IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 9, 2021, accepted July 25, 2021, date of publication July 30, 2021, date of current version August 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3101650

SDN-Based Architecture for Transport and Application Layer DDoS Attack Detection by Using Machine and Deep Learning

NOE MARCELO YUNGAICELA-NAULA, CESAR VARGAS-ROSALES, (Senior Member, IEEE), AND JESUS ARTURO PEREZ-DIAZ

School of Engineering and Sciences, Tecnologico de Monterrey, Monterrey 64849, Mexico

Corresponding author: Noe Marcelo Yungaicela-Naula (a00821711@itesm.mx)

This work was supported in part by the Joint Seed Funding Program from The University of Texas at San Antonio and the Tecnologico de Monterrey "UTSA and ITESM Seed Funding Program," in part by the Project "Red temática Ciencia y Tecnología para el desarrollo (CYTED)" through the Ibero-American Science and Technology Program for Development CYTED under Grant 519RT0580, in part by the 2020 Seed Fund Award from the Tecnologico de Monterrey & the Center for Information Technology Research in the Interest of Society (CITRIS) and the Banatao Institute, University of California, in part by the School of Engineering and Sciences, Tecnologico de Monterrey, and in part by the Telecommunications Research Group.

ABSTRACT Distributed Denial of Service (DDoS) attacks represent the most common and critical attacks targeting conventional and new generation networks, such as the Internet of Things (IoT), cloud computing, and fifth-generation (5G) communication networks. In recent years, DDoS attacks have become not only massive but also sophisticated. Software-Defined Networking (SDN) technology has demonstrated effectiveness in counter-measuring complex attacks since it provides flexibility on global network monitoring and inline network configuration. Although several works have proposed to detect DDoS attacks, most of them did not use up-to-date datasets that contain the newest threats. Furthermore, only a few previous works assessed their solutions using simulated scenarios, easing the migration to production networks. This document presents the implementation of a modular and flexible SDN-based architecture to detect transport and application layer DDoS attacks using multiple Machine Learning (ML) and Deep Learning (DL) models. Exploring diverse ML/DL methods allowed us to resolve which methods perform better under different attack types and conditions. We tested the ML/DL models using two up-to-date security datasets, namely CICDoS2017 and CICDDoS2019 datasets, and they showed accuracy above 99% on classifying unseen traffic (testing set). We also deployed a simulated environment using the network emulator Mininet and the Open Network Operating System (ONOS) SDN controller. In this experimental setup, we demonstrated high detection rates, above 98% for transport DDoS attacks and up to 95% for application-layer DDoS attacks.

INDEX TERMS Software defined networking, deep learning, machine learning, DDoS attack, transport layer, application layer, slow-rate attacks.

I. INTRODUCTION

The Denial of Service and Distributed Denial of Service (DoS/DDoS) attacks continue to be the most frequent and worst threats targeting conventional and new generation network environments, such as Internet of Things (IoT) [1], cloud computing [2], and fifth-generation (5G) communication networks [3]. According to the Kaspersky Q4 2020 DDoS attacks report [4], the number of DDoS attacks in 2020 increased three-fold compared to 2019.

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

DoS/DDoS attacks have not only become more frequent and critical but also smarter over time. Initially, transport layer DDoS threats, such as TCP-SYN, UDP, and network layer like ICMP flooding, were the most common threats to networks. As the state-of-the-art detection techniques, such as Machine Learning (ML) and Deep Learning (DL), became capable of detecting these threats, more complex and specialized DDoS attacks appeared, namely, application-layer attacks. DoS/DDoS application-layer attacks are more sophisticated and dedicated threats that affect the servers' resources. Therefore, conventional attack detection techniques that use packet-level information result ineffective [5].

For detection of DoS/DDoS attacks, it is relevant to use information from network traffic flows to design a network-based Intrusion Detection System (IDS), using innovative networking technologies such as the Software-Defined Networking (SDN). SDN is a modern networking paradigm that decouples the control plane (CP) from network devices. This technology operates differently from the conventional network architecture. While the forwarding engine is located in the switches, all network control functions, such as traffic monitoring and routing, take place in a centralized software-based controller [6]. This technology allows us to dynamically reconfigure routing rules in the network devices such as switches and routers. Such capabilities can enable the implementation of in-line and network-based attack detection/mitigation mechanisms.

Previous proposals have demonstrated that SDN architecture is suitable to deploy ML algorithms to detect DDoS attacks, thanks to the higher computational resources in the CP and the global visibility of the network [7]–[10]. However, most of the proposals did not use up-to-date security datasets [11]–[13], which did not permit the study of the most recent DoS/DDoS attacks. Moreover, most of these proposals have been evaluated only in an offline manner, using benchmark or generated datasets [14]–[18]. Although offline assessment could be a valid approach to establish the best DDoS detection, it could not fit the demands for characterization, detection, and practical evaluation of the network security solutions using SDN before their deployment in production environments. An implementation on a real or emulated network will provide us a better analysis of a proposed solution. In addition, previous works focused their studies on a single type of DoS/DDoS attack and a limited number of intelligent-based detection mechanisms.

In this paper, we present a modular and flexible SDN-based architecture to detect DoS/DDoS attacks by using artificial intelligence methods. The modularity property helps us separately modifying and improving each component in the architecture, that is, the flow collector, the preprocessing, the detection, and the flow manager modules, independently. Also, this design provided us flexibility in experimenting with various ML/DL detection techniques to countermeasure different DoS/DDoS attacks. On the one hand, we explored the two most frequent and hazardous transport layer DDoS attacks, namely, SYN Flood and UDP flood [4]. We also explored the seven most common and critical application-layer attacks, including high-volume and slow-rate DoS/DDoS attacks, which are harder to detect [5]. On the other hand, we evaluated three ML methods, named, support vector machine (SVM), random forest (RF), K-nearest neighbor (K-NN), and four DL mechanisms, namely, multilayer perceptron (MLP), convolutional neural network (CNN), gated recurrent units (GRU), and long short-term memory (LSTM) neural network, to detect DoS/DDoS attacks. We evaluated and compared the performance and complexity of the ML/DL techniques. Exploring several ML/DL techniques allowed us to resolve which methods perform better under different attack types and conditions.

Additionally, we used two up-to-date datasets, the CIC-DoS2017 and the CICDDoS2019 datasets [19] that contain application-layer and transport-layer attacks respectively for training diverse ML/DL models. The use of up-to-date datasets permits the study of the most recent DoS/DDoS attacks. Afterward, the models were deployed in an emulated testbed, using Mininet and an ONOS controller for real-time testing. To assess the robustness of each ML/DL technique, we experimented with different attack conditions. We varied parameters such as the traffic rate and connection rate of each DDoS attack.

This research presents new ideas with remarkable improvements over the work previously published in [20]. Unlike this previous work, we included a comprehensive performance and robustness assessment of several new ML models. Also, we expanded the scope of the study by including new attack types, new datasets, and the most promising DL models. The accuracy of the ML and DL models is highly improved, and we added a comprehensive evaluation of the models in a simulated environment.

In summary, the contributions of this work are:

1) A modular and flexible SDN-based architecture to detect diverse transport-layer and application-layer DoS/DDoS attacks.
2) A ranking of the best ML/DL models against different DDoS attacks included in up-to-date security datasets to help in the creation of new generation firewalls.
3) Real-time performance and robustness evaluation of the proposed solution in a testbed deployed using the network emulator Mininet and ONOS controller. Only a few previous works assessed their solutions using real or simulated environments. This evaluation demonstrates that our solution is ready to be implemented in production environments since we obtained high detection rates for high-volume and slow-rate DDoS attacks.

A. EXISTING WORKS

Recently, several works have been developed to detect transport layer and application-layer DoS/DDoS attacks using intelligent techniques and SDN technology. Next, we provide a brief review of recent works related to our study to explain the differences with our approach.

The transport layer threats TCP-SYN, UDP, and the network layer ICMP are the most explored DDoS attacks. The work in [21] uses the LSTM method and fuzzy logic (FL) to detect and mitigate DDoS and Port Scan attacks in SDN environments. The used features were the bit/s, packet/s, source IP entropy, destination IP entropy, source Port entropy, and the destination Port entropy. The authors evaluated their proposal in two ways: 1) Using a testbed with Mininet network emulator and Floodlight controller and 2) using the CICD-DoS2019 dataset. They compared their solution with SVM, KNN, MLP, LSTM-2, and Particle Swarm Optimization

Digital Signature (PSO-DS) techniques. Their proposal presented the best results, achieving 99.74% of accuracy and 0.25% of false-positive rate (FPR). On the mitigation part, the abnormal packets discarded were above 99.88%. This work focused only on high-volume attacks, though.

The work in [11] proposed to combine supervised and unsupervised ML techniques to detect DDoS attacks, including Probe, R2L, and U2R attacks. The unsupervised method reduces the irrelevant and noisy traffic data. The supervised technique minimizes the FPRs of the unsupervised method to classify the DDoS traffic accurately. The evaluation of this approach used the NSL-KDD, the UNB ISCX IDS 2012, and the UNSW-NB15 datasets. The authors achieved a classification accuracy of 98.23%. This work did not use up-to-date datasets and was limited in the number of ML/DL methods and attack types explored.

The Intrusion Prevention System (IPS) proposed in [13] uses the KNN model to detect TCP-SYN and ICMP flooding attacks. This solution was evaluated using the CAIDA 2007 dataset and a dataset generated using an SDN-based testbed. The scheme has a mitigation efficiency of 99.4% for TCP-SYN attacks and 98.9% for ICMP attacks. The work in [12] proposed a source-based defense, using an LSTM model at the packet level. The authors used 192 packet features, and they trained the model using the ISCX 2012 IDS and CTU-13 datasets. Moreover, the authors deployed a laboratory-based testbed with a Floodlight controller to generate a dataset for testing the model and achieved an accuracy of 98.88%. The authors in [22] explored the RF, decision tree, and logistic regression methods to detect DDoS in two datasets: KDD Cup 99 dataset and their own generated real-time dataset. The RF technique provided the best performance with an accuracy of 99.21% and an FPR of 0.3%. The authors in [23] evaluated the LSTM, CNN, and MLP models to detect DDoS attacks using the KDD Cup 99, NSL KDD, and their own generated real-time dataset. The LSTM approach demonstrated the best accuracy, achieving up to 99.09% and FPR of 0.029. These studies explored only high-volume DDoS threats with outdated security datasets and did not perform online assessments of their proposed solutions in a testbed.

Application-layer attacks are harder to detect compared to transport-layer attacks since their traffic patterns are like those of legitimate clients. Therefore, more complex detection mechanisms are needed. The Q-MIND framework proposed in [24] defeats stealthy DoS attacks in SDN environments. This solution mitigates slow-rate DoS attacks using a reinforcement learning reward-based policy to delete all flows stemming from an identified attacker IP. Then, the solution installs a flow rule to block sources for a certain period. The framework was tested using hosts connected to an OpenFlow switch and an ONOS controller, achieving attack detection performance of 99.5%. A disadvantage of this scheme is that every time an IP is detected, a complete game is executed, which could slow down the execution in large networks.

The authors in [20] proposed a flexible architecture to detect and mitigate slow-rate DoS attacks, such as slow body, slow headers, and slow read attacks, using ML methods. In this scheme, an IDS detects attacks, and an intrusion prevention system (IPS) mitigates them. They used the CICDoS2017 dataset to train the ML models. To mitigate attacks, they proposed to create a blacklist and to increase its probability until it is certain that it is not legitimate traffic, thus mitigating it. The proposed architecture was tested in a simulated environment, using Mininet and an ONOS controller. However, the detection accuracy is not very high (95%) due to the complexity of identifying low-rate DDoS attacks.

The work in [14] proposed an LSTM enabled framework for slow-rate DDoS detection. They used the CICIDS2017 dataset that includes the slow body, slow header, and the DoS Get attacks to test the framework. The authors used about 15 flow parameters to train the LSTM model and compared the proposed mechanism with the Decision Tree, ANN, and SVM methods. The model performance achieved with training was of recall above 99.9%, precision above 99.9%, and F1 score above 99.9%. Testing the model on a different dataset reduced the precision, recall, and F1 score to 70%, 80%, and 80%, respectively. The work in [15] used an MLP neural network to detect HTTP-based slow-rate attacks. The neural network analyzes the requesting behavior of the botnet in terms of network flow characteristics. This approach used 14 flow characteristics and used three different datasets to test the proposed mechanism, namely, the WIDE dataset, CTU dataset, and an own generated MANET dataset. The detection rate achieved was above 95% on separating legitimate flows from attack flows, but 80% in detecting specifically the attack types. These studies did not perform a testbed-based inline model assessment to evaluate the models.

Based on our brief review, we can conclude that many previous works have not used up-to-date datasets, as we considered in this work. In addition, most works studied transport or application layer attacks separately. We propose to evaluate different ML/DL techniques on both transport and application layer DoS/DDoS attacks. This assessment will allow us to define the accuracy of each ML/DL method under different attacks, which increases the scope of the functionality of our proposed architecture. Moreover, many previous approaches performed an offline analysis of their proposals, using traffic captured from testbeds. The implementation of a security solution in a real or emulated network brings several issues and challenges, such as the limited capability of the SDN controller (memory and processing) to manage real-time flow collection and classification and the decrease in the detection performance of the models. Our proposal uses an emulated network to assess the models' performance in real-time. Finally, we evaluate the robustness and the time and space complexity of each ML/DL method to detect different attacks and vary the attack conditions. This comprehensive assessment has not been reported in previous works.

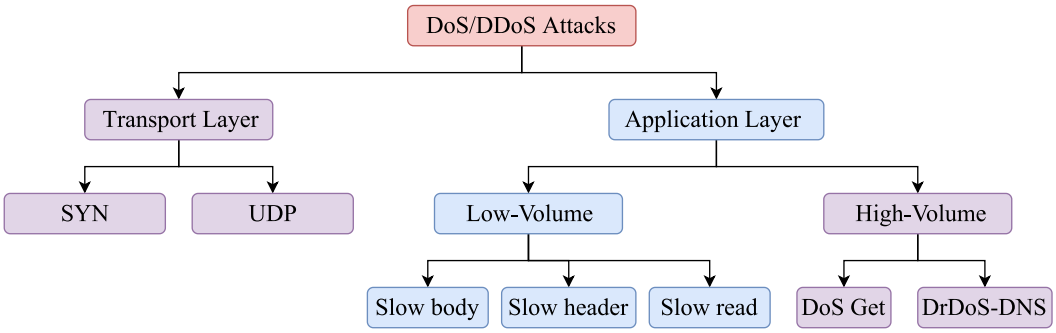

FIGURE 1. Taxonomy of the studied DoS/DDoS attacks.

The remaining content of this work is organized as follows. Section II describes the DoS/DDoS attacks targeted in this work. Section III explains our SDN-based architecture proposed to detect DoS/DDoS attacks. Next, different modules of the proposed architecture are presented, including the Flow Collector module in Section IV, and the Detection and Flow Manager modules in Section V. Experimental results and complexity analysis using two up-to-date datasets are presented in Section VI. Experimental results analysis using a simulation-based testbed are presented in Section VII. Finally, Sections VIII and IX draw the discussions, conclusion and future work.

II. DoS/DDoS ATTACKS

DoS/DDoS attacks are the most frequent and critical threats that use protocols of different layers of the OSI model. Transport layer DDoS attacks have been the most widely studied threats, and still, they cause several damages to current network environments. These attacks are generally high-volume and expensive to deploy. However, they can be detected using state-of-the-art artificial intelligence techniques such as ML and DL.

Application-layer DoS/DDoS attacks are more recent threats targeting the resources of specific applications running on the servers. They can be low-volume or high-volume attacks, and the former is the most complex to detect. Additionally, the creation of these threats requires a high level of expertise, but its deployment consumes low computational and bandwidth resources [5].

Fig. 1 shows the taxonomy of the attacks studied in this work. Next, the description of each type of attack is presented.

A. TRANSPORT LAYER DDoS ATTACKS

These kinds of threats use protocols such as TCP and UDP to deploy attacks. TCP-SYN flood and UDP flood are well-known attacks in the literature.

1) SYN

The SYN flood attack is often generated by botnets that consume the server resources by exploiting TCP-three-way handshake [25]. This attack initiates by sending repeated SYN packets to the target server at a high rate. However, the attackers never end the TCP-three-way handshake, which causes server crashes/malfunctions or reboots. Besides, the bandwidth of the network gets exhausted, and network performance degrades.

2) UDP

The UDP flood attack initiates on a remote host by sending many UDP packets to random ports on the target machine at a very high rate [25], which results in the exhaustion of available network bandwidth, the system crashes, and its performance degrades.

B. APPLICATION-LAYER DoS/DDoS ATTACKS

These kinds of threats target the resources of the applications in a server.

1) HIGH-VOLUME APPLICATION-LAYER ATTACKS

These threats are similar to traditional DoS/DDoS attacks. They use high-volume application-layer requests, such as HTTP GET and HTTP POST requests transmitted to a victim. However, these threats target the resources of the server and require smaller volumes of malicious traffic. In this work, we study the DoS GET attack which uses what appears to be legitimate HTTP GET requests to attack a web server or application. This attack can use multiple coordinated computers to send numerous GET requests of some asset from a targeted server, such as images, files, etc. When the target is inundated with incoming requests and responses, denial-of-service will occur to additional requests from legitimate traffic sources [26].

Distributed and reflected Domain Name Server (DrDNS) amplification threat is also studied in this work. In a DrDNS attack, bots are used to make UDP-based requests to open DNS resolvers with a spoofed IP address, which has been changed to the real source IP address of the targeted victim. Since UDP is connectionless, with a small request, very large answers can be obtained. Besides, because the real IP address is hidden, it becomes a reflection attack [27].

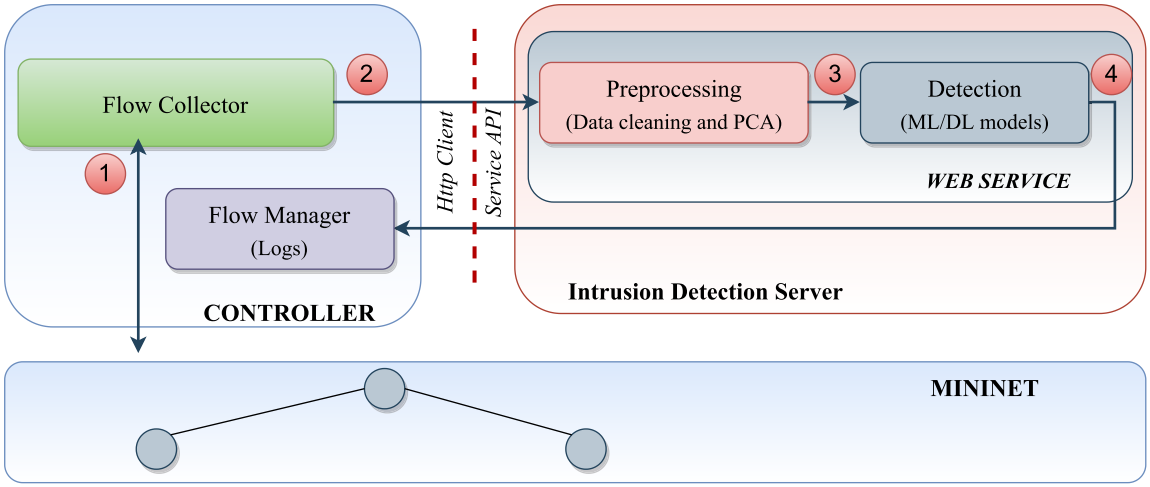

FIGURE 2. Proposed architecture for intelligent SDN-based DoS/DDoS attacks detection.

2) LOW-VOLUME APPLICATION-LAYER ATTACKS

Low-volume application-layer attacks are characterized by the small amount of traffic needed to beat a victim. These threats can be classified into three categories: i) Low-rate attacks that send traffic in periodic short-time pulses, ii) slow-rate attacks that exploit timing parameters on the server's side by sending or receiving traffic slower than expected, and iii) one-shot attacks that damage victims in a single request [15].

In this work, we experiment with slow-rate attacks, which can be seen in two variations: Slow send and slow read attacks. Slow send attacks aim to collapse the server resources by slowly sending legitimate but incomplete requests causing the victim server to reserve resources for open connections waiting for their completion. These threats can be implemented with the focus on the header or body request, namely, slow header and slow body attacks. Slow read attackers start with a legitimate request to a victim server followed by a slow consumption of the response sent by the victim. Therefore, the attacker can maintain multiple active and concurrent connections with a server.

We selected different kinds of threats to explore most of the combinations of DoS/DDoS attacks. We study transport/application layer attacks of high- and low-volume, as established in Table 1.

TABLE 1. Characteristics of the DoS/DDoS attacks studied in this work.

Layer/Rate	Low	High
Transport		SYN flood, UDP flood
Application	Slow body, Slow header, Slow read	HTTP Get flood

III. PROPOSED ARCHITECTURE

Fig. 2 shows the architecture proposed to detect transport and application layer DoS/DDoS attacks. This architecture consists of four modules: Flow Collector, Preprocessing, Detection, and Flow Manager. We based the design of the flow collector module on the CICFlowMeter application [28], [29]. This module gathers the data packets from the network (1), creates the flows, and forwards them to the preprocessing module (2). The preprocessing module reduces the dimension of the flow characteristics. For that, the application performs data cleaning and principal component analysis (PCA). The Detection module employs a trained model to classify the preprocessed input flows (3) as suspicious or benign. The Flow Manager module sends the information of suspicious flows to the controller (4). This module creates and visualizes logs of the flows' classification received from the Detection module.

The Flow Collector and Flow Manager modules are embedded in an application and installed in the controller. A web service application contains the Preprocessing and Detection modules. This web service can be deployed in a different physical or virtual server. We call this server Intrusion Detection Server (IDS). This separation will alleviate the processing overhead in the controller. Additionally, this architecture provides flexibility since it permits improvements or modifications of the IDS without affecting the rest of the architecture.

We used the network emulator Mininet [30] and the ONOS [31] controller to experiment with the proposed architecture. Mininet is an SDN-based network emulator widely used in academia. A relevant advantage of Mininet is the vastly available documentation. Also, we can adapt this emulator with any open-source SDN controller. ONOS is a Java-based controller developed by the Open Networking Foundation (ONF) that supports complex and realistic SDN application development. This controller is easy to use and has noteworthy CLI, REST, and GUI interfaces.

We detail the implementation of each module of the proposed architecture in the following sections.

IV. FLOW COLLECTOR MODULE

The available protocol OF v 1.3 in Mininet allows us to access limited statistics variables. Therefore, we focused on developing a more complete and inline flow capturing application, combining the flow processors Flowbag [32] and CICFlowMeter [28].

CICFlowMeter, developed by the Canadian Institute for Cybersecurity (CIC), is a network traffic flow generator and analyzer. This tool generates bidirectional flows, where the first packet determines the forward (source to destination) and backward (destination to source) directions. Hence, 76 statistical network traffic features can be calculated separately in the forward and backward directions. On the other hand, Flowbag was developed in *Go* and allows us to calculate flow statistics from a given packet capture file. Flowbag was designed with offline processing as the primary focus. Therefore, the application requires adaptation for online packet processing. The combination of the features of the CICFlowMeter and Flowbag applications provided us a powerful flow collector module. The characteristics of this module are shown in Table 2. The Flow Collector module outputs the flows with a five-tuple identification (ID): {Source IP, Source Port, Destination IP, Destination Port, Protocol} and 76 features. Details of these features can be found in [28].

TABLE 2. Characteristics of the flow collector module.

Characterist	Detail	Observation
Flow Characteristics	Five Flow ID parameters and 76 Flow characteristics, e.g., Flow duration.	Flow ID: {Source IP, Source Port, Destination IP, Destination Port, Protocol}
Flows building	Bidirectional TCP and UDP Flows.	The first packet defines the flow's forward direction.
Configurable	Flow time out: 600 s for TCP and UDP, and Idle time out: 1 s	Characterization according to [33].
Language	Java	Deployed on the ONOS controller.

V. PREPROCESSING, DETECTION, AND FLOW MANAGER MODULES

A. PREPROCESSING MODULE

The Preprocessing module receives a flow from the Flow Collector module with an ID of five parameters and 76 characteristics. The use of all these flows' features will increase the training complexity and inline prediction delays of the intelligent mechanisms installed in the Detection module. Therefore, data cleaning and dimensionality reduction were performed. We explain in detail the preprocessing methodology in Section VI-B, using two specific datasets.

B. DETECTION MODULE

The Detection module uses the preprocessed information to classify the flows as benign or attacks using the trained ML/DL models.

Fig. 3 shows the taxonomy of the classification models used in this study. We used three ML models, namely, k-nearest neighbor (KNN), support vector machine (SVM), and random forest (RF), and four DL models, namely, multilayer perceptron (MLP), convolutional neural network (CNN), gated recurrent units (GRU), and long short-term memory (LSTM) neural network. All models were selected based on related works, as they have shown high performance in detecting DDoS attacks. Next, the relevant characteristics of each model are briefly explained.

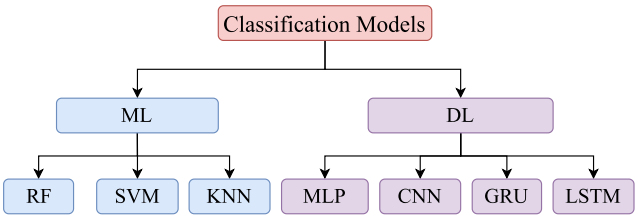

FIGURE 3. Taxonomy of the classification models studied in this work.

1) RF

Random Forest is an ensemble classifier with low classification error that consists of many individual decision trees uncorrelated, which operate as a single set with the power of handling large data sets, outliers, and noise in the data. Each tree produces a classification that helps to achieve greater accuracy. This method could present slow real-time prediction because of the formation of many trees. Nevertheless, RF has shown high performance in detecting attacks on networks [20], [34], [35].

2) SVM

SVM is a supervised learning model able to perform nonlinear classification. This model uses what is called the kernel, which implicitly maps its inputs into high-dimensional feature spaces. SVM can detect intrusions with limited training data [34], [36].

3) KNN

KNN is a supervised ML algorithm used for solving classification problems. This model determines the class of new data by looking at its nearest K neighbors. KNN uses different distance functions, namely, Manhattan, Minkowski, and Euclidean, to calculate the distance between two data. KNN has shown high performance on attack detection applications [37].

4) MLP

MLP consists of multiple hidden layers and has powerful learning and mapping capabilities than a single-layer neural network. This model consists of three or more layers of nodes: An input layer, one or more hidden layers, and an output layer. The MLP uses back-propagation to learn the weights and biases of each layer and has shown high effectiveness in the area of network attack detection [15].

5) CNN

CNN is usually used for object recognition and detection in images, labeling, and others. Nevertheless, CNN has also

shown effectiveness on attack detection [38]. The conventional form of CNN consists of an input layer, a convolutional layer, a pooling layer, a fully connected layer, and an output layer. The training process can use forward- and back-propagation for learning the convolutional filters, weights, and biases of the layers.

6) GRU

GRU is a type of Recurrent Neural Network (RNN). An RNN performs the same task for every element of a sequence. The outputs are dependent on the previous computations. An RNN is trained using the back-propagation through time (BPTT) algorithm. However, BPTT for the RNN is usually hard to use due to the problem known as vanishing/exploding gradient. GRU networks can tackle this problem. GRUs have what is called an update gate and a reset gate. Using these two vectors, the model refines outputs by controlling the flow of information through the layers. GRU can retain information over a period. Thus, this model represents a system with memory. In the attack detection applications, GRU neural networks have demonstrated high-performance [39].

7) LSTM

LSTM model is another type of RNN capable of tackling the vanishing/exploding gradient. Its long-term memory capability could better capture the DDoS behavior in time than other methods that only include short-term memory, such as the MLP. The classification error is back-propagated in the network to update weights of the connections among the input, hidden, and output layers. LSTM has shown high effectiveness in detecting complex attacks to networks [14].

C. FLOW MANAGER MODULE

This module is in charge of saving and visualizing the logs of the predictions received from the Detection module. As this work is part of a broader project, the next step will be to complement this module with a mitigation strategy to countermeasure the DoS/DDoS attacks on the network, based on the predictions obtained using the ML/DL models.

VI. MODEL TRAINING AND TESTING USING DATASETS

We trained the ML/DL models using two public datasets that contain our targeted DoS/DDoS attacks. This section describes the offline flows capturing from the datasets, data preprocessing, model hyperparameters tuning, and the models training and testing using datasets.

A. DATASETS

We worked with two public datasets, CICDoS2017 [26] and CICDDoS2019 [29], from the Canadian Institute for Cybersecurity (CIC) [19]. These datasets contain pcap files for different DoS/DDoS attacks. CICDoS2017 dataset carries 24 hours of network traffic simulated on a testbed of application-layer DoS attacks intermixed with attack-free traces. This dataset presents six DoS/DDoS application-layer attacks: slow-send body, slow-send header, slow read, DoS HTTP Get flooding, improved DoS HTTP Get flooding, and DDoS HTTP Get flooding. CICDDoS2019 dataset consists of two capturing days of the attacks for training and testing. This dataset contains pcap files of the most up-to-date common DDoS attacks, including transport and application-layer attacks. We filtered (using Wireshark) from this dataset the three attacks targeted in our study: TCP-SYN flood attack, UDP flood attack, and DrDNS attack.

We used a flow capturing application, whose implementation is based on CICflowMeter and Flowbag applications, to process the pcap files. From this processing, we obtained comma-separated values (CSV) files containing the captured flows. Then, we labeled these flows according to the information presented in [26] and [29]. These works outline the details and underlying principles of the datasets CICDoS2017 and CICDDoS2019, respectively.

We divided the flows captured from the two datasets into training and testing subsets, according to the distributions presented in Table 3. The CICDoS2017 dataset contains 26 attack events and six different attacks. For them, we used 17 events for model training and nine events for testing. Note that we included at least one event for each attack type in the testing set. In terms of the number of flows, we used 65% for training and 35% for testing.

TABLE 3. Datasets' statistics.

Dataset	Events/Attacks	Training events (flows)	Testing events (flows)
CICDoS 2017	Slow-send body	6 (3025)	2 (1002)
	Slow read	1 (2133)	1 (2133)
	Slow-send header	4 (5436)	3 (3952)
	DoS GET	3 (2304)	1 (766)
	Improved GET	2 (1222)	1 (532)
	DDoS GET	1 (5497)	1 (5015)
	Benign	1 (83075)	1 (42473)
	TOTAL FLOWS	102692 (65%)	55873 (35%)
CICDDoS 2019	SYN	1 (886817)	1 (989944)
	UDP	1 (989247)	1 (1022869)
	DrDNS	1 (22982)	1 (4902)
	Benign	1 (92720)	1 (41853)
	TOTAL FLOWS	1991766 (49%)	2059568 (51%)

For the CICDDoS2019 dataset, we captured SYN, UDP, and DrDNS attack events from the training and testing days. In this dataset, a small number of benign traffic is present, compared to attack traffic, which is a problem in the training phase since benign events are under-represented. Therefore, we added most of the benign traffic captured from the CICDoS2017 dataset to the CICDoS2019 dataset. In terms of the number of flows, we used 49% of total flows for training and 51% for testing. This distribution is due to the balance of data presented in the original dataset CICDDoS2019.

B. PREPROCESSING

Before training the ML/DL models, preprocessing is necessary to improve the data quality. Fig. 4 shows the data preprocessing methodology executed in this work. First, we performed data cleaning to eliminate irrelevant variables, which consists of removing certain variables with missing

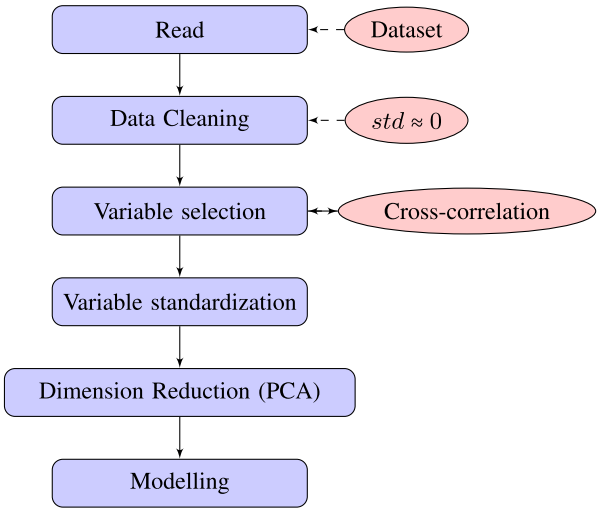

FIGURE 4. Preprocessing methodology.

or invariable data ($std \approx 0$), which do not add important information for the models. This process resulted in the elimination of 15 variables for both datasets.

In the next step, we removed highly correlated variables that represent redundant information. Variables with a correlation higher than 95% are excluded. For each pair of strongly correlated variables, we maintain the most frequent variable in the correlation map, given that it replaces the higher number of variables. From this process, the CICDoS2017 dataset excluded 12 variables, and the CICDDoS2019 dataset excluded 11 variables. After data cleaning and variable selection, we obtained 49 features for the CICDDoS2017 dataset and 50 features for the CICDDoS2019 dataset, without considering the flow ID (Src IP, Src Port, Dst IP, Dst Port, and Protocol) and Label variables.

The next step consisted of applying variable standardization to put different variables on the same scale. Finally, the standardized variables pass to the PCA mechanism. PCA is a dimensional reduction method that retains most of the variation information of the dataset. This process resulted in a reduced set of 15 parameters (principal components) for both CICDoS2017 and CICDDoS2019 datasets that explain 85% of the variance information of both datasets.

We performed the offline preprocessing using R software. Once we captured the selected variables and PCA parameters, we implemented and embedded the preprocessing module in the IDS.

C. MODELS HYPER-PARAMETERS TUNING

We used the preprocessed data for training the three ML and four DL models studied in this work. We built the ML models using the open-source library Scikit-learn [40]. For DL models, we used Keras [41], an open-source framework that allows us fast implementations of DL mechanisms. In addition, we trained and tested the models using Google Colaboratory, a cloud service based on Jupyter Notebook, which allowed us free use of Google GPUs and TPUs, with several ML/DL libraries.

To find the best models' hyper-parameters, we used the well-known Random Search approach. In this approach, we first define a wide search space as a bounded domain of hyper-parameter values. Then, we randomly sample points in that domain based on a uniform distribution. Finally, the hyper-parameters that provided the best performance (accuracy) for a model are selected. In the case of DL models, we used the hyper-parameter optimization tool named Talos Autonomy [42]. This tool helped us to obtain the best hyper-parameters following the Random Search approach.

Table 4 finally shows the optimal hyper-parameters for each proposed model.

D. EVALUATION

We adopted the evaluation metrics widely used to assess the performance of classification models. These metrics are Accuracy, Precision, Recall, and F1 Score, with the following definitions:

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}, \tag{1}$$

$$Precision = \frac{(TP)}{(TP + FP)}, \tag{2}$$

$$F1 - Score = \frac{(2 * TP)}{(2 * TP + FP + FN)}, \tag{3}$$

$$Recall = \frac{(TP)}{(TP + FN)}, \tag{4}$$

where TP, TN, FP, and FN stand for True Positive, True Negative, False Positive, and False Negative, respectively.

To better define the models' performance and compare them with previous works, we present both binary and multiclass metric measures. The binary metrics assessed the model's ability to detect attacks collectively and helped us evaluate if the method can correctly separate all attacks from legitimate traffic. The binary metrics are defined using weighted average scores. Calculating metrics in this manner is required for unbalanced datasets. Another particular and important binary metric is the False Positive Rate (FPR), defined as $FPR = (FP)/(FP + TN)$, which allows us to evaluate if the model can cause problems on the mitigation part. Low FPR rates are desirable to avoid eliminating/affecting legitimate traffic.

The multiclass metrics assess the models in terms of distinguishing each type of attack. The definitions of each metric presented before are applied in this evaluation. High performance on multiclass metrics could allow us to implement attack-type customized mitigation strategies if required. Table 5 shows the evaluation metrics of the different ML and DL-based mechanisms on the testing subsets.

E. DISCUSSION

1) MODELS PERFORMANCE

According to Table 5, among the ML techniques, KNN and SVM models were able to identify high-volume

TABLE 4. Optimized models' hyper-parameters using random search.

Model	CICDDoS2019 Dataset	CICDoS2017 Dataset
kNN	N neighbors=13, metric = 'Euclidean', weights = 'Uniform'.	N neighbors=13, metric = 'Euclidean', weights = 'Uniform'.
SVM	Kernel: Linear (optimized for large dataset), C = 1.0.	Kernel = 'Polynomial', degree = 3, C = 1.4.
RF	350 trees, min. sample split = 4, number of leaf nodes: 5, criterion: 'Gini' bootstrap=True.	750 trees, max depth=25, criterion='Entropy'.
MLP	Input: [Dense (15), reLU], Hidden: 3 layers ([L1: Dense(10), reLU], [L2: Dense(8), reLU], [L3: Dense(6), reLU]), output: [L4: Dense (4), softmax], TP: (optimizer: Adam, LF: sparse categorical cross entropy, 10 epochs, minibatch size = 300).	Input: [Dense (15), reLU], Hidden: 2 layers ([L1: Dense(12), reLU],[L2: Dense(10), reLU]), output: [Dense(7), softmax], TP: (optimizer: Adam, LF: sparse categorical cross entropy, 80 epochs, minibatch size = 300).
CNN	Input: [Conv1D(128), kernel = 3, padding, reLU], Hidden: 5 layers ([L1: Conv1D(32), kernel = 3, relu, padding], [L2: MaxPooling1D, poolsize 1], [L3: Flatten], [L4: Dense(100), reLU], [L5: Dropout (0.75)]), output: [Dense(4), softmax], TP: (optimizer: Adam, LF: sparse categorical cross entropy, 6 epochs, minibatch size = 200).	Input: [Conv1D(128), kernel = 3, padding, reLU], hidden: 5 layers ([L1: Conv1D(128), kernel = 3, padding, reLU,], [L2: MaxPooling1D, poolsize 1], [L3: Flatten], [L4: Dense(150) neurons, reLU], [L5: Dropout(0.75)], output: [Dense (7), softmax]), TP: (optimizer: Adam, LF: sparse categorical cross entropy, 110 epochs, minibatch size = 200).
GRU	Input: [GRU(100), reLU], hidden: 4 layers ([L1: GRU(100), reLU], [L2: Dense(150), reLU], [L3: Dense(128), reLU], [L4: Dropout(0.25)], output: [Dense(4), softmax]), TP: (optimizer: Adam, LF: sparse categorical cross entropy, 5 epochs, minibatch size = 100).	Input [GRU(100), reLU], hidden: 3 layers ([L1: GRU(100), reLU], [L2: Dense(150), reLU], [L3: Dense(128), reLU], [L4: Dropout(0.25)]), output: [Dense(7), softmax]), TP: (optimizer: Adam, LF: sparse categorical cross entropy, 100 epochs, minibatch size = 250).
LSTM	Input: [LSTM(80), reLU], hidden: 4 layers ([L1: LSTM(50), tanh], [L2: Dense(30), tanh], [L3: Dense(20), reLU], [L4: Dropout(0.25)]), output: [Dense(4), softmax], TP: (optimizer: Adam, loss function: sparse categorical cross entropy, 10 epochs, batch size = 2000).	Input: [LSTM(50), reLU], hidden: 4 layers ([L1: LSTM(50), reLU], [L2: Dropout(0.25)], [L3: Dense(40), reLU], [L4: Dense(25), reLU]), output: [Dense(7), softmax], TP: (optimizer: Adam, LF: sparse categorical cross entropy, 150 epochs, batch size = 300).

C: Regularization parameter, Ln: nth hidden layer, Dense(N)/Conv1D(N)/GRU(N): N stands for number of neurons, Conv1D: One dimensional convolution layer, Dropout(P): P stands for dropout probability, tanh/reLU/softmax: activation functions, TP: Training parameters, LF: loss function, Adam: Adaptive moment estimation.

TABLE 5. Multiclass and binary metrics to assess the model on testing sets.

		Binary metrics (benign/attack)					Multiclass metrics			
Dataset	**Method**	**Accu. (%)**	**Prec. (%)**	**F1-S. (%)**	**Recall (%)**	**FPR. (%)**	**Accu. (%)**	**Prec. (%)**	**F1-S. (%)**	**Recall (%)**
CICDDoS 2019	KNN	99.971	99.972	99.971	99.970	1.816	99.811	99.836	99.812	99.811
	SVM	99.774	99.773	99.769	99.774	10.23	99.719	99.718	99.713	99.719
	RF	96.361	98.514	97.148	96.361	7.832	95.415	97.517	96.099	95.415
	MLP	99.897	99.899	99.898	99.897	1.503	99.281	99.704	99.447	99.281
	CNN	99.130	99.335	99.194	99.130	4.392	99.086	99.317	99.156	99.086
	GRU	99.808	99.873	99.872	99.8873	4.454	99.809	99.843	99.819	99.819
	LSTM	99.877	99.879	99.878	99.877	1.571	99.849	99.851	99.849	99.848
CICDoS 2017	KNN	99.365	99.371	99.367	99.364	0.654	95.481	97.718	95.450	95.481
	SVM	98.776	98.800	98.782	98.776	1.262	97.972	98.101	98.006	97.972
	RF	96.182	96.208	96.108	96.182	0.855	90.992	93.036	90.675	90.992
	MLP	98.133	98.127	98.124	98.133	0.772	97.328	97.377	97.297	97.328
	CNN	98.881	98.886	98.883	98.881	0.974	97.405	97.442	97.395	97.405
	GRU	99.427	99.432	99.428	99.427	0.594	98.353	98.402	98.359	98.353
	LSTM	99.473	99.474	99.473	99.473	0.389	98.319	98.357	98.293	98.319

attacks (CICDDoS2019) with an accuracy above 99.77%. RF achieved 96.36% of accuracy. All DL models achieved an accuracy above 99.13% to detect high-volume attacks. Particularly GRU and LSTM showed the highest values on the accuracy with 99.81%, and 99.88%, respectively. For the FPRs results, KNN, MLP, and LSTM models presented the lowest values, which means they can better identify benign traffic. The multiclass metrics were slightly inferior for all ML/DL techniques, which indicates that it is a little bit more difficult to identify the high-volume attacks among them than just with the binary classes. Nevertheless, these differences were not significant.

In the case of the application-layer attacks (CICDoS2017), the metrics presented in Table 5 show that these attacks are more difficult to detect than the high-volume attacks of the CICDDoS2019 dataset. As in the previous case, DL mechanisms showed the highest accuracy values, above 98.13%. Especially, CNN achieved the highest accuracy (98.88%), but GRU and LSTM showed lower values of FPR (0.59% and 0.39%, respectively). Moreover, the slight reduction on the multiclass metrics, compared to binary metrics, indicates that the models slightly decreased their accuracy to discern among different application-layer attacks. However, the difference was not significant.

2) COMPLEXITY OF THE STUDIED MODELS

Two types of complexity were analyzed: 1) Time complexity, which defines the execution time of a method, and 2) space complexity, which explains the amount of memory required. The analyzed complexity was related to the model testing phase. This analysis is of particular interest when deploying our model in the simulated testbed. Table 6 shows the results of this analysis.

In the case of the time complexity, as we proposed to inspect the traffic through individual flow analysis, we measured the number of flows per second an ML/DL method can analyze and classify the flows. Thus, we collected and averaged the rates of 100 different executions for each ML/DL technique. We conducted this analysis in the Google Colaboratory platform with an Intel Xeon processor (not specified) with two cores @2.3 GHz, 13 GB RAM. Moreover, this platform has an NVIDIA Tesla K80 (GK210 chipset), 12 GB RAM, 2496 CUDA cores @560 MHz.

The work in [18] identified that in a real-world scenario with about 7000 different active hosts, peaks of 1681 flows/s are achieved on heavy traffic days. Given that the minimum value of flows/s of all our ML/DL methods was 3754.265 (SVM for CICDoS2017), we concluded that their implementation is feasible in real-world scenarios.

The time complexity is more critical for high-volume attacks, most present in the CICDDoS2019 dataset. The high-volume attacks lead to heavy traffic in the controller. The processing in the IDS must be as efficient as possible to avoid long delays that will turn in performance degradation of the entire network.

TABLE 6. Time and space models' complexity.

Dataset	Method	Time (Flows/s)	Space (floating-point parameters)
CICDDoS 2019	KNN	15220	29876490
	SVM	10650557	64
	RF	33041	3150
	MLP	42394	570
	CNN	39248	16612
	GRU	13744	131081
	LSTM	15893	59154
CICDoS 2017	KNN	6117	1540380
	SVM	3754	128280
	RF	14215	699546
	MLP	28663	639
	CNN	32757	75575
	GRU	13298	131081
	LSTM	19598	36647

In the space complexity analysis, all ML/DL methods have an equal number of inputs (flows) with the same number of features (15 principal components). Thus, what makes them different is the memory space that each trained model occupies. The space complexity information shown in Table 6 expresses the number of variables represented as 64-bit double-precision values (floating-point numbers) of each trained technique. These values are rough estimations and serve mainly for comparison among the explored methods.

For the DL methods MLP, CNN, GRU, and LSTM, the space complexity was estimated by the number of learnable parameters of each model (weights and biases). The KNN method stores all samples of the training set, and the memory of this method was estimated by the number of training samples times the number of features (principal components). We calculated the space requirement of the RF technique by the total number of nodes of all the trees. For the SVM method, we used different classifiers for each dataset. For the CICDoS2017 dataset, the SVM used support vector classification with a polynomial kernel. We estimated the memory space by the total number of support vectors (8552) times the input features (15). For the CICDDoS2019 dataset, the SVM used linear support vector classification. We calculated the memory space by the number of weights assigned to the features and the number of constants in the decision function.

Most ML models need to become very complex (time or space) to achieve a performance comparable to that obtained by DL models. In contrast, DL models demonstrated effectiveness with simple structures. In terms of time complexity, the SVM and the CNN methods were the most efficient for the CICDDoS2019 and CICDoS2017 datasets, respectively. The KNN method is the most expensive in terms of space complexity since it stores the whole training set. The use of large training datasets for the KNN model, as the CICDDoS2019, could represent a problem running the model in large network environments.

The DL methods provided the best trade-off between the classification performance and the models' complexity. As seen in Table 4, the maximum number of hidden layers used for all DL models is four layers. Remarkably, the MLP model used only two hidden layers to achieve high performance on classifying application-layer attacks (CICDoS2017).

GRU and LSTM models provided the best trade-off between prediction performance and complexity. The configuration of the trained LSTM requires less memory than the GRU. However, the GRU can process flows faster than LSTM.

3) COMPARISON WITH PREVIOUS WORKS

The metrics presented in this work are comparable with those demonstrated in the previous studies that used ML/DL models to detect attacks and that used the CICDDoS2019 dataset. The authors in [43] applied KNN, Decision Tree (DT), and RF and achieved accuracy levels less than 95.19%. The combined LSTM-Fuzzy method presented in [21] obtained an accuracy of 99.74% and FPR of 0.25%. The work in [16] combined an RNN with autoencoder and achieved a binary classification accuracy of 99%. The work in [17] combined LSTM and CNN models and obtained a detection accuracy above 98.9% on an altered version of the CICDDoS2019 dataset. This solution was not evaluated in a real or simulated network environment, though. The authors in [44] explored DT, Gradient Boosting (GB), and RF techniques and achieved accuracy up to 99.87% (GB) and FPR of 2.01%. However, they did not evaluate their security solution online in a real or simulated testbed. Finally, the recent work presented in [18] assessed four DL methods, including MLP, CNN, LTSM, and GRU, and four ML techniques that include SVM, Logistic Regression, kNN, and Gradient descendent. The GRU technique was the best technique that demonstrated accuracy higher than 99.94% using the

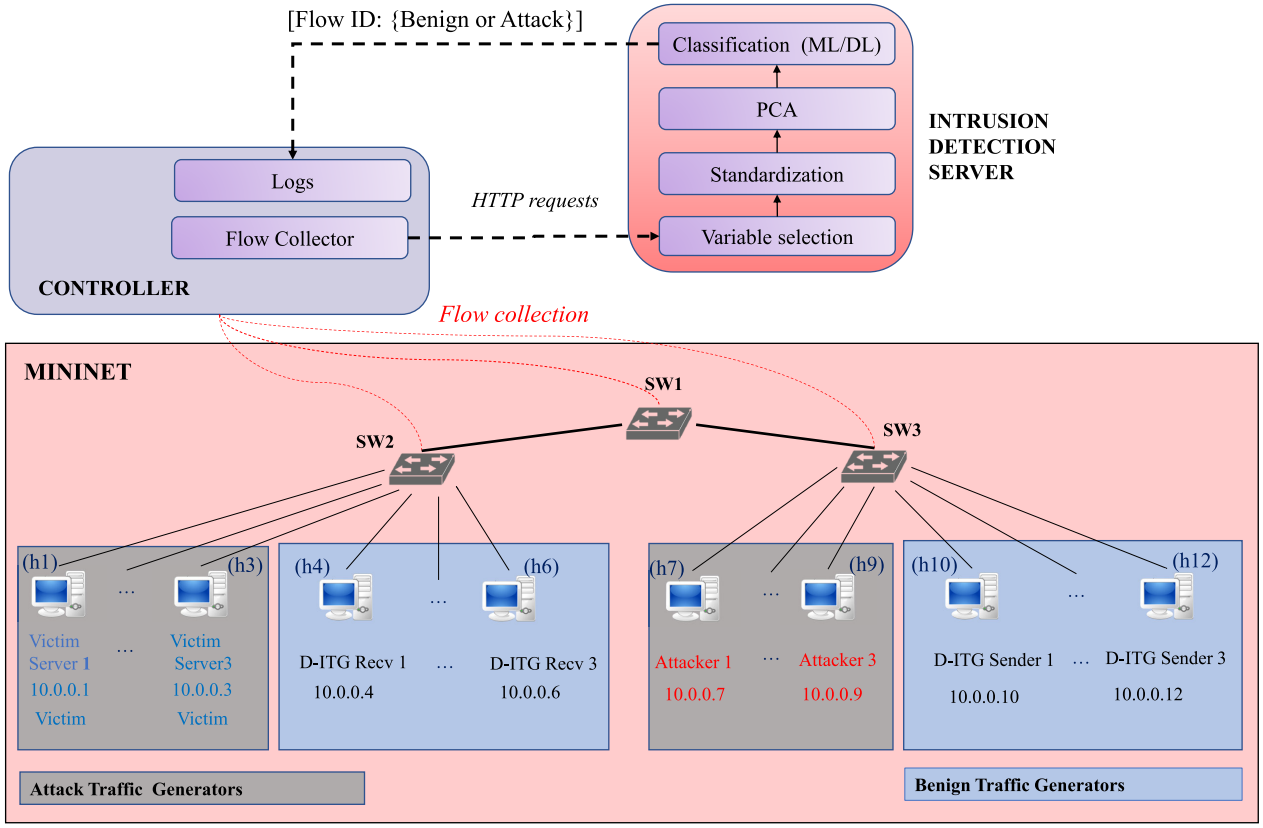

FIGURE 5. Experimental setup.

CICDDoS2019 dataset. However, this work did not present the evaluation using a simulated or real scenario that could reduce the offline performance achieved using the datasets.

Regarding the ML/DL models' performance for the CICDoS2017 dataset, the work in [20] explored six ML techniques, including RF, SVM, and MLP, achieving accuracy up to 95%. The high performance of the ML achieved in our work using the same dataset reveals the importance of our design of the flow collector and the preprocessing methodology.

We noted that most previous works did not test their proposals in a real or simulated network. Thus, our work offers a relevant contribution since several issues and challenges can appear when deploying the models in production network environments.

VII. EXPERIMENTAL SETUP

The threats considered in the experimental setup are UDP, SYN, and DrDNS from the CICDDoS2019 dataset. The attacks contemplated from the CICDoS2017 dataset are slow body, slow read, and slow header. We propose to evaluate the models trained with the CICDoS2017 and CICDDoS2019 datasets in a simulated testbed. The FPR is a fundamental measure of the performance of an IDS to be implemented in a production network. Thus, we excluded the SVM and CNN models from the testbed-based models' assessment since these methods presented the worst FPRs using both datasets (Table 5).

The proposed experimental setup implements a different network topology to the ones used for the models' training. The sizes of the topologies are similar, though. The CICDDoS2019 dataset was generated in a testbed with two separate networks: the victim and the attack networks. The victim network has one server, two switches, and four PCs. The attack network executed the different types of DDoS attacks to the victim server [29]. The testbed used to generate the CICDoS2017 dataset consisted of 10 victim web servers. All attacks were deployed sequentially to these servers, using one attacker at a time [26]. In short, both configurations used only a few network elements as the experimental setup proposed in this work.

Fig. 5 shows the experimental setup to test the proposed architecture. We used a single physical computer to simulate the Mininet network emulator, the ONOS controller, and the IDS, although we installed all these applications in different virtual servers.

We deployed a customized topology with three switches and 12 hosts using the Mininet network emulator. We set three victim hosts, h_1, h_2, and h_3, that implemented the victim servers during DoS/DDoS attacks. Also, we defined

three attacker hosts, h_7, h_8, and h_9, that ran the DoS/DDoS attacks using different attack tools. To generate benign traffic, we used the Distributed Internet Traffic Generator (D-ITG) [45]. This tool requires sender and receiver agents. We used three receivers, h_4, h_5, and h_6, that worked as the D-ITG receiver agents, and four senders, h_{10}, h_{11}, and h_{12}, that acted as the D-ITG sender agents.

The proposed architecture works as follows. An application installed in the controller runs the Flow Collector and Logs Manager modules. The Flow collector gathers the network traffic flows and sends them to the IDS. The IDS preprocesses the flows by performing variable selection, standardization, and dimensionality reduction with PCA. We defined the hyperparameters of the preprocessing execution during the offline preprocessing definition (Section VI.B). The trained ML/DL methods use the principal components to detect attacks. Finally, the IDS sent back the decisions over each flow back to the application in the controller, which saves the logs of the detection models.

In future work, we envision replacing the log manager module with a mitigation module. This module will deploy threat mitigation policies in the network by leveraging the programmability feature of the SDN architecture. In a simple approach, the mitigation module uses the decisions of the IDS over the individual flows to blacklist recurrent suspicious connections. These suspicious connections in the blacklist are blocked in the edge switches until they demonstrate a legitimate behavior. We will also explore other complete mitigation solutions, such as using optimization mechanisms that minimize the legitimate users affected and bandwidth used by malicious users.

A. BENIGN TRAFFIC PROFILE

We used the benign flows captured from the CICDoS2017 dataset to profile the legitimate traffic reproduced on the experimental testbed. Thus, we identified two packet-level characteristics and two flow-level characteristics from the CICDoS2017 benign flows. These characteristics were:

- *Packet-level characteristics:*
  - *Inter-departure Time (IDT)*, in milliseconds.
  - *Packet Size (Pkt Size)*, in bytes.
- *Flow-level characteristics:*
  - *Flow Duration*, in Number of Packets / flow.
  - *Inter-flow Time (Flow Delay)*, in milliseconds.

We performed survival and mean excess value analysis. This analysis demonstrated the four characteristics, IDT, Pkt Size, flow duration, and flow delay, follow a heavy-tailed distribution. Therefore, we proceeded with the distribution fitting to the most common heavy-tailed distributions, namely, Weibull, General Pareto, and Gamma. We determined the Kolmogorov-Smirnov statistic (K-S) [46],

$$D_N = \max_x |F_N(x) - F(x)|, \tag{5}$$

where $F_N(X)$ is the sample distribution and $F(x)$ is the empirical distribution of the data. The maximum value of the difference between these distributions is calculated for all values in support of X. The K-S statistic allowed us to establish the distribution model that can represent each parameter. Lower values of the K-S statistics indicate a better fit of the distribution model. As shown in Table 7, Pareto, Weibull, Weibull, and Pareto were the distribution models that best represented the parameters flow duration, flow delay, packet size, and IDT, respectively.

TABLE 7. K-S statistic for each parameter and different distributions.

Distribution	Flow duration	Flow Delay	Packet Size	IDT
Weibull	0.20058	**0.12613**	**0.40766**	0.21794
Pareto	**0.15085**	0.12795	0.53355	**0.13991**
Gamma	0.19655	0.1779	0.4257	0.35071
Exponential	0.33439	0.30994	0.5407	0.78935

We used D-ITG tool [45] to reproduce the benign traffic profile in the testbed. D-ITG is a platform capable of generating IPv4 and IPv6 traffic. This tool accurately replicates the workload of current Internet applications. We used D-ITG to generate traffic at the packet level accurately, producing appropriate stochastic processes for both IDT and Packet Size random variables. Also, we embedded the commands of the D-ITG in a Python-based script, which replicated the flow level characteristics, namely, flow delay and flow duration.

B. ATTACK TRAFFIC

We generated attack traffic using different free tools available online. Table 8 shows the tools used to generate each attack type. Moreover, this table shows the attack parameters that we explored and varied to evaluate the models under different attack conditions.

C. EXPERIMENTS

To evaluate the robustness of the intelligent classification mechanisms on the testbed, we ran several experiments varying the parameters of different tools, according to Table 8.

1) TRANSPORT LAYER ATTACKS

For transport layer attacks UDP and TCP-SYN, we explored different attack rates (r), as shown in Table 8. Our purpose was to evaluate the intelligent classification mechanisms at different rates r: Low, medium, high, and flood.

Fig. 6 a) shows the running of four experiments for TCP-SYN attacks. The horizontal axis contains the sequence of flows captured, and the vertical axis represents the type of event that each flow represents, that is, a benign or an attack flow. We started and finished each experiment with legitimate traffic, which allowed us to evaluate whether the model can avoid false positives in the free-attack condition. In total, we performed eight experiments for transport layer attacks. We noted that these attacks in flood mode made the entire network unavailable for benign users (Fig. 6 a)). In addition, on the low-rate mode of these attacks, the malicious flows have similar characteristics to benign traffic, which will make

TABLE 8. DoS/DDoS attack tools and parameters explored.

Layer	Attack (tool)	Attack's parameters	Attackers	Victim
Transport	UDP (Hping3 [47])	r : {Low, medium, high, flood}	h_7, h_8, h_9	h_2: SimpleHTTPServer [48]
	SYN (Hping3 [47])	r: {Low, fast, high, flood}	h_7, h_8, h_9	h_3: SimpleHTTPServer [48]
Application	Slow body (Slowhttptest [49])	r: {200, 300}, c: {1000, 2000, 3000, 4000, 5000}	h_7	h_1: SimpleHTTPServer [48]
	Slow read (Slowhttptest [49])	r: {200, 300}, c: {1000, 2000, 3000, 4000, 5000}	h_8	h_2: SimpleHTTPServer [48]
	Slow header (Slowhttptest [49])	r: {200, 300}, c: {1000, 2000, 3000, 4000, 5000}	h_9	h_3: SimpleHTTPServer [48]
	DrDNS (Hping3 [47])	r : {Flood}	h_7, h_8, h_9	h_1: SimpleHTTPServer [48]

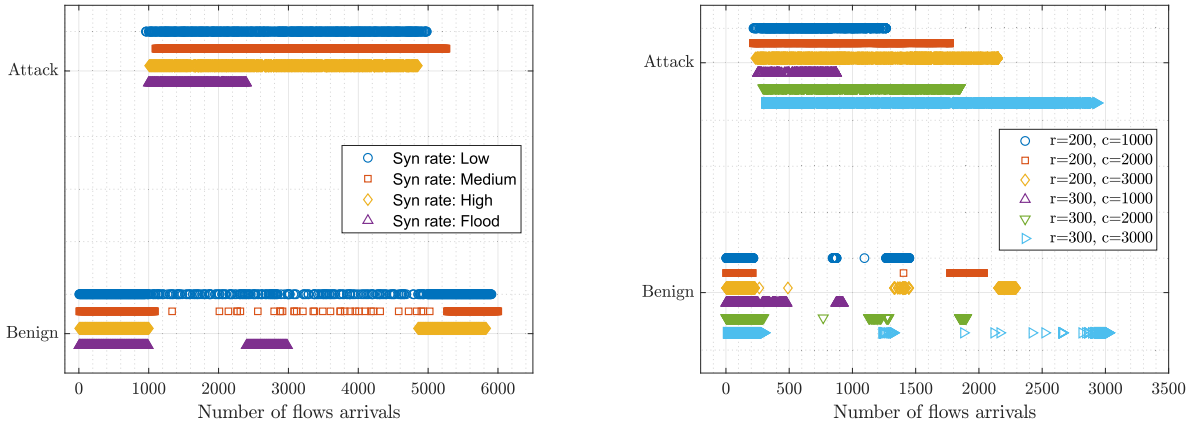

FIGURE 6. Experiment design. (a) Sample of four runs of TCP-SYN attacks. (b) Sample of six runs of slow read attacks.

them more challenging for intelligent mechanisms to detect. Nevertheless, if the attackers use too low attack rates, they will never deny the service to legitimate users.

2) APPLICATION-LAYER ATTACKS

For slow-rate application layer attacks, slow send body, slow send header, and slow read, we explored different attack connection rates (r) and different numbers of attack connections (c) for each attack. We performed 10 experiments for each slow-rate attack, combining the parameters r and c, as shown in Table 8.

Fig. 6 b) shows six runs for slow read attack. Axis are represented as in Fig. 6 a). As for the transport layer attacks, we started and finished the experiments with benign traffic. In total, we ran 30 experiments for the slow-rate attacks. We noted that for low r and low c, the slow-rate attacks were more challenging for the intelligent algorithm to detect since they have similar characteristics to benign traffic.

For the DrDNS attack, we used hping3 to perform the last step of this attack, namely, the reflection and amplification part. Therefore, UDP packets with long size were sent at a high rate to the victim host h_1 from the attackers (h_7, h_8 and h_9). As in previous cases, we considered legitimate traffic before and after the attack was executed.

D. INLINE MODEL EVALUATION

We evaluated the ability of the models to detect attacks on the experimental testbed. For the dataset-based models' evaluation, we used the classification metrics accuracy, precision, F1-score, and FPR. Using these metrics in the inline models' evaluation was not suitable. We performed multiple experiments with different attack conditions. The assessment of these experiments using classification metrics would have resulted in numerous and possibly confusing tables. Moreover, different from dataset-based model evaluation, we aimed to identify the model performance for different attack rates (r). Therefore, to examine the efficiency of the prediction of each model, we computed the detection rate.

We defined the detection rate (DR) as the ratio of the number of attack and benign flows correctly predicted to the total number of flows predicted by the model,

$$DR = \frac{\text{Total number of flows correctly predicted}}{\text{Total number of flows}}. \quad (6)$$

We computed the detection rate for every 1000 flow arrivals. After obtaining the detection rates, we averaged them over the attack rate (r). Thus, we presented a clear comparison of the performance of the ML/DL models.

1) TRANSPORT LAYER DDoS ATTACKS

Fig. 7 shows the averaged detection rate for TCP-SYN and UDP attacks using all intelligent models trained with the CICDoS2019 dataset. As shown in Fig. 7, the LSTM and GRU models provided the best performance in detecting SYN and UDP attacks, with an averaged detection rate above 90%. Notably, the GRU model reached an averaged detection rate

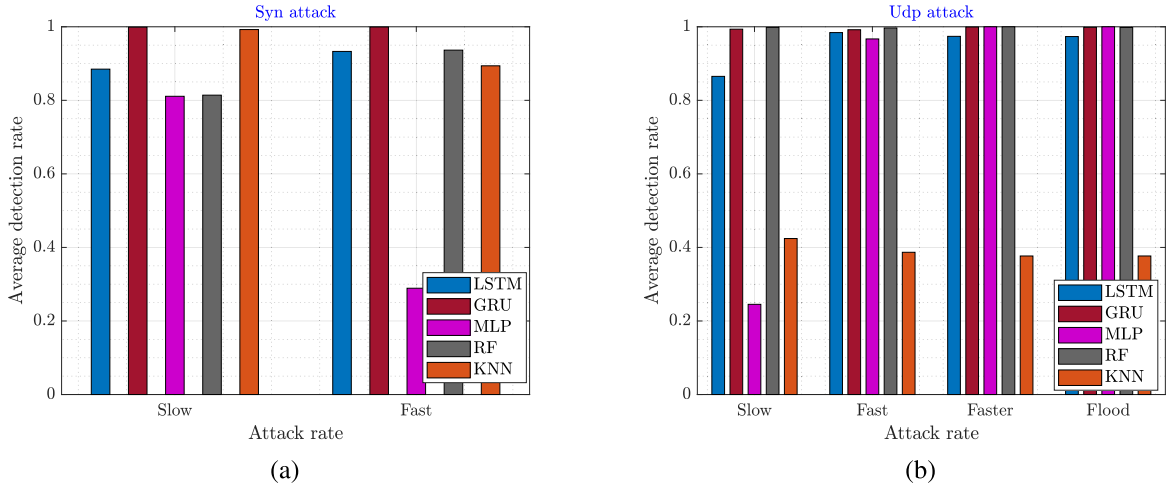

FIGURE 7. Testbed-based evaluation of transport layer attacks. (a) TCP-SYN. (b) UDP.

above 99%. Regarding the ML models, we observed that KNN and RF provided acceptable performance levels, above 80% for TCP-SYN attacks in fast rate mode.

We were limited in hardware resources since we used a single personal computer to execute all simulations. This factor restricted us to comprehensively evaluate the ML/DL models for TCP-SYN attacks in a faster rate and flooding mode. The limited computational capabilities of the ONOS controller did not allow us to process flows of TCP-SYN attacks at high rates and send them to the IDS. We discuss a solution to this issue in Section VIII.

In the case of the UDP attack, LSTM and GRU models provided the highest averaged detection rates that reached values above 85%. As in the TCP-SYN attack, the GRU model provided the highest performance that obtained an averaged detection rate above 99%. Regarding the ML methods, RF presented high performance, reaching averaged detection rates superior to 99%. KNN resulted ineffective in detecting UDP attacks.

2) APPLICATION-LAYER DoS/DDoS ATTACKS

Fig. 8 shows the averaged detection rate for the application layer attacks, slow body, slow read, slow header, and DrDNS using all ML/DL models trained with the CICDoS2017 dataset.

For application-layer slow-rate attacks, in general, DL mechanisms performed better than the ML methods. Most ML/DL models performed better for high r (300 attack connections/second), which means that for low values of r, legitimate and attack flows' characteristics become identical, which made it harder for ML/DL models to detect them. Nevertheless, it is relevant to recognize that for low values of r, the victim server will never fail, which means that the services provided by the server will remain available. Thus, our models are practical to avoid DDoS attacks.

From Fig. 8a) - 8c), we note that LSTM, GRU, and KNN demonstrated the highest performance for slow-rate attacks. Particularly, LSTM provided averaged detection rates above 85% for all slow-rate attacks with $r = 300$ attack connections/second. MLP showed a high detection rate only for slow header attack.

In general, the LSTM and GRU neural networks presented the best performance, with an averaged detection rate above 80% for all slow-rate attacks.

Even though the averaged detection rates for slow-rate attacks achieved in this work were not as high as the detection rates for high-volume attacks, their impact is significant, as explained in the following. The flows' arrival frequency of slow-rate attacks is low, and they can exhaust all the server resources and deny the availability to legitimate users with less bandwidth. Therefore, predicting even a low number of slow-rate attacks will reduce the impact of these threats and thus improve the availability of the services to legitimate users. Moreover, slow-rate attackers need to maintain a session for long times to complete the attack. Hence, we need only to recognize part of the attack flows to identify the intruder (IP and Source Port) to mitigate them.

For the application-layer DrDNS attack (Fig. 8 d)), all ML/DL models presented high performance, with average detection rates above 98%. This attack is high-volume traffic, which eases the differentiation from the legitimate traffic.

TABLE 9. Averaged FPR for testbed-based evaluation.

Layer	Attack	KNN	RF	MLP	GRU	LSTM
Transp.	SYN	0.006	0.005	0.165	0.003	0.155
	UDP	0.008	0.003	0.276	0.032	0.198
App.	DrDNS	0.003	0.006	0.000	0.002	0.069
	Slow body	0.011	0.003	0.003	0.003	0.014
	Slow read	0.010	0.009	0.001	0.011	0.009
	Slow header	0.012	0.004	0.012	0.005	0.038

Table 9 presents the average false-positive rate of all ML/DL techniques and attack types. For application-layer attacks, we observe that most ML/DL methods provide an FPR of less than 2%, which means legitimate users will

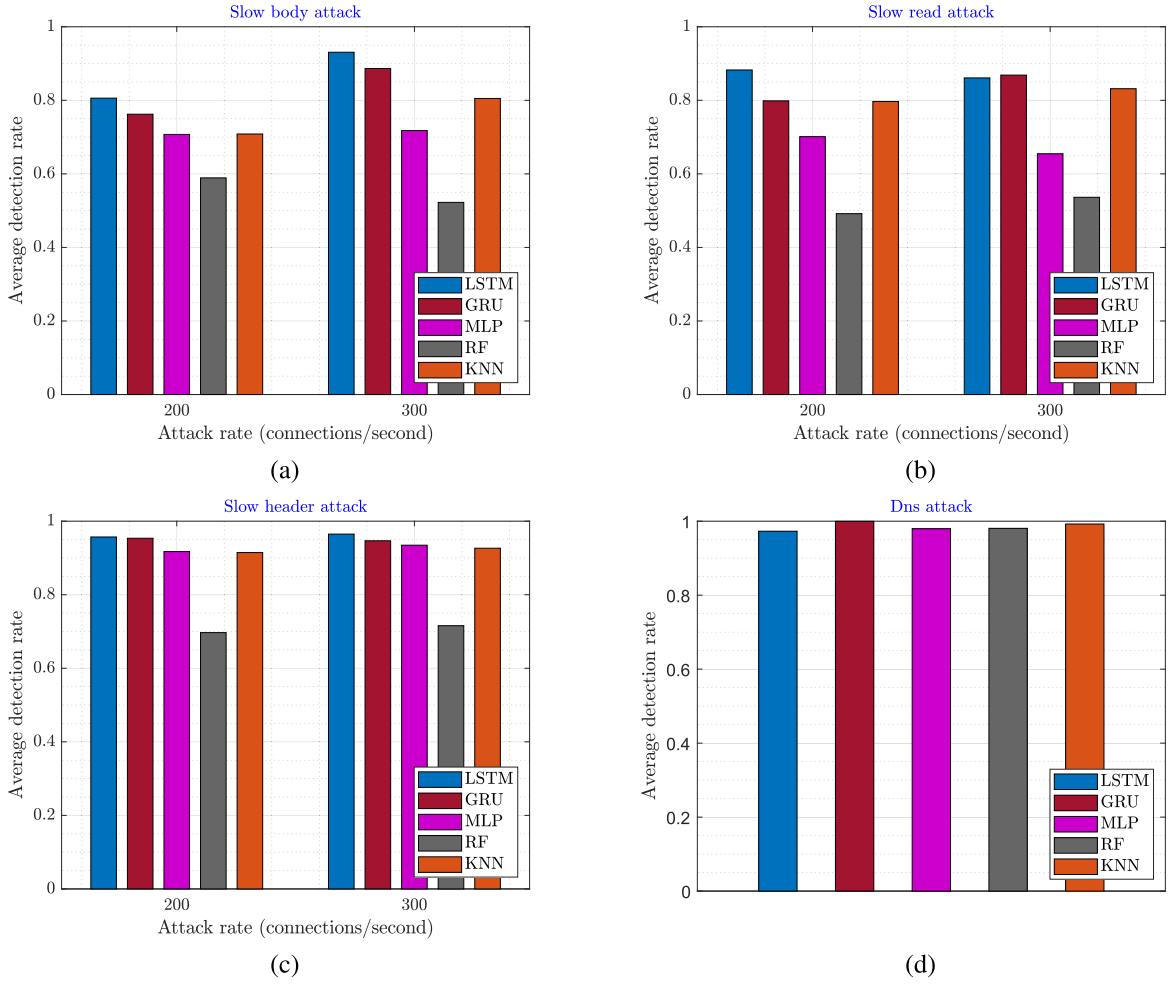

FIGURE 8. Testbed-based evaluation of application-layer attacks. (a) Slow body. (b) Slow read. (c) Slow header. (d) DrDNS.

be minimally affected when deploying mitigation strategies. For transport layer threats, FPRs are higher due to the high volume of flows during the attacks that complicate to discriminate a legitimate from a malicious flow. Nevertheless, most models present a proper FPR for high-volume DDoS attacks (less than 3%).

TABLE 10. Models' ranking for testbed-based evaluation.

Rank	Attacks					
	SYN	UDP	DrDNS	Slow body	Slow read	Slow header
1	GRU	RF	GRU	GRU	LSTM	GRU
2	KNN	GRU	KNN	KNN	GRU	MLP
3	LSTM	LSTM	RF	LSTM	KNN	LSTM

Table 10 shows the ranking of the top three models, based on the detection rate and FPR, for each attack evaluated on the testbed. GRU and LSTM models performed effectively high-volume and low-volume attacks in different conditions, which means these methods were the most robust.

We consider that the two RNN methods, the GRU and the LSTM, performed better than the other tested methods because of their ability to retain short-term and long-term information. Other ML/DL techniques presented in this work only maintain short-term memory. The GRU and LSTM methods implement a series of mechanisms called gates to regulate the learning and forgetfulness rates of data, guaranteeing that long-term (pass data) and short-term (new entries) information is maintained. Thus, both the network's state before an attack event occurs and the current analyzed flow's behavior are used to generate alarms. These essential features of the LSTM and GRU methods could have improved the classification of both benign and malicious flows, generating high outcomes regarding detection rates.

Moreover, the methods KNN, RF, and MLP performed successfully for different types of attacks. These results can help to develop next-generation firewalls to select the appropriate ML or DL model depending on the threat we want to avoid or minimize.

E. COMPARISON WITH PREVIOUS WORKS

Very few related works have performed an inline assessment of their security proposals using simulated environments.

The authors in [21] deployed a simulated scenario using Floodlight controller and Mininet to test an LSTM-FUZZY model, achieving an accuracy superior to 98%. However, they used the scenario only to generate datasets to train and test the model, but not in an inline evaluation manner. The work in [20] used the ONOS controller and Mininet to evaluate multiple ML models for detecting low-rate attacks. However, the accuracy achieved was low.

Unlike previous works, the scope of our study is extensive since we comprehensively evaluated the performance of seven ML/DL mechanisms in detecting diverse low-volume and high-volume DDoS attacks. Also, we varied the attack conditions to assess the robustness of each ML/DL method. Under this analysis, LSTM and GRU methods demonstrated to be the most robust detection mechanisms. Finally, the performances achieved in this work prove that our solution is realistic and can be implemented in production networks.

VIII. DISCUSSION

Next, we discuss our findings, limitations, and future work.

A. MODELS' PERFORMANCE AND ADAPTATION

We recognized the high performance of our models on the test subsets of the CICDoS2017 and CICDDoS2019 datasets. In these cases, the ML/DL models showed effectiveness for correctly predicting unseen traffic (testing set). However, when we changed the network topology, the performance of the models slightly decreased. The experimental network topology was different from the network configuration used for the models' training. Nevertheless, even in these conditions, the LSTM and GRU models achieved high enough performance in detecting high/low-volume transport and application-layer attacks.

Furthermore, generating datasets in our network configuration and using them to train the models would increase the model performance. We performed such experiments for the LSTM mechanism and achieved an average detection rate above 96% for DDoS slow-rate attacks. However, we know that the network environment is a nonstationary system. Thus, periodic offline training is required to maintain the high performance of our architecture. Furthermore, in the future, we will explore adaptive mechanisms to detect attacks even if the network architecture/environment changes drastically.

B. FUTURE WORK

Future work includes the introduction of the scalability component and a mitigation module in our proposed architecture.

1) SCALABILITY

To add more scalability to our proposed architecture (Fig. 2), we will include the following characteristics in future work. In the architecture presented in Fig. 2, the flow collector module gathers all flows from the network devices, which could introduce a bottleneck in large or high traffic networks. We aim to implement a packet level filter and a flow level filter to reduce the traffic monitoring intensity. A packet-level filter will select only a subset of packets to be used to build the flows. Regarding the slow-rate attacks, we need to be careful on conserving enough packets since most of the application-layer attacks contain flows with only a few packets. Filtering too many packets will result either in single-packet flows or eliminating some flows, producing low detection performance [26]. Thus, a flow level filter is desirable for slow-rate attacks. In transport layer DDoS attacks, the packet level filter is appropriate since the attacks use high-volume traffic. Only part of the packets to build the flows will be enough to obtain a good model performance.

We also plan to embed the preprocessing mechanism in the Flow Collector module in the ONOS controller (see Fig. 2). We defined the relevant variables in the offline preprocessing methodology. Therefore, we only need to calculate those relevant variables of the flows in the flow collector module. Moreover, the real-time application of variable standardization and PCA processing requires PK sums + PK multiplications, considering the input flow characteristic vector $X_{1\times K}$, with K selected features and P principal components. Thus, we can introduce this preprocessing mechanism in the flow collector module. These changes will reduce the information traffic from the controller to the external IDS, which will improve the scalability of our proposed architecture.

Furthermore, in this work, we considered only one controller in the experimental setup. Nevertheless, using multiple controllers is also of particular interest in an SDN-based security system design that can increase the scalability factor of our proposed solution. This future research line is promising, and we will consider it in future work.

Finally, we used simplified network topology in the experimental setup since we aimed to validate the performance of ML/DL models under different attack conditions. The parameters explored were the attack connection rate (r) and the number of attack connections (c) for each attack. Varying the number of attackers and evaluating the IDS response is also of particular importance in our research line, and we will consider it in future work. We can predict that using more distributed DDoS attackers will make the attacks even more challenging for their detection. However, we consider that, because of their high accuracy and low FPR, the trained models will keep a very acceptable performance.

2) MITIGATION MODULE

A mitigation module will be included in our proposed architecture, which will countermeasure the attacks studied in this work. Although we can implement a simple mitigation strategy using the programmability feature of SDN, such as closing all suspicious communications through inserting blocking rules in edge switches, the design of an effective and efficient mitigation strategy demands solving several challenges. Among these challenges are how to optimize the use of computational and storage resources of the controller and switches to deploy the mitigation policies, how to minimize the reaction time of the mitigator, how to obtain a scalable

solution, etc. We envision exploring a solution to these issues in future work.

IX. CONCLUSION

DoS/DDoS attacks are the most harmful threats affecting the networks. In this work, we presented a solution using artificial intelligence to detect two types of threats: transport-layer and application-layer DDoS attacks. First, we proposed a modular SDN-based architecture with components that can be modified or improved separately, providing flexibility to test different intelligent methods to detect diverse attacks. Moreover, we explored four DL models and three ML models that demonstrated an accuracy performance above 99% on the testing phase, using two up-to-date datasets with real network traces. Our proposed solution was evaluated in an emulated testbed, using Mininet and the ONOS controller. In this configuration, the detection rate of the trained models remained high enough. We presented a ranking of the best models evaluated on the testbed, and we concluded that GRU and LSTM models maintained the highest detection rates in the inline model evaluation. These models achieved high detection rates, up to 95% for slow-rate attacks and above 98% for high-volume attacks, demonstrating high robustness. In general, DL models exhibited higher detection rates than ML models for all types of attacks studied in this work.

The use of open and complete tools (e.g., ONOS controller) for the deployment of the architecture proposed eases its migration to production environments. The next step in our project includes increasing its scalability factor to the solution. Moreover, for future work, we plan to implement and experiment with an optimized mitigation strategy.

ACKNOWLEDGMENT

The authors would like to thank Maritza Rosales Hernández, Fátima Sánchez Suárez, and Martín Helmut Domínguez Álvarez, for their assistance in training and testing the ML/DL models using the datasets and the simulated architecture.

REFERENCES

[1] M. M. Salim, S. Rathore, and J. H. Park, "Distributed denial of service attacks and its defenses in IoT: A survey," *J. Supercomput.*, vol. 76, pp. 5320–5363, Jul. 2019.

[2] K. Srinivasan, A. Mubarakali, A. S. Alqahtani, and A. D. Kumar, "A survey on the impact of DDoS attacks in cloud computing: Prevention, detection and mitigation techniques," in *Intelligent Communication Technologies and Virtual Mobile Networks*. Cham, Switzerland: Springer, 2019, pp. 252–270.

[3] D. Gurusamy, M. Deva Priya, B. Yibgeta, and A. Bekalu, "DDoS risk in 5G enabled IoT and solutions," *Int. J. Eng. Adv. Technol.*, vol. 8, no. 5, pp. 1574–1578, 2019.

[4] Kaspersky. (2021). *Kaspersky Q4 2020 DDoS Attacks Report*. [Online]. Available: https://securelist.com/ddos-attacks-in-q4-2020/100650/

[5] A. Praseed and P. S. Thilagam, "DDoS attacks at the application layer: Challenges and research perspectives for safeguarding web applications," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 661–685, 1st Quart., 2019.

[6] J. C. Correa Chica, J. C. Imbachi, and J. F. Botero Vega, "Security in SDN: A comprehensive survey," *J. Netw. Comput. Appl.*, vol. 159, Jun. 2020, Art. no. 102595.

[7] N. Sultana, N. Chilamkurti, W. Peng, and R. Alhadad, "Survey on SDN based network intrusion detection system using machine learning approaches," *Peer-Peer Netw. Appl.*, vol. 12, no. 2, pp. 493–501, Jan. 2019.

[8] R. Swami, M. Dave, and V. Ranga, "Software-defined networking-based DDoS defense mechanisms," *ACM Comput. Surv.*, vol. 52, no. 2, p. 28, 2019.

[9] C. Birkinshaw, E. Rouka, and V. G. Vassilakis, "Implementing an intrusion detection and prevention system using software-defined networking: Defending against port-scanning and denial-of-service attacks," *J. Netw. Comput. Appl.*, vol. 136, pp. 71–85, Jun. 2019.

[10] P. Wang, L. T. Yang, X. Nie, Z. Ren, J. Li, and L. Kuang, "Data-driven software defined network attack detection : State-of-the-art and perspectives," *Inf. Sci.*, vol. 513, pp. 65–83, Mar. 2020.

[11] M. Idhammad, K. Afdel, and M. Belouch, "Semi-supervised machine learning approach for DDoS detection," *Appl. Intell.*, vol. 48, no. 10, pp. 3193–3208, 2018.

[12] 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.*, pp. 1–7, Apr. 2019.

[13] N. N. Tuan, P. H. Hung, N. D. Nghia, N. V. Tho, T. V. Phan, and N. H. Thanh, "A DDoS attack mitigation scheme in ISP networks using machine learning based on SDN," *Electronics*, vol. 9, no. 3, p. 413, Feb. 2020.

[14] X. Liang and T. Znati, "A long short-term memory enabled framework for DDoS detection," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.

[15] V. Punitha, C. Mala, and N. Rajagopalan, "A novel deep learning model for detection of denial of service attacks in HTTP traffic over internet," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 33, no. 4, pp. 240–256, 2020.

[16] M. S. Elsayed, N.-A. Le-Khac, S. Dev, and A. D. Jurcut, "DDoSNet: A deep-learning model for detecting network attacks," in *Proc. IEEE 21st Int. Symp. 'A World Wireless, Mobile Multimedia Netw.' (WoWMoM)*, Aug. 2020, pp. 391–396.

[17] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng, "FlowGuard: An intelligent edge defense mechanism against IoT DDoS attacks," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9552–9562, Oct. 2020.

[18] M. V. O. Assis, L. F. Carvalho, J. Lloret, and M. L. Proença, "A GRU deep learning system against attacks in software defined networks," *J. Netw. Comput. Appl.*, vol. 177, Mar. 2021, Art. no. 102942.

[19] CIC Datasets. (2020). *Canadian Institute for Cybersecurity*. [Online]. Available: https://www.unb.ca/cic/datasets/index.html

[20] J. A. Pérez-Díaz, I. A. Valdovinos, K.-K. R. Choo, and D. Zhu, "A flexible SDN-based architecture for identifying and mitigating low-rate DDoS attacks using machine learning," *IEEE Access*, vol. 8, pp. 155859–155872, 2020.

[21] M. P. Novaes, L. F. Carvalho, J. Lloret, and M. L. Proenca, "Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment," *IEEE Access*, vol. 8, pp. 83765–83781, 2020.

[22] S. Gumaste and S. Shinde, "Detection of DDoS attacks in OpenStack-based private cloud using apache spark," *J. Telecommun. Inf. Technol.*, vol. 4, pp. 62–71, Jan. 2021.

[23] A. V. Kachavimath and D. G. Narayan, "A deep learning-based framework for distributed denial-of-service attacks detection in cloud environment," in *Advances in Computing and Network Communications*, S. M. Thampi, E. Gelenbe, M. Atiquzzaman, V. Chaudhary, and K.-C. Li, Eds. Singapore: Springer, 2021, pp. 605–618.

[24] T. V. Phan, T. M. R. Gias, S. T. Islam, T. T. Huong, N. H. Thanh, and T. Bauschert, "Q-MIND: Defeating stealthy DoS attacks in SDN with a machine-learning based defense framework," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.

[25] N. Dayal, P. Maity, S. Srivastava, and R. Khondoker, "Research trends in security and DDoS in SDN," *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 6386–6411, Dec. 2016.

[26] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling," *Comput. Netw.*, vol. 121, pp. 25–36, Jul. 2017.

[27] K. Ozdincer and H. A. Mantar, "SDN-based detection and mitigation system for DNS amplification attacks," in *Proc. 3rd Int. Symp. Multidisciplinary Stud. Innov. Technol. (ISMSIT)*, Oct. 2019, pp. 1–7.

[28] CIC Flow Meter. (2020). *Canadian Institute for Cybersecurity*. [Online]. Available: https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter

[29] 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. (ICCST)*, Oct. 2019, pp. 1–8.

[30] (2020). *Mininet*. [Online]. Available: http://mininet.org/
[31] ONF. (2020). *Onos*. [Online]. Available: https://www.opennetworking.org/onos/
[32] D. Arndt. (2020). *Flowbag*. [Online]. Available: https://github.com/DanielArndt/flowtbag
[33] A. Habibi Lashkari, G. Draper Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy*, 2017, pp. 253–262.
[34] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Raheem, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE Access*, vol. 6, pp. 33789–33795, 2018.
[35] G. Karatas, O. Demir, and O. K. Sahingoz, "Increasing the performance of machine learning-based IDSs on an imbalanced and up-to-date dataset," *IEEE Access*, vol. 8, pp. 32150–32162, 2020.
[36] J. Ye, X. Cheng, J. Zhu, L. Feng, and L. Song, "A DDoS attack detection method based on SVM in software defined network," *Secur. Commun. Netw.*, vol. 2018, Apr. 2018, Art. no. 9804061.
[37] H. Polat, O. Polat, and A. Cetin, "Detecting DDoS attacks in software-defined networks through feature selection methods and machine learning models," *Sustainability*, vol. 12, no. 3, p. 1035, Feb. 2020.
[38] S. Haider, A. Akhunzada, I. Mustafa, T. B. Patel, A. Fernandez, K.-K. R. Choo, and J. Iqbal, "A deep CNN ensemble framework for efficient DDOs attack detection in software defined networks," *IEEE Access*, vol. 8, pp. 53972–53983, 2020.
[39] T. A. Tang, D. McLernon, L. Mhamdi, S. A. R. Zaidi, and M. Ghogho, "Intrusion detection in sdn-based networks: Deep recurrent neural network approach," in *Deep Learning Applications for Cyber Security*. Cham, Switzerland: Springer, 2019, pp. 175–195.
[40] (2020). *Scikit-Learn*. [Online]. Available: https://scikit-learn.org/stable/
[41] (2020). *Keras*. [Online]. Available: https://keras.io/
[42] Autonomio Talos. (2020). *[Computer Software]*. [Online]. Available: http://github.com/autonomio/talos
[43] D. V. V. S. Manikumar and B. U. Maheswari, "Blockchain based DDoS mitigation using machine learning techniques," in *Proc. 2nd Int. Conf. Inventive Res. Comput. Appl. (ICIRCA)*, Jul. 2020, pp. 794–800.
[44] J. P. Abreu Maranhão, J. P. Carvalho Lustosa da Costa, E. Pignaton de Freitas, E. Javidi, and R. Timóteo de Sousa Júnior, "Error-robust distributed denial of service attack detection based on an average common feature extraction technique," *Sensors*, vol. 20, no. 20, p. 5845, Oct. 2020.
[45] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre, "D-ITG distributed internet traffic generator," in *Proc. 1st Int. Conf. Quant. Eval. Syst. (QEST)*, Sep. 2004, pp. 316–317.
[46] M. Bonamente, *Statistics and Analysis of Scientific Data*. New York, NY, USA: Springer, 2017.
[47] (2019). *Hping3*. [Online]. Available: https://tools.kali.org/information-gathering/hping3
[48] (2020). *Simple Http Server*. [Online]. Available: https://docs.python.org/2/library/simplehttpserver.html
[49] (2020). *Slowhttptest*. [Online]. Available: https://code.google.com/p/slowhttptest/

NOE MARCELO YUNGAICELA-NAULA received the B.Sc. degree in electronic and telecommunication engineering from the Universidad de Cuenca, Cuenca, Ecuador, in 2015, and the M.Sc. degree in intelligent systems from the Tecnologico de Monterrey, in 2018, where he is currently pursuing the Ph.D. degree. From November 2017 to March 2018, he was a Visiting Scholar with Concordia University, Montreal, QC, Canada. His current research interest includes the security automation on new generation networks using techniques of artificial intelligence.

CESAR VARGAS-ROSALES (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering, majoring in communications and signal processing from Louisiana State University. He has coauthored the book *Position Location Techniques and Applications* (Academic Press/Elsevier). His research interests include personal communications, 5G/6G, cognitive radio, MIMO systems, stochastic modeling, traffic modeling, intrusion/anomaly detection in networks, position location, interference, network and channel coding, and optimum receiver design. He is a Senior Member of the IEEE Communications Society Monterrey Chapter Chair and the Faculty Advisor of the IEEE-HKN Lambda-Rho Chapter at the Tecnologico de Monterrey. He is a member of the Mexican National Researchers System (SNI), the Mexican Academy of Science (AMC), and the Academy of Engineering of Mexico. He was also the Technical Program Chair of the IEEE Wireless Communications and Networking Conference (IEEE WCNC). He is an Associate Editor of IEEE Access and *International Journal of Distributed Sensor Networks*.

JESUS ARTURO PEREZ-DIAZ received the B.Sc. degree in computer science from the Autonomous University of Aguascalientes, in 1995, and the Ph.D. degree in new advances in computer science systems from the Universidad de Oviedo, in 2000. He became a Full Associate Professor with the University de Oviedo, from 2000 to 2002. He is currently a Researcher and a Professor with the Tecnologico de Monterrey at Querétaro, Mexico. He received a research stay at the Louvain-la-Neuve University, Belgium. His research interests include in cyber-security in SDN and design of communications protocols, where he has supervised several master and Ph.D. theses in the field. He is a member of the Mexican National Researchers Systems. He received the Best Student Award for the B.Sc. degree. He was recognized by the COIMBRA Group, as one of the Best Young Latin-American Researchers, in 2006. He has been awarded by the CIGRE and by Intel for the development of innovative systems.