification model described in this study.

A model with these same results (which are very promising) has not yet been reported in the scientific literature. One of the models that employs a DFNN algorithm and has yielded similar results to those reported here is a thirteen-class (twelve DoS and benign attacks) classification model that used the NSL-KDD dataset for training and obtained a precision of 0.78, a recall of 0.65, and a F1-score of 0.69 [6]. The authors that proposed such model employed ML techniques which allowed them to compare the performance of DL with that of ML in detecting DoS attacks. Regarding two-class (malicious and benign) classification models using DL techniques, two models with similar results have been reported in the literature: (1) the first one employed the NSL-KDD dataset for training and reported an accuracy of 0.9378 [19]. And (2) the second one used the DARPA 2009 DDoS attack dataset for training and achieved an accuracy of 0.9963, a precision of 0.998, a recall of 0.994, and a F1-score of 0.996 [18].

C. PHASE 4: DEVELOPING THE IDS/IPS (INTRUSION PREVENTION/DETECTION SYSTEM) SOFTWARE

The developed system is a web application called *Dique*. It was given this name because a *Dique* (dam in English) prevents floods. In this case, this application acts as a barrier to detect and prevent DoS attacks. *Dique* uses the trained model with two labels in its logic to classify packets and display their behavior in a two-dimensional graph. The different sections of this preventive system are described below.

1) LOGIN PAGE

It contains a login form through which the application is accessed. Registered users are stored in a database, along with their encrypted passwords. Figure 16 shows the login page of *Dique*.

2) HOME PAGE

It contains all the functionalities of the proposed DoS attack detection and prevention system. The working environment

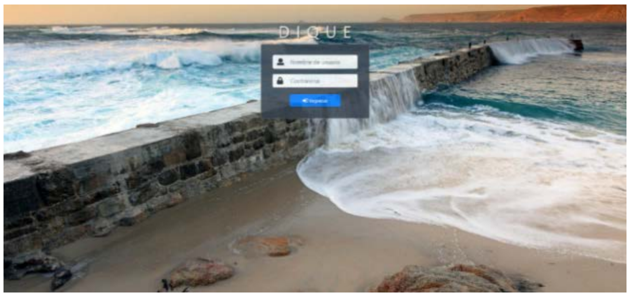


FIGURE 16. **Login page of *Dique*.**

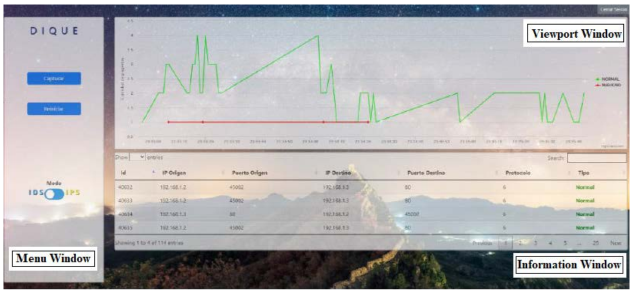


FIGURE 17. **Home page of *Dique*.**

is divided into three windows: a menu window (on the left), a packet graphic window (on the top right), and a packet information window (on the bottom right), as can be seen in Figure 17.

The three windows on the system's home page are described below.

3) MENU WINDOW

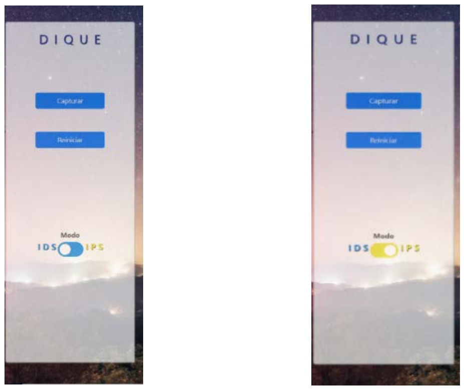
It contains three buttons that allow users to control the web application. The first button is the *Capture* button, which launches the preventive system, that is, executes all the internal logic that captures and classifies the packets. When the application starts capturing and classifying packets, this button label changes to *Stop value,* which, when clicked, stops the packet capture and classification operation.

The second button is the *Restart* button, which allows users to clear all the information on the classified packets displayed in the packet graphic and information windows and prepares the system to start a new capture.

The third button is the *Mode* toggle button, which allows users to switch between IDS (Intrusion Detection System) mode and IPS (Intrusion Prevention System) mode, as observed in Figure 18. In the IDS mode, the system will only be responsible for classifying packets, while, in the IPS mode, it will apply rules on the operating system firewall to prevent the sources of malicious packets from still entering the network system by blocking the source IP address.

4) VIEWPORT WINDOW

It shows, in a two-dimensional graph, the number of packets that are classified every second, as observed in Figure 19. The vertical axis indicates the number of packets, and the horizontal axis indicates the time in seconds when that set of packets was captured. Benign packets are shown in green; and malicious packets, in red.



A) IDS (Detection) active mode B) IPS (Prevention) active mode

FIGURE 18. **Menu window of *Dique*.**

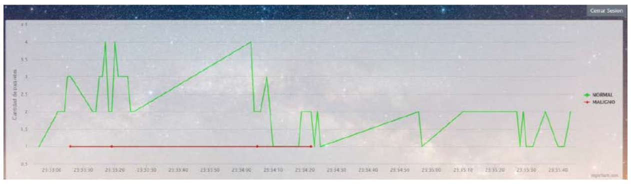
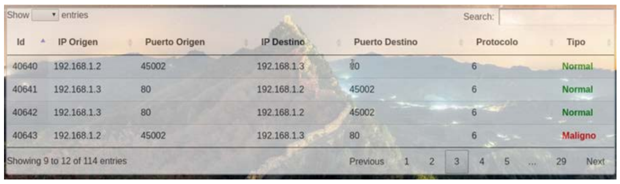


FIGURE 19. **Packet graphical section in *Dique*.**



Show entries Search:

Id	IP Origen	Puerto Origen	IP Destino	Puerto Destino	Protocolo	Tipo
40640	192.168.1.2	45002	192.168.1.3	80	6	Normal
40641	192.168.1.3	80	192.168.1.2	45002	6	Normal
40642	192.168.1.3	80	192.168.1.2	45002	6	Normal
40643	192.168.1.2	45002	192.168.1.3	80	6	Maligno

Showing 9 to 12 of 114 entries Previous 1 2 3 4 5 ... 29 Next

FIGURE 20. **Packet information section in *Dique*.**

5) INFORMATION WINDOW

It displays a table with information about the network packets that have been captured and classified (see Figure 20). This table includes the following columns:

- ID: Unique identifier of each packet. This is the first element to be stored in the database.
- Source IP address: The IP address from which the packet was sent.
- Source port: The port from which the packet was sent.
- Destination IP address: The address to which the packet was sent.
- Destination port: The port to which the packet was sent.
- Protocol: Protocol port number assigned according to IANA guidelines.
- Type: Class (malicious or normal) assigned by the classifier.

Dique is the result of integrating different Java libraries (e.g., Spring Boot, DL4J, and Pcap4J) into the trained model. This preventive system analyzes, classifies, detects, prevents, and displays, in real time, DoS attacks against a web server.

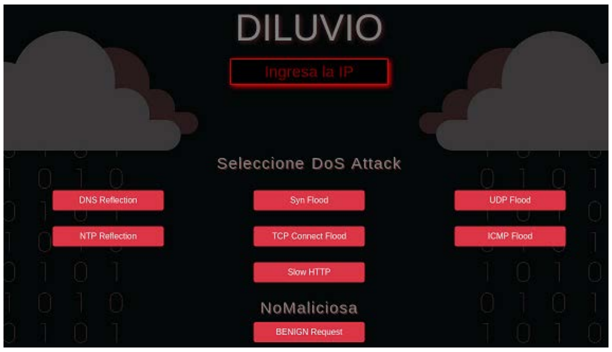


FIGURE 21. Graphical user interface of *Diluvio*.

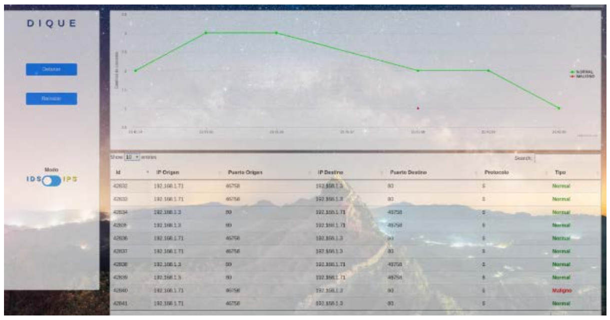
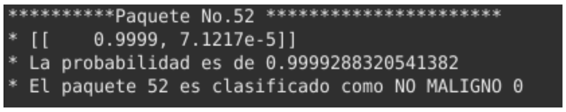


FIGURE 22. View from *Dique* of a normal HTTP request sent to the web server.



```
**********Paquete No.52 **********************
* [[    0.9999, 7.1217e-5]]
* La probabilidad es de 0.9999288320541382
* El paquete 52 es clasificado como NO MALIGNO 0
```

FIGURE 23. Classification percentage in a benign request.

The detection system employs the trained model, and the operating system's firewall uses iptables to block the source IP address of the packet. In addition, *Dique* can run on any Linux operating system with Java installed and can be accessed through the web from any device connected to it.

D. PHASE 5: CHECKING DIQUE's OPERATION

To verify and validate *Dique*'s operation, we developed a system that performs DoS attacks and named it *Diluvio*. The word *Diluvio* in Spanish (deluge in English) refers to the heavy rains that may cause dams to collapse. This system is a web application (independent of *Dique*) containing a text field in which users enter the IP address of the computer system to which they want to send the packets. Additionally, it has eight buttons. When clicked, seven of them send a different DoS attack to the IP address entered in the text field; and one of them, a benign packet. Figure 21 shows the GUI of *Diluvio*.

To check the operation of the proposed preventive system, we executed each attack in *Diluvio* and analyzed how *Dique* responded to each. The results obtained in this phase are presented below.

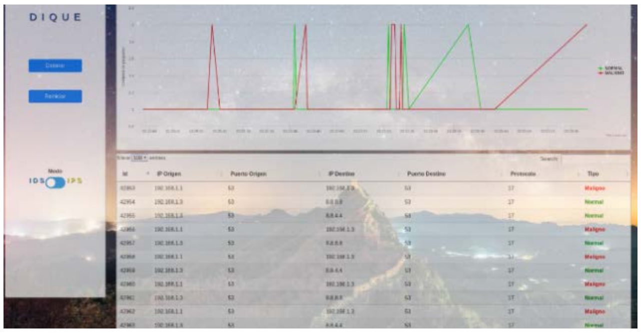
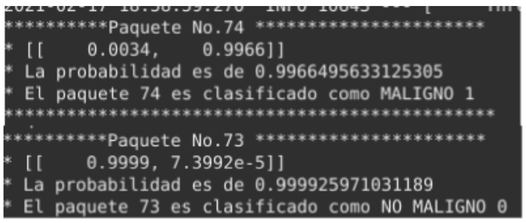


FIGURE 24. View from *Dique* of a DNS reflection attack.



```
**********Paquete No.74 **********************
* [[    0.0034,     0.9966]]
* La probabilidad es de 0.9966495633125305
* El paquete 74 es clasificado como MALIGNO 1
**********************************************
**********Paquete No.73 **********************
* [[    0.9999, 7.3992e-5]]
* La probabilidad es de 0.999925971031189
* El paquete 73 es clasificado como NO MALIGNO 0
```

FIGURE 25. Classification probability in DNS reflection attack.

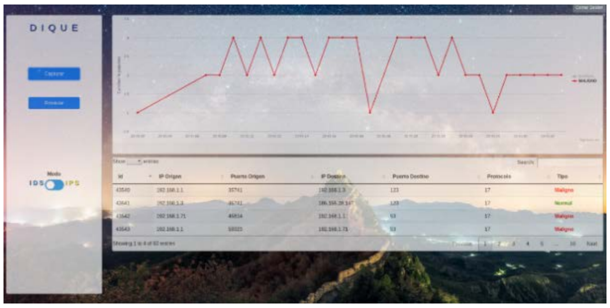


FIGURE 26. View from *Dique* of a NTP reflection attack.

1) BENIGN HTTP REQUEST

When a benign request was sent to the web server, *Dique* classified all packets as normal, as shown in Figure 22.

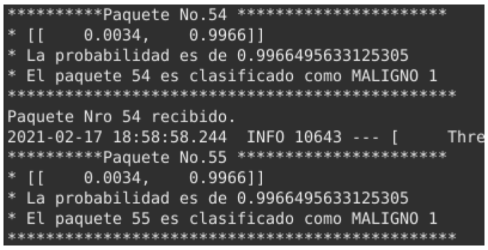
This classification is based on the output of the NN for a new packet, which is a vector containing two elements. These elements represent the classification accuracy of the two classes that the algorithm was trained to detect, which, in this case, were 99.99% for benign packets and 0.0071217% for malicious packets.

2) DNS REFLECTION ATTACK

As observed in Figure 24, in a DNS reflection attack, the algorithm classified incoming packets as malicious; and out coming packets, as benign.

Packets were classified as malicious with a probability of 99.66% and as benign with a probability of 99.99%, as shown in Figure 25.

In this case, the NN must be trained using more types of packets containing the features of this attack because the classification probability is very high for both classes.



```
**********Paquete No.54 **********************
* [[    0.0034,    0.9966]]
* La probabilidad es de 0.9966495633125305
* El paquete 54 es clasificado como MALIGNO 1
**********************************************
Paquete Nro 54 recibido.
2021-02-17 18:58:58.244  INFO 10643 --- [     Thre
**********Paquete No.55 **********************
* [[    0.0034,    0.9966]]
* La probabilidad es de 0.9966495633125305
* El paquete 55 es clasificado como MALIGNO 1
**********************************************
```

FIGURE 27. Classification probability in a NTP reflection attack.

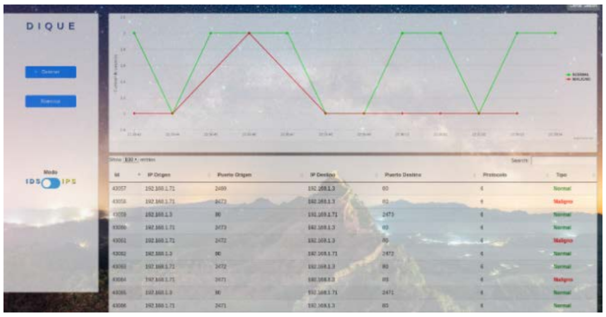
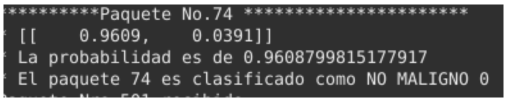


FIGURE 28. View from *Dique* of a SYN flood attack.



```
**********Paquete No.74 **********************
 [[    0.9609,    0.0391]]
 La probabilidad es de 0.9608799815177917
 El paquete 74 es clasificado como NO MALIGNO 0
```

FIGURE 29. Classification probability in a SYN flood attack.

3) NTP REFLECTION ATTACK

Packets containing a NTP reflection attack were all classified as malicious, as observed in Figure 26.

Figure 27 shows the output vector of two packets containing a NTP reflection attack. These packets were classified as malicious with a 99.66% probability and had a 0.34% probability of being normal.

According to this, the features of packets classified as NTP are relevant to the MALICIOUS class, as reported in the study conducted by the Canadian Institute for Cybersecurity, while NTP and BENIGN packets contain very different relevant features.

4) SYN FLOOD ATTACK

Packets containing this type of attack were mostly classified as benign, with a 96.09% probability, as shown in Figure 29.

The reason for the similarity in classification with Benign packets, is because there are 2 relevant common features that have these packets that are ACK Flag Count and Init_Win_bytes_forward.

5) UDP FLOOD ATTACK

When a UDP flood attack was performed, packets were all classified as malicious, as shown in Figure 30.

As observed in Figure 31, packets containing a UDP flood attack were classified as malicious with a probability of 99.99%, which demonstrates the effectiveness of the NN in classifying such attacks.

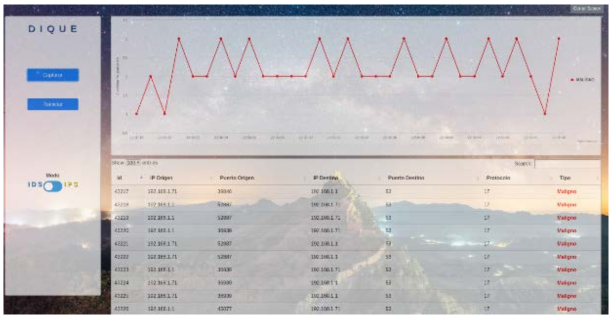
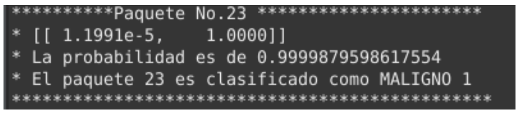


FIGURE 30. View from *Dique* of a UDP flood attack.



```
**********Paquete No.23 ***********************
* [[ 1.1991e-5,     1.0000]]
* La probabilidad es de 0.9999879598617554
* El paquete 23 es clasificado como MALIGNO 1
************************************************
```

FIGURE 31. Classification probability in UDP flood attack.

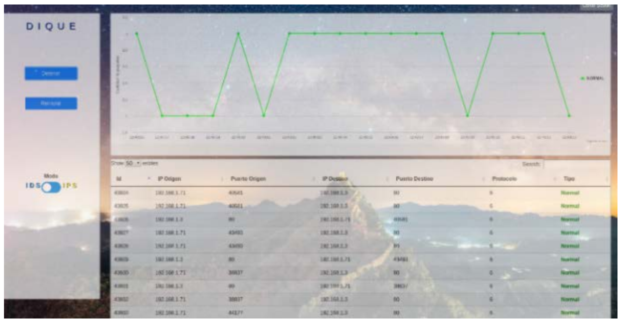


FIGURE 32. View from *Dique* of a TCP flood attack.

6) TCP FLOOD ATTACK

All packets containing a TCP flood attack were classified as benign, as observed in Figure 32.

According to Figure 33, packets containing this type of attack were all classified as benign, with a probability of 62.60% and 99.99%. This suggests that their features are similar to those of normal packets.

7) ICMP FLOOD ATTACK

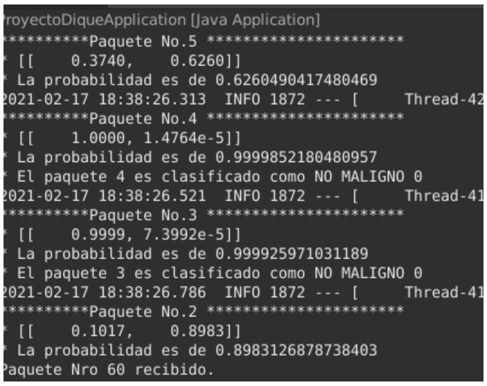
As shown in Figure 34, packets containing an ICMP flood attack were mostly classified as malicious.

According to Figure 35, packets containing an ICMP flood attack were mostly classified as malicious, with a probability of 99.61%. Although the NN was not trained to classify these packets, the aim was to show how the system would react to this type of attack, classifying as malicious in a probability of 99.66%.

8) SLOW HTTP ATTACK

As observed in Figure 36, packets containing a slow HTTP attack were classified as benign and malicious.

As shown in Figure 37, packets containing this attack were mostly classified as benign because they are quite similar to those containing a normal HTTP request (analyzed at the beginning of this section).



```
royectoDiqueApplication [Java Application]
*********Paquete No.5 **********************
* [[    0.3740,    0.6260]]
* La probabilidad es de 0.6260490417480469
2021-02-17 18:38:26.313  INFO 1872 --- [     Thread-42
*********Paquete No.4 **********************
* [[    1.0000, 1.4764e-5]]
* La probabilidad es de 0.9999852180480957
* El paquete 4 es clasificado como NO MALIGNO 0
2021-02-17 18:38:26.521  INFO 1872 --- [     Thread-41
*********Paquete No.3 **********************
* [[    0.9999, 7.3992e-5]]
* La probabilidad es de 0.999925971031189
* El paquete 3 es clasificado como NO MALIGNO 0
2021-02-17 18:38:26.786  INFO 1872 --- [     Thread-41
*********Paquete No.2 **********************
* [[    0.1017,    0.8983]]
* La probabilidad es de 0.8983126878738403
Paquete Nro 60 recibido.
```

FIGURE 33. Classification probability in TCP flood attack.

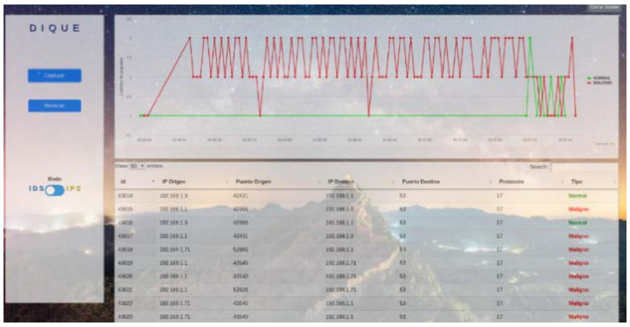
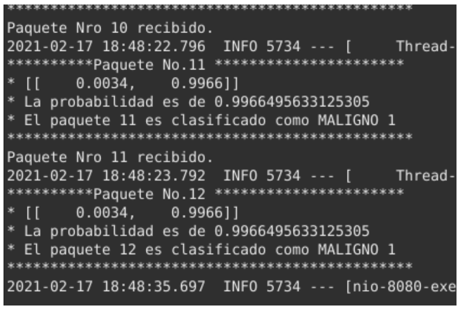


FIGURE 34. View from *Dique* of a ICMP flood attack.



```
**********************************************
Paquete Nro 10 recibido.
2021-02-17 18:48:22.796  INFO 5734 --- [     Thread-
*********Paquete No.11 **********************
* [[    0.0034,    0.9966]]
* La probabilidad es de 0.9966495633125305
* El paquete 11 es clasificado como MALIGNO 1
**********************************************
Paquete Nro 11 recibido.
2021-02-17 18:48:23.792  INFO 5734 --- [     Thread-
*********Paquete No.12 **********************
* [[    0.0034,    0.9966]]
* La probabilidad es de 0.9966495633125305
* El paquete 12 es clasificado como MALIGNO 1
**********************************************
2021-02-17 18:48:35.697  INFO 5734 --- [nio-8080-exe
```

FIGURE 35. Classification probability in ICMP flood attack.

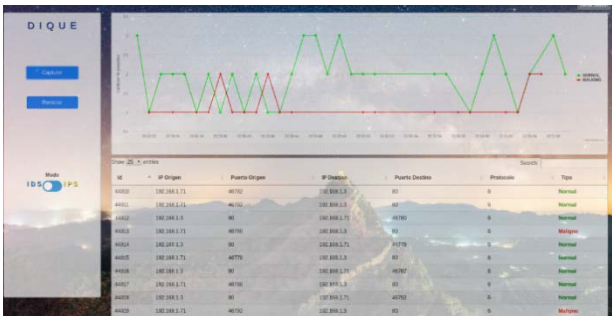
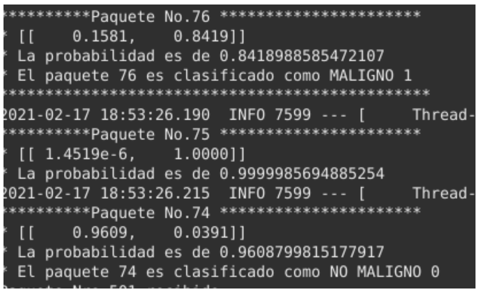


FIGURE 36. View from *Dique* of a Slow HTTP attack.

Finally, to verify the functionality of the IPS mode, the system starts applying rules automatically by combining its detection potential using DL with other tools such as its default firewall. The system, thus, begins to apply rules to block the IP addresses of packets classified as malicious, so when new DoS attacks are performed, the system will



```
*********Paquete No.76 **********************
* [[    0.1581,    0.8419]]
* La probabilidad es de 0.8418988585472107
* El paquete 76 es clasificado como MALIGNO 1
**********************************************
2021-02-17 18:53:26.190  INFO 7599 --- [     Thread-
*********Paquete No.75 **********************
* [[ 1.4519e-6,    1.0000]]
* La probabilidad es de 0.9999985694885254
2021-02-17 18:53:26.215  INFO 7599 --- [     Thread-
*********Paquete No.74 **********************
* [[    0.9609,    0.0391]]
* La probabilidad es de 0.9608799815177917
* El paquete 74 es clasificado como NO MALIGNO 0
```

FIGURE 37. Classification probability in a slow HTTP attack.

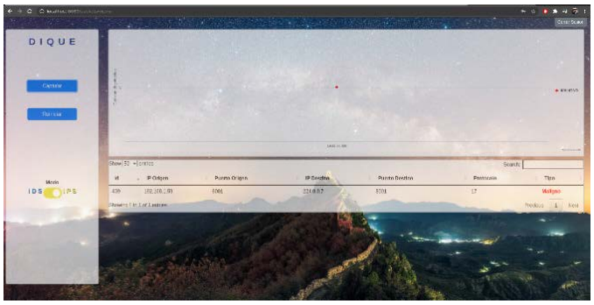


FIGURE 38. Dique with IPS mode activated.

not display new packets from that IP address, as shown in Figure 38. To prevent future DoS attacks, we applied firewall rules using the Java language when a malicious packet is detected.

Diluvio allows users to send seven types of DoS attacks, five of which are part of the training dataset and two of which are not included in such dataset. The trained model is, therefore, capable of classifying different DoS attacks that it was trained to detect. ICMP flood and slow HTTP attacks, which were not included in the training dataset, classified as malicious, with a probability above 0.84. In addition, by integrating the proposed preventive system with the operating system's firewall, its IPS mode could be validated by automatically applying rules to prevent future DoS attacks.

VI. CONCLUSION AND FUTURE WORK

The contributions that have been obtained in this research are summarized below.

Among ML and DL neural network techniques, Deep Learning is the most suitable for detecting DoS attacks, according to the SLR.

According to the criteria defined in this study, the DL algorithms of Deep Feed Forward, Recurrent Neural Network and Long Short Term Memory are the ones that report the best behavior to train models to detect DoS attacks. In this research, the DFNN algorithm was chosen to develop the training model, which, when trained with the CICDDOS2019 data set (which was adapted to recognize two classes: malicious and benign), yielded an accuracy of 0.9994, an accuracy of 0.9995, a recall of 0.999, and an F1 score of 0.9993. Which are very promising metrics, despite the hardware limitations.

One of the models reported in the literature that uses the DFNN algorithm, which has yielded results similar to those of this study, is a classification model of thirteen classes (12 DoS attacks and 1 benign) and that used the NSL- KDD for training. He obtained a precision of 0.78, a recall of 0.65, and an F1 score of 0.69 [6]. The authors who proposed such a model used ML techniques, which allowed them to compare the better performance of DL with respect to ML in detecting DoS attacks.

Regarding the two-class classification models (malicious and benign) using DL techniques, two models with similar results to this research have been reported in the literature:: (1) the first used the NSL-KDD dataset for training and obtained a precision of 0.9378 [19]. And (2) the second used the DARPA 2009 DoS dataset for training and achieved a precision of 0.9963, a precision of 0.998, a recall of 0.994, and an F1 score of 0.996 [18].

In this research, the *Dique* application is designed and built; integrating the java libraries (such as Spring Boot, DL4J, Pcap4J) and the trained model, which allows: analyze, classify, detect, prevent and show in real time, DoS attacks made to a web server. *Dique's* detection system (IDS) is carried out through the trained model (it is visualized graphically and textually) and the prevention (IPS) is executed through the IPTables of the firewall to block the source IP of the packet. The studies reported in the literature focused on analyzing the performance, but not on testing its operation and the application of its models. The *Dique* system can be deployed on any server with a Linux-type operating system that has java installed.

To validate the operation of *Dique* (and the effectiveness of the trained model), a web application called *Diluvio* was designed and developed. This application allows users to submit seven types of DoS attacks, five of which are part of the training dataset and two of which are not included in the training dataset. Verifying that the trained model is capable of classifying the DoS attacks for which it was trained to detect, in addition to the attack that is not part of the Dataset. These last attacks were recognized with a probability greater than 0.84, putting the learning of the model to the test.

Finally, although the trained models that have been proposed have achieved high-performance accuracy, this does not guarantee that, in practice, they optimally classify unknown packets entering a web server, since the recognition probability is high, but has a degree of uncertainty. In theory, a higher classification percentage is expected, if the model is trained with more data.

As future work, it is recommended:

- Develop a new DL algorithm for DoS attacks; based on DFNN or optimize the performance of existing ones.
- Improve the training of the classification models obtained in this study, so that the proposed preventive system can classify in more classes (eg, predict the type of DoS attack) with greater probability.
- Add more data to the CICDDOS2019 dataset to allow models to detect other DoS attacks and improve training models.

APPENDIX

METHOD FOR LITERATURE REVIEW

To carry out a systematic review of the literature (SLR), it is necessary to build an initial protocol, in which the methodology is structured to carry out a state of the art on the subject of interest. In this research, the protocol used is defined in Phase 1 of the Methodology session and for its construction we rely on the work of the Software Engineering Group [26] and Serna M and Serna A [27]. The phases or steps used to carry out the SLR and the quantitative results obtained are described below:

1) Definition of the research questions for the SLR: What are the deep learning methods used in the identification of denial of service attacks?

2) Specify the search terms to be entered in the database search box: Keywords are; Deep Learning, Denial of Service, Intrusion Detection System. The search terms used are; Intrusion Detection System + Denial of Service, Deep Learning + Denial of Service, and Deep Learning + Intrusion Detection System.

3) Selection of bibliographic databases: IEEE Explore, ACM Digital, Science Direct, Scopus and Cielo.

4) Define the inclusion and exclusion criteria for the pre-selection of relevant studies; Source of the document, author(s), the time window is 2009-2020 and what topic is related to this research. The result was the preselection of 157 publications.

5) *Evaluation of the quality of the research*: the criteria to evaluate it were: Theme of the work directly related to this investigation, methodology, verifiable results and conclusions. The works that met the quality evaluation criteria are 42, of which 24 are part of the bibliographic references of this work.

6) Result of the systematic review: Part of the result of the SLR is found in the Concepts and Literature Review sections of this document. In addition, as a synthesis, Table 2 was built to characterize the Deep Learning algorithms for the identification of DoS attacks, the Datasets served to train models and the reference of the study.

REFERENCES

[1] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35365–35381, 2018.

[2] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, "On the effectiveness of machine and deep learning for cyber security," presented at the 10th Int. Conf. Cyber Conflict, 2018.

[3] A. B. M. A. Al Islam and T. Sabrina, "Detection of various denial of service and Distributed Denial of Service attacks using RNN ensemble," presented at the 12th Int. Conf. Comput. Inf. Technol., Dhaka, Bangladesh, 2009.

[4] R. Priyadarshini and R. K. Barik, "A deep learning based intelligent framework to mitigate DDoS attack in fog environment," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 3, pp. 825–831, Mar. 2022.

[5] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 2046–2069, Mar. 2013.

[6] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," in *Proc. Int. Carnahan Conf. Secur. Technol.*, 2019, pp. 1–8.

[7] H. Liu, B. Lang, M. Liu, and H. Yanb, "CNN and RNN based payload classification methods for attack detection," *Knowl.-Based Syst.*, vol. 163, pp. 322–341, Jan. 2019.

[8] B. A. Ng and S. Selvakumar, "Deep radial intelligence with cumulative incarnation approach for detecting denial of service attacks," *Neurocomputing*, vol. 340, pp. 294–308, May 2019.

[9] Z. Chiba, N. Abghour, K. Moussaid, A. El Omri, and M. Rida, "Intelligent approach to build a deep neural network based IDS for cloud environment using combination of machine learning algorithms," *Comput. Secur.*, vol. 86, pp. 291–317, Sep. 2019.

[10] S. Yadav and S. Subramanian, "Detection of application layer DDoS attack by feature learning using stacked AutoEncoder," presented at the Int. Conf. Comput. Techn. Inf. Commun. Technol. (ICCTICT), 2016.

[11] A. S. Unal and M. Hacibeyoglu, "Detection of DDOS attacks in network traffic using deep learning," presented at the Int. Conf. Adv. Technol., Comput. Eng. Sci. (ICATCES), 2018.

[12] T.-Y. Kim and S.-B. Cho, "Web traffic anomaly detection using C-LSTM neural networks," *Expert Syst. Appl.*, vol. 106, pp. 66–76, Sep. 2018.

[13] A. Thilina, S. Attanayake, S. Samarakoon, D. Nawodya, L. Rupasinghe, N. Pathirage, T. Edirisinghe, and K. Krishnadeva, "Intruder detection using deep learning and association rule mining," presented at the IEEE Int. Conf. Comput. Inf. Technol., 2016.

[14] S. M. Kasongo and Y. Sun, "A deep long short-term memory based classifier for wireless intrusion detection system," *ICT Exp.*, vol. 6, no. 2, pp. 98–103, Jun. 2020.

[15] C. Xu, J. Shen, X. Du, and F. Zhang, "An intrusion detection system using a deep neural network with gated recurrent units," *IEEE Access*, vol. 6, pp. 48697–48707, 2018.

[16] Y. Imamverdiyev and F. Abdullayeva, "Deep learning method for denial of service attack detection based on restricted Boltzmann machine," *Big Data*, vol. 6, no. 2, pp. 159–169, Jun. 2018.

[17] N. Chockwanich and V. Visoottiviseth, "Intrusion detection by deep learning with TensorFlow," presented at the 21st Int. Conf. Adv. Commun. Technol. (ICACT), 2019.

[18] P. Khuphiran, P. Leelaprute, P. Uthayopas, K. Ichikawa, and W. Watanakeesuntorn, "Performance comparison of machine learning models for DDoS attacks detection," presented at the 22nd Int. Comput. Sci. Eng. Conf. (ICSEC), 2018.

[19] K. Amarasinghe, K. Kenney, and M. Manic, "Toward explainable deep neural network based anomaly detection," presented at the 11th Int. Conf. Hum. Syst. Interact. (HSI), 2018.

[20] N. G. B. Amma and S. Subramanian, "VCDeepFL: Vector convolutional deep feature learning approach for identification of known and unknown denial of service attacks," presented at the TENCON-IEEE Region Conf., 2018.

[21] X. Yuan, C. Li, and X. Li, "DeepDefense: Identifying DDoS attack via deep learning," in *Proc. IEEE Int. Conf. Smart Comput.*, May 2017, pp. 1–8.

[22] M. Z. Alom and T. M. Taha, "Network intrusion detection for cyber security using unsupervised deep learning approaches," in *Proc. IEEE Nat. Aerosp. Electron. Conf.*, Jun. 2017, pp. 63–69.

[23] M. Siracusano, S. Shiaeles, and B. Ghita, "Detection of LDDoS attacks based on TCP connection parameters," in *Proc. Global Inf. Infrastruct. Netw. Symp. (GIIS)*, Oct. 2018, p. 6.

[24] CloudFlare. *CloudFlare NTP Attack*. Accessed: Oct. 4, 2020. [Online]. Available: https://www.cloudflare.com/learning/ddos/ntp-amplification-ddos-attack/

[25] ResearchGate. *Ratio Training Set and Validation Set*. Accessed: Oct. 4, 2020. [Online]. Available: https://www.researchgate.net/post/Is-there-an-ideal-ratio-between-a-training-set-and-validation-set-Which-trade-off-would-you-suggest

[26] *Guidelines for Performing Systematic Literature Reviews in Software Engineering*, EBSE Technical Report, Software Engineering Group, Belvoir, VA, USA, 2007.

[27] M. E. Serna and A. A. Serna, "Is it in crisis engineering in the world?: A literature review," *Revista Facultad de Ingeniería Universidad de Antioquia*, vol. 66, pp. 199–208, Mar. 2013.

JUAN FERNANDO CAÑOLA GARCIA received the master's degree in computer security, systems engineer and information systems technician from the Instituto Tecnológico Metropolitano (ITM), Colombia. In 2016, he was an ITM Young Researcher. He is currently an IT Security Specialist at Grupo Éxito S.A. He is a member of the IT Systems Security research hotbed.

GABRIEL ENRIQUE TABORDA BLANDON received the master's degree in computer security from the International University of La Rioja, Spain, and the master's degree in computer science from Atlantic International University, USA. He is currently pursuing the Ph.D. degree with Real World Multiversity Complex Thinking Edgar Morin, Mexico. From 2015 to 2016, he was a Coordinator of the master's degree in computer security. Since mid-2006, he has been a Research Professor in the area of software engineering and computer security. Since 1994, he has been a Professor in the area of systems at the Higher Education. He is currently linked to the Metropolitan Technological Institute (OTC Teacher) and the University of Antioquia (Teacher). Since 2016, he has been with the Leading Line of Research in computer science and computer science of the Systems Security Research Seedbed, since 2014. He is the director and the jury of master's and degree projects. He is an evaluator of research projects and research articles. He has participant in several research projects in the area of knowledge of information systems.