Received 30 March 2023, accepted 9 May 2023, date of publication 17 May 2023, date of current version 24 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3277397

RESEARCH ARTICLE

Detecting and Mitigating Botnet Attacks in Software-Defined Networks Using Deep Learning Techniques

MUHAMMAD WAQAS NADEEM[1], HOCK GUAN GOH[1], YICHIET AUN[1], AND VASAKI PONNUSAMY[2]
[1]Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman, Kampar, Perak 31900, Malaysia
[2]Higher Colleges of Technology, Fujairah Men's Campus, Fujairah, United Arab Emirates

Corresponding author: Muhammad Waqas Nadeem (waqasnadeem@1utar.my)

ABSTRACT Software-Defined Networking (SDN) is an emerging architecture that enables flexible and easy management and communication of large-scale networks. It offers programmable and centralized interfaces for making complex network decisions dynamically and seamlessly. However, SDN provides opportunities for businesses and individuals to build network applications based on their demands and improve their services. In contrast, it started to face a new array of security and privacy challenges and simultaneously introduced the threats of a single point of failure. Usually, attackers launch malicious attacks such as botnets and Distributed Denial of Service (DDoS) to the controller through OpenFlow switches. Deep learning (DL)-based security applications are trending, effectively detecting and mitigating potential threats with fast response. In this article, we analyze and show the performance of the DL methods to detect botnet-based DDoS attacks in an SDN-supported environment. A newly self-generated dataset is used for the evaluation. We also used feature weighting and tuning methods to select the best subset of features. We verify the measurements and simulation outcomes over a self-generated dataset and real testbed settings. The main aim of this study is to find a lightweight DL method with baseline hyper-parameters to detect botnet-based DDoS attacks with features and data that can be easily acquired. We observed that the best subset of features influences the performance of the DL method, and the prediction accuracy of the same method could be variated with a different set of features. Finally, based on empirical results, we found that the CNN method outperforms the dataset and real testbed settings. The detection rate of CNN reaches 99% for normal flows and 97% for attack flows.

INDEX TERMS Botnet attack, convolutional neural network, deep learning, distributed denial-of-service attack, network security, software-defined networking.

I. INTRODUCTION

The development of the internet is rapidly growing; the limitations of traditional networks have been explored. The emerging issues of the conventional networks can be solved by patching the network, which makes the network more bloated and the control ability of the network becomes weaker. The invention of Software-Defined Networking (SDN) [1], [2] has resolved these problems by decoupling the data and control planes. SDN became famous among the network community due to its novel architecture and can fulfill the demands of fast-growing networks. SDN has a centralized control architecture, so the SDN controllers can access all the OpenFlow switches in their range and control the entire network through the open south API interfaces [3], [4]. It is also known as the three-layer network architecture, application, control, and data layers. The application layer runs all the policies and rules the network administrator defines, and the SDN controller can adopt these rules dynamically. Any modification in the application layer may change the behavior of the whole network. The application layer is an excellent development by the open-source platform,

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

which does not force the administrator to entirely relies on vendors [5]. Positively, the SDN allows administrators to eliminate license constraints and cloud-develop customized network applications over general-purpose hardware. The control layer is known as the brain of the architecture, and SDN controllers run in this layer. The controllers receive the rules from the application layer, decode them into readable messages, and forward them to the underlying data layer; after that, they collect the feedback from the data layer and pass it back to the application layer. Moreover, a decision is made on the control layer, and the rules are implemented in the data layer. The data layer is non-intelligent, and different hardware devices, such as routers, OpenFlow switches, etc., exist in this layer, and instructions are passed by the control layer [6].

Furthermore, it simplifies and eases the network's development, deployment, and maintenance. The network functions and features can be easily enhanced by updating and introducing new applications. Besides being cost-effective, SDN-based networks require simple hardware devices and have almost no compatibility issues [7]. Network access is allowed without revealing the details of the different underlying layers.

Although SDN is an inordinate invention that can improve the network's flexibility and controllability and is considered a double-edged sword due to central control, a single controller can easily manage the network. SDN also helps to improve traditional networks' security measures, but SDN has yet to be extensively trustworthy security measures to support the vision of next-generation networking [8]. Due to its centralized control nature and innovative architecture, it may introduce new security threats and can lead toward a single point of failure. Furthermore, the centralized nature of SDN architecture unlocks the way for attackers to launch different attacks such as botnets, DDoS [6], saturation, and other types.

Botnet attacks are considered malicious attacks that become a critical threat in the next generation of networking [9]. The botnet is a network-based attack that breaches multiple computers into "bots" to launch malicious activities such as DDoS, identifying theft, domain name system spoofing, spamming, phishing, etc. In a botnet attack, a malicious actor, "Bot master," tries to get unauthorized access to a single device and then implements botnet malware to take control of the device without alienating its legitimate users. After that, establish a connection of bots with a Command and Control (C&C) center owned by the attacker, and the bots remain ready to launch malicious activities under the instructions of C&C.

Currently, botnet technology is typically used to launch the most sophisticated DDoS attacks in SDN and Internet of Things (IoT) based networks. The power and flexibility of the technology enable the botnet technology to generate several types of DDoS attacks. There are four main reasons to use advanced technology by the attackers: (i) a powerful flooding attack can be quickly generated with a large number of bots, (ii) there is difficulty in finding the actual attacker, (iii) they can use different protocols to dodge the security mechanism (iv) the attack and normal traffic have similarity, so, the real-time detection is difficult. With the advancement in technology, the attacker becomes more competent, and they know the structure of the SDN-based networks. They know they can control the whole network if they access the controller. The attackers can easily breach multiple computers into bots and make a member of the botnet force perform malicious actions on the SDN controller. So, the botnets have resulted in the progress of severe and massive DDoS attacks against the SDN and may cause a single failure point.

Positively, deep learning-based applications are widely used for the detection of intrusion in various fields of networking. Examining the behavior in SDN becomes a new research area because using deep learning to study the behavior could be a first attempt for early detection [10], [11], as machines can respond faster than humans. Deep learning techniques allow machines to make decisions through training, features, trial, and error methods. DL approaches use historical data to realize the network behavior and predict upcoming traffic flows. The DL techniques proved excellent performance in classifying normal and attack traffic flows. These techniques do not rely on packet payload and take a set of a particular set of features to make predictions. DL techniques could help the SDN controllers to realize the network's current status. In existing solutions [12], [13], most researchers try to detect the simple DDoS attack in the SDN using machine learning and deep learning approaches. Moreover, the botnet-based DDoS attack potentially affects the SDN due to its centralized nature, so we need to focus on botnet-based DDoS. In most studies [8], [14], the researchers used many traffic features to train ML/DL methods. Extracting and collecting significant features from a real traffic flow is challenging and time-consuming due to authorization or accessibility. Furthermore, most existing methods are verified by simulations or experimental datasets only; real-time testbed validation is intermittent.

A. MOTIVATION AND CONTRIBUTION

This work aims to conduct a comparative study on deep learning methods with optimized feature selection from network traffic flow to detect the botnet-based DDoS attack in an SDN environment. The selection of optimal features can improve DL methods' detection performance and accuracy and can help to reduce the SDN controller's overhead. As we know, botnet attacks can hijack the controller and launch malicious activities. So, DL-based techniques and dedicated bot management are optimal solutions for SDN to deal with botnet traffic. The progressive development in security measures of SDN has motivated us to develop a DL method named "DepBot" to detect and mitigate botnet-based DDoS. In short, our contributions are as follows:

- Optimal Features: Compressing multiple features and identifying the most optimal features is our one contribution. In this, a comparison of DL methods is performed to detect the botnet-based DDoS. The methods have experimented on a custom-developed dataset.

- Factor Analysis: We select and retrieve the optimal features into different subsets. We use feature weighting and threshold tuning-based evaluation techniques to choose the best feature subset. Based on this, we have identified optimal features and converted them into five subsets.
- DL Analysis: We select and use the five DL methods, such as Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), Multilayer Perception (MLP), Long Short Term Memory (LSTM), and Deep Neural Network (DNN), that mostly appears in network intrusion detection-based applications. We use time-series-based techniques in RNN and LSTM to develop the detection methods.
- Performance: The performance of the DL methods is evaluated using accuracy, precision, detection rate, and F1-score metrics. These methods build a relationship with raw data through multiple levels of abstraction. We observed that the DL methods produce good results on subset 3. This study would help to select the optimal features and best DL method for botnet-based DDoS attack detection in SDN.
- Mitigation: We used graph theory and dynamic flow deletion concepts to adopt a more targeted flow-dropping strategy.

II. RELATED WORK

This section discusses the recently developed network intrusion methods for SDN. Researchers have recently proposed various ML and DL-based methods for detecting botnet attacks. Some appropriate research works have been done in DDoS and botnet attack detection. [15] proposed "high-level PSI-rooted subgraph-based features" to detect the botnets in IoT, and they also used a hybrid combination of machine learning and deep learning methods. Their main aim was to detect the attack with a minimum number of features that can help to reduce the space and faster the detection speed. Reference [16] developed an IoT botnet sensor system based on unsupervised learning. They do the hyper parameterization of SVM using the "Grey Wolf Optimization (GWO) algorithm" to determine the critical features for a botnet attack. The suggested technique helps to detect IoT botnet attacks launched from intelligent, vulnerable nodes. We observed another network-based anomaly detection approach called N-BaIoT in [17], which takes snapshots of the network behavior and employs deep autoencoders to detect the attack traffic. Reference [18] proposed a hybrid method of PSO algorithms with a voting mechanism to detect botnet attacks in IoT. In this method, the PSO is used to pick the critical and remarkable features, and the election process is performed using DNN, Decision Tree (DT) C4.5, and SVM to detect the botnet attacks. In another research [19], the authors proposed a CNN-based method named BoTIDS. They observed that this method produced good results on a complete dataset and a predefined subset of 10 features compared to LSTM and RNN. However, the evaluation results on a subset are comparatively better than the full dataset. We observed another CNN-based Anomaly Intrusion Detection System (AIDS) [20]. This system is tested on BoT-IoT and Network Intrusion Detection (NID) datasets. The proposed system produced effective results on the NID dataset compared to BoT-IoT.

In [8], we observed a two-level DDoS attack detection method based on DL and Information Entropy (IE). First, they used information entropy to detect the suspicious port and components in coarse granularity. Then, they executed a CNN-based fine-grained detection mechanism to classify the attack and normal traffic. Reference [21] developed a DNN-based network intrusion detection system to show the potential of DL methods for malicious traffic detection. A combined method of Autoencoder and RNN named DDoS-Net has been proposed by [22] to detect DDoS attacks in SDN environments. Reference [23] proposed an LSTM-based DDoS detection method and compared the performance of this method with Random-Forest (RF) based approach, and it reduced the error rate from 7.517% to 2.103%. Reference [24] present a deep learning-based lightweight and practical DDoS detection system called LUCID, which shows the capabilities of CNN to classify traffic flows as normal or malicious. Reference [25] proposed a method based on SVM, assisted by Genetic Algorithm (GA) and Kernel Principle Component Analysis (KPCA). In this method, the authors used the KPCA to reduce the dimensions of the features, and GA was used to tune the hyperparameters of the SVM. They introduce an improved Kernel Function (N-RBF) to reduce the noise caused by feature differences. Reference [26] used Particle Swarm Optimization(PSO)-BP neural networks and normal entropy metrics in their research to detect the DDoS attack.

Reference [27] improved the DDoS attack detection performance using a trigger mechanism for the first time. This method helps reduce the overhead of the switches and controller. Another hybrid method based on Artificial Neural Networks (ANNs) and DNN was proposed in [28]. To classify the major bot attacks as Emotet, Zbot, Dridex, and Sality, the authors used a combined method of RNN and LSTM [29]. Another hybrid method based on LSTM and CNN was proposed in [30]. They deployed this method in a real testbed which used data gathered from nine commercial IoT devices to detect the attack. In [31], the authors proposed a distributed method based on CNN and LSTM with an additional cloud-based component for detecting DDoS and phishing attacks. Reference [32] introduced a method combining Fuzzy and Taylor-Elephant Herd Optimization (F-TEHO) inspired by Deep Belief Network (DBN) to detect DDoS attacks. This method compromises three modules: feature extraction, selection, and classification. The feature extraction module extracts feature from raw packet data, and then the Holo-entropy method is used to select essential features. Finally, the classification process is performed using the FT-EHO method. Another method based on cognitive-inspired computing with dual address entropy is proposed in [33]. In this method, the dual address entropy is first used to extract the features from the flow table, and then SVM classifies the traffic as normal or attack. This method helps to detect the

TABLE 1. Summary of recent deep learning development for intrusion detection and their limitations.

Study	Algorithm used	Feature Engineering	Flow-based	Packet-based	SDN Environment	Botnet	DDoS
[8]	IE-CNN	×	×	✓	✓	×	✓
[15]	Random Forest, Decision Tree, Bagging, k-Nearest Neighbor, and Support Vector Machine	✓	×	✓	×	✓	×
[17]	Deep autoencoders	×	✓	×	×	✓	✓
[18]	PSO, DNN, DT, and SVM	✓	✓	×	×	✓	✓
[25]	SVM	✓	✓	×	✓	×	✓
[28]	Artificial Neural Networks (ANNs) and Deep Neural Networks (DNNs)	×	✓	×	×	✓	×
[30]	LSTM-CNN	×	×	✓	×	✓	×
[32]	FT-EHO-DBN	×	×	✓	×	×	✓
[33]	SVM	×	✓	×	✓	×	✓
Proposed method	RNN, CNN, MLP, LSTM, and DNN	✓	✓	×	✓	✓	✓

attack at the preliminary stage and restore the usual communication in time. We summarize the recent deep learning-based intrusion detection development in TABLE 1.

III. OVERVIEW OF THE PROPOSED METHOD

To successfully protect the SDN against botnet-based DDoS attacks, detecting and blocking attack flows from the attackers is significant. A flow comprises several packets with the same information (source and destination IP address, source and destination port number, protocol, and other features are listed in TABLE 3). In an attack flow, the source IP address belongs to the attacker. Assume we have N number of flow samples and y classes. Let the flow samples are $\boldsymbol{X} = \{\boldsymbol{F_1},\boldsymbol{F_2},\boldsymbol{F_3},\ldots\ldots\ldots,\boldsymbol{F_N}\}\,\boldsymbol{\varepsilon}\mathbb{R}^{\boldsymbol{d}\times\boldsymbol{N}}$, where $\boldsymbol{F_i}$ represents the ith flow, $\boldsymbol{d}$ is the number of original features in a flow, and $\boldsymbol{N}$ represents the total number of flows. In a $\boldsymbol{F_i}$ flow, the actual tables are defined as $\boldsymbol{y_i} = \{\boldsymbol{0},\boldsymbol{1}\}$. We aim to develop an end-to-end method to predict a label as an actual label $(\boldsymbol{y_{pred(i)}} = \boldsymbol{y_i})$.

This section discusses a feature selection and deep learning-based comparative study for flow classification. It would help classify the normal and attack flows and improve efficiency and accuracy. Fig.1 shows the operation phases of the proposed research. First, the incoming packets are placed into a pretty table. Second, the features according to TABLE 3 are computed and extracted for each packet flow individually. Third, the optimal features are selected based on the optimal threshold values of the feature weights. Finally, the selected features are combined into a subset, passing this subset as input to the DL classifiers (e.g., RNN, CNN, MLP, LSTM, and DNN) for flow classification.

A. BRIEF DESCRIPTION OF DL METHODS AND HYPER-PARAMETERS SETTING

This section gives an overview of deep learning methods.

1) RECURRENT NEURAL NETWORK (RNN)

RNNs [34] are known as a dynamic type of feed-forward neural network. This network can learn the sequential data overtime-steps. In conventional networks, each unit output depends on the current input with no dependency between the previous or output of the same unit. However, time-series-based applications rely on sequential data; each current sample depends on the last sample. Therefore, more conventional networks are needed for these applications. RNNs use a time series mechanism to handle these issues. The sequential process for each time step is computed as follows:

$$\begin{aligned} S_t &= f\left(W_{sx,x_t} + W_{ys,s_{t-1}} + b_s\right) \\ Y_t &= g\left(W_{ys,s_{t-1}} + b_y\right) \end{aligned} \tag{1}$$

where f and g represent the encoder and decoder functions, W_{sx,x_t} are current inputs, and $W_{ys,s_{t-1}}$ is the output of the

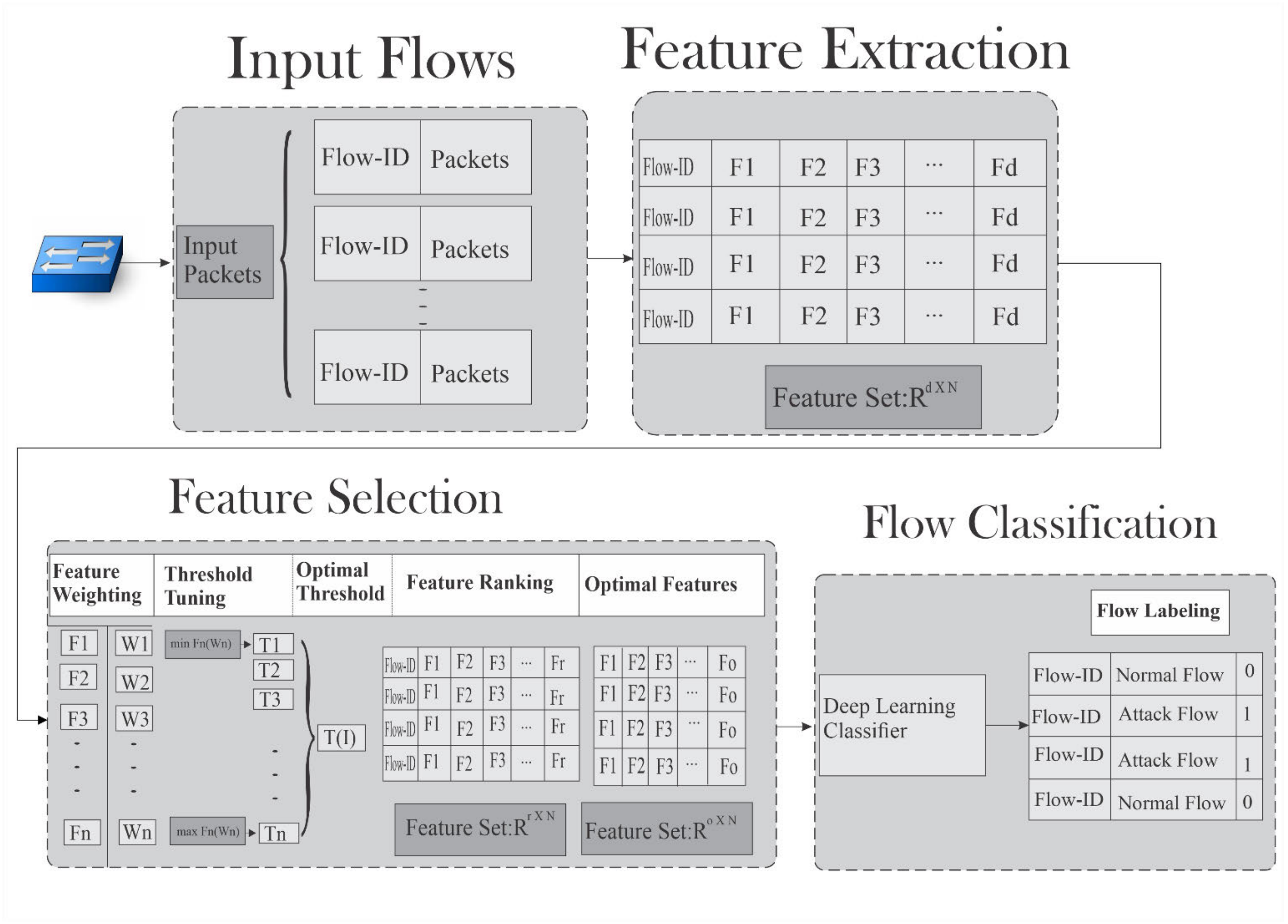


FIGURE 1. Flow diagram of the proposed method. Phase 1 converts the packet data into flows. Phase 2 computes and extracts the features from each flow. Phase 3 selects the optimal features subset using feature weighting and tuning methods. Phase 4 trains the DL methods with optimal feature subsets and classifies the flow as normal or attack.

previous layer at time-step t, b_s, and b_y are bias. The parameter setting is given in TABLE 2.

2) CONVOLUTIONAL NEURAL NETWORKS (CNNS)

CNNs were introduced to handle the intensive connections between the DNN layers. CNNs use nonlinear mappings to train their layers and can classify high-dimensional input data into multiple classes at the output layers. A simple CNN consists of an input layer, convolutional layers, pooling layers, and an activation function. Convolutional layers contain different filters, which decompose the input data into smaller dimensional slices, and feature maps are also produced by the filters. The pooling layer performs subsampling over the feature maps and reduces the dimensions of the maps [35]. The convolutional layers of the CNNs can receive multi-dimensional input data. The convolutional calculation process is formulated as follows:

$$f\left(\sum_{\alpha=1}^{m} w^{j}_{\alpha,k} r^{l-1,j}_{i+(k-1)\times s+\alpha-1} + b_j\right) \qquad (2)$$

where $X^{i,j}_{i,k}$ is one of ith unit of feature map j of kth section of layer lth, s represents the range of that section, and f is non-linear mapping function.

Furthermore, CNNs are suitable for multi-dimensional data problems. CNNs need fewer parameters than other deep networks, which speeds up the learning process and reduces complexity. CNNs' use for intrusion detection is enhanced due to their capability to covenant with complex data [36]. The CNN's hyper-parameters setting for our experiments is given in TABLE 2.

3) MULTI-LAYER PERCEPTRON

In MLP [37], the feed-forward maps the input data to the corresponding output. In our experiments, a fully connected network connects each layer with the next and previous layers. To reduce the difference between the actual and obtained outputs, the connection weights are used during the training of MLP. Each node's output is considered a weighted unit followed by an activation function that discriminates the linear and nonlinearly separable data. Generally, the following mathematical expression is used to obtain the output activation $\alpha^{(l+1)}$ at layer $l+1$.

$$\alpha^{(l+1)} = \sigma\left(w^{(l)}\alpha^{(l)} + b^{(l)}\right) \qquad (3)$$

Here, l represents the layer number, $w^{(l)}$ and $b^{(l)}$ are weights and biases at a particular layer. Where σ is an activation function (hyperbolic tangent, sigmoid, rectified linear units, etc.). The hyper-parameter setting is given in TABLE 2.

4) DEEP NEURAL NETWORKS (DNNS)

DNN is an advanced variant of MLP, a Feed Forward Neural Network (FNN) type with two or more layers [12]. It consists of one input and output layer and more than one hidden layer. Every layer in the network has multiple neurons fully connected with other neurons in a forward direction. Mathematically DNN method is defined as $Q : \mathbb{R}^m \times \mathbb{R}^n$. The input vector $x = x_1, x_2, x_3 \ldots\ldots\ldots x_m$ with size m, and the output vector is $Q(x)$ with size n. The computational process on a hidden layer can be defined as follows:

$$h_j\left(x_j^{l+1}\right) = f\left(Z_{ij} + b_j^{l+1}\right)$$
$$Z_{ij} = x_i^l w_{ij}^{(l.l+1)} \tag{4}$$

In Eq. (4), x_i^l is a current neuron at layer l with activation function i, and Z_{ij} is a contribution of neuron i at layer l to the activation neuron j at layer $l+1$. The $w_{ij}^{(l.l+1)}$, b_j^{l+1} are the weights and bias of a neuron j respectively, where f is a non-linear activation function. For our experiments, the hyper-parameters setting of the DNN is given in TABLE 2.

5) LONG SHORT-TERM MEMORY (LSTM)

The network traffic's nature is sequential, making the LSTM suitable for network intrusion detection problems [38]. In our experiments, the records or adjacent flows of the dataset have been grouped to make input sequences for the LSTM method because they occur in a temporal sequence. The method's output is a label sequence, one label of each flow in an input sequence. The expectation from an LSTM network is that it will learn the temporal relationship among adjacent flows. The hyper-parameter setting of the LSTM is given in TABLE 2. The training process of the deep learning methods for the detection of botnet-based DDoS attacks is shown in Algorithm 1.

TABLE 2. Hyper-parameters settings of DL methods.

Hyper-parameters settings	RNN	CNN	MLP	DNN	LSTM
Number of hidden layers	1	1	1	3	1
Hidden layer neurons	128	128	128	128	128
Activation function in hidden layers	-	relu	relu	relu	-
Activation function at the output layer	sigmoid	sigmoid	sigmoid	sigmoid	sigmoid
Learning rate (p)	0.1	0.1	0.1	0.1	0.1
Optimizer	adam	adam	adam	adam	adam
Epochs	20	20	20	20	20
Batch size	32	32	32	32	32

Algorithm 1 Training of Deep Learning Methods for Attack Detection

```
Input: dataset: data-subsets, learning rate: learning_rate(p), optimizer: adam, training rounds: epochs
Output: accuracy: acc
1: Set the parameters of the DL classifiers according to TABLE 2.
2: Initialize a parameter matrix using random values
3:  for each training round, do
4:      select a subset from the dataset to form a batch for the training
5:      if the number of training rounds%100=0then
6:          return acc
7:      end if
8:      input batch and calculate predicted value y
9:  Calculate the loss value between the actual value of the label y and the predicted value y
10: Calculate loss value for gradient descent direction with optimizer adam
11: Update the parameter matrix with the gradient descent direction and learning rate (p)
12:     if training rounds reach the epochs, then
13:         stop training of DL classifier
14:     end if
15: end for
```

B. FEATURE EXTRACTION AND LABELING

The dataset is captured in a pcap format in the software-defined network environment. We used the CIC Flow Meter V4 (version 4) [39], [40] to convert the dataset from pcap format to CSV to train deep learning methods. CIC Flow Meter V4 takes the captured file and outputs bidirectional network flow statistics with 83 features. A network flow is a unidirectional sequence of packets traveling from source to destination with a specific protocol over a period. The converted dataset has a total of 89,632 flow records with 83 features. Furthermore, the dataset has 48,390 normal flows and 41,242 attack flows with the same 83 features. The descriptions of all the extracted features are given in TABLE 3.

C. DATA PRE-PROCESSING

The features flow ID, source and destination IPs, source and destination ports, protocol, and timestamp are removed from the dataset to generalize it. Removing the features above-mentioned precludes the DL methods from attributing specific IPs, ports, and protocols as attack nodes, thereby adopting the generalized DL methods'. After that, the dataset is appraised for infinite and missing values. After normalization, the dataset is encoded (the normal and attack flows were labeled as 0 and 1, respectively). As mentioned in the above section, the dataset has 48,390 normal and 41,242 attack flows. The imbalance is retained in the dataset to imitate the

TABLE 3. List of extracted features and their weights.

Feature Code and Names	Description	Assigned Weights	Feature Code and Names	Description	Assigned Weights
F1. Flow ID	The assigned ID of a Flow	-	F43. Fwd Pkts/s	Number of forwarding packets per second	6.17
F2. Src IP	Source IP address	-	F44. Bwd Pkts/s	Number of backward packets per second	5.25
F3. Src Port	Source port number	-	F45. Pkt Len Min	The minimum length of a flow	1.17
F4. Dst IP	Destination IP address	-	F46. Pkt Len Max	The maximum length of a flow	2.53
F5. Dst Port	Destination port number	-	F47. Pkt Len Mean	The mean of a flow length	1.22
F6. Protocol	Type of protocol	-	F48. Pkt Len Std	The Standard deviation of a flow length	5.12
F7. Timestamp	Capture time	-	F49. Pkt Len Var	The minimum inter-arrival time of a packet	1.05
F8. Flow Duration	The duration of a flow	4.51	F50. FIN Flag Cnt	Number of packets with FIN Flag of a flow	2.75
F9. Tot Fwd Pkts	The total number of forwarding packets	2.05	F51. SYN Flag Cnt	Number of packets with SYN Flag of a flow	6.23
F10. Tot Bwd Pkts	The total number of backward packets	4.30	F52. RST Flag Cnt	Number of packets with RST Flag of a flow	1.94
F11. Tot Len Fwd Pkts	The total length of forwarding packets	4.70	F53. PSH Flag Cnt	Number of packets with PSH Flag of a flow	1.56
F12. Tot Len Bwd Pkts	The total length of backward packets	9.48	F54. ACK Flag Cnt	Number of packets with ACK Flag of a flow	2.07
F13. Fwd Pkt Len Max	The maximum length of forwarding packets	1.58	F55. URG Flag Cnt	Number of packets with URG Flag of a flow	0.00
F14. Fwd Pkt Len Min	The minimum length of forwarding packets	2.92	F56. CWE Flag Count	Number of packets with CWE Flag of a flow	0.00
F15. Fwd Pkt Len Mean	The length Mean of the forwarding packets	2.13	F57. ECE Flag Cnt	Number of packets with ECE Flag of a flow	0.00
F16. Fwd Pkt Len Std	The length variance of the forwarding packets	5.70	F58. Down/Up Ratio	Download and Upload ratio	4.82
F17. Bwd Pkt Len Max	The maximum length of backward packets	1.16	F59. Pkt Size Avg	The average packet size	2.10
F18. Bwd Pkt Len Min	The minimum length of the backward packets	3.78	F60. Fwd Seg Size Avg	The average packet size observed in the forward direction	2.13
F19. Bwd Pkt Len Mean	The length mean of the backward packets	2.81	F61. Bwd Seg Size Avg	The average packet size observed in the backward direction	2.81
F20. Bwd Pkt Len Std	The length variance of the backward packets	3.38	F62. Fwd Byts/b Avg	The average number of bytes per bulk in the forward direction	0.00
F21. Flow Byts/s	Number of bytes of the flow per second	1.01	F63. Fwd Pkts/b Avg	The average number of packets per bulk in the forward direction	0.00
F22. Flow Pkts/s	Number of packets of the flow per second	1.55	F64. Fwd Blk Rate Avg	The average bulk rate in the forward direction	0.00
F23. Flow IAT Mean	The mean of packet flow inter-arrival time	1.81	F65. Bwd Byts/b Avg	The average number of bytes per bulk in the backward direction	0.00
F24. Flow IAT Std	The standard deviation of packet flow inter-arrival time	3.39	F66. Bwd Pkts/b Avg	The average number of packets per bulk in the backward direction	0.00
F25. Flow IAT Max	The maximum packet flow inter-arrival time	1.16	F67. Bwd Blk Rate Avg	The average bulk rate in the backward direction	0.00
F26. Flow IAT Min	The minimum packet flow inter-arrival time	1.82	F68. Subflow Fwd Pkts	Number of packets in a sub-flow in the forward direction	2.05
F27. Fwd IAT Tot	The total time interval of the forward packets	4.18	F69. Subflow Fwd Byts	Number of bytes in a sub-flow in the forward direction	4.70
F28. Fwd IAT Mean	The mean time interval of the forward packets	1.09	F70. Subflow Bwd Pkts	Number of packets in a sub-flow in the backward direction	4.30
F29. Fwd IAT Std	The standard deviation time interval of the forward packets	1.30	F71. Subflow Bwd Byts	Number of bytes in a sub-flow in the backward direction	9.48
F30. Fwd IAT Max	The maximum time interval of the forward packets	6.00	F72. Init Fwd Win Byts	Number of bytes sent in the initial window in the forward direction	0.00
F31. Fwd IAT Min	The minimum time interval of the forward packets	1.86	F73. Init Bwd Win Byts	Number of bytes sent in the initial window in the backward direction	1.16
F32. Bwd IAT Tot	The total time interval of the backward packets	5.72	F74. Fwd Act Data Pkts	Number of packets with at least 1 byte of TCP data payload in the forward direction	1.05
F33. Bwd IAT Mean	The mean time interval of the backward packets	9.59	F75. Fwd Seg Size Min	The minimum segment size observed in the forward direction	0.00
F34. Bwd IAT Std	The standard deviation time interval of the backward packets	8.10	F76. Active Mean	The mean of time when a flow was active before becoming idle	1.48
F35. Bwd IAT Max	The maximum time interval of the backward packets	9.67	F77. Active Std	The standard deviation time when a flow was active before becoming idle	3.02

TABLE 3. ***(Continued.)*** **List of extracted features and their weights.**

F36. Bwd IAT Min	The minimum time interval of the backward packets	1.27	F78. Active Max	The maximum time when a flow was active before becoming idle	1.17
F37. Fwd PSH Flags	Number of times the PSH flag was set in packets traveling in the forward direction	0.00	F79. Active Min	The minimum time when a flow was active before becoming idle	7.51
F38. Bwd PSH Flags	Number of times the PSH flag was set in packets traveling in the backward direction	1.56	F80. Idle Mean	The mean of time when a flow was idle before becoming active	6.81
F39. Fwd URG Flags	Number of times the URG flag was set in packets traveling in the forward direction	0.00	F81. Idle Std	The standard deviation time when a flow was idle before becoming active	1.61
F40. Bwd URG Flags	Number of times the URG flag was set in packets traveling in the backward direction	0.00	F82. Idle Max	The maximum time when a flow was idle before becoming active	1.07
F41. Fwd Header Len	The total number of bytes used for headers in the forward direction	2.52	F83. Idle Min	The minimum time when a flow was idle before becoming active	5.27
F42. Bwd Header Len	The total number of bytes used for headers in the backward direction	2.09			

real-time scenarios for the DL methods, where the number of normal flows is always greater than attack flows.

D. FEATURE SELECTION

Although some of the features mentioned above are important for detecting attack flows, and some may have little effect on the classification results, they upsurge the computational cost and time. We must select the optimal features that can discriminate the normal, and attack flows to boost and be accurate in classification. In our experiments, we use a feature selection method that does not change the original features but selects an optimal subset of features from a given set of features. Let $\{F_1, F_2, F_3, \ldots\ldots\ldots F_d$ are d features of X, where d represents the high-dimension features. So, to select the optimal flow features for attack recognition, we assume a method that can convert the high-dimension features d to low-dimension features $r(r < d)$. In our experiments, the feature selection method is based on two principles: (i) Feature Weighting and (ii) Threshold Tuning. The feature selection procedure is shown in Algorithm 2.

Algorithm 2 Procedure of Feature Selection

```
Input: Feature Set F = {F1, F2, F3, F4, ......... Fd}, the threshold
of weight $\alpha$, and the number of selected features r;
Output: Selected feature set F'
// Calculate the weights of the features
1:  F' Ø $\leftarrow$
2:  for (i = 0; i < d; i++) do
3:      r(i) $\leftarrow$ compute weights ($F_i$)
4:      if (|r(i)| < $\alpha$) then
5:          remove feature $F_i$;
6:      else
7:          FA [] = $F_i$
8:      end if
9:  end for
10: F' $\leftarrow$ store (FA [])
11: return F'
```

E. FEATURE WEIGHTING

Our experiments used iterative wrapper-based feature selection with SVM to convert the whole dataset into five subsets with optimal features. The SVM assigns the weights of all the features based on their importance in predicting the output. The features with higher weights are considered important for detecting the attack. The assigned absolute value of the weights by the SVM for distinct features is given in TABLE 3. A threshold value derived from the threshold tuning method for the weights is set for selecting features during each iteration. The features with weights greater than or equal to the threshold $(F_n(w_n) \geq \alpha)$ were selected and converted into a subset. Then, subsets with optimal features were given as input to the DL classifiers.

F. THRESHOLD TUNING

A simple threshold tuning method determines the optimal threshold value based on feature weights. The tuning method takes the feature weights from minimum to maximum and returns a threshold value between the range. The optimal threshold values can be computed as the threshold that can reduce the dimension of the features.

IV. EXPERIMENTAL ENVIRONMENT

The experiments are conducted in the Mininet virtual environment [41] with a POX controller [42]. We used Mininet version 2.3.2 in our experiments, which supports Open Virtual Switches (OVS) [43]. Usually, the SDN-based network emulations are performed in Mininet, and OVS is an open-source virtual machine that supports OpenFlow protocols. The POX is an interface-rich SDN controller that authorizes the network and research community to cultivate control and network applications using Python.

The operating system and hardware settings are Intel Core i7, 8GB RAM, and Windows 10 operating system. The deep learning classifiers are implemented in Python language with the Keras framework. A customized centralized network topology based on SDN is built in Mininet to conduct experiments. A complex tree network structure is adopted in this topology, and the SDN controller is installed in the control layer that centrally controls the hosts and switches at the data layer. The application layer consists of four modules: flow statistics collector, feature extractor, DL classifier, and

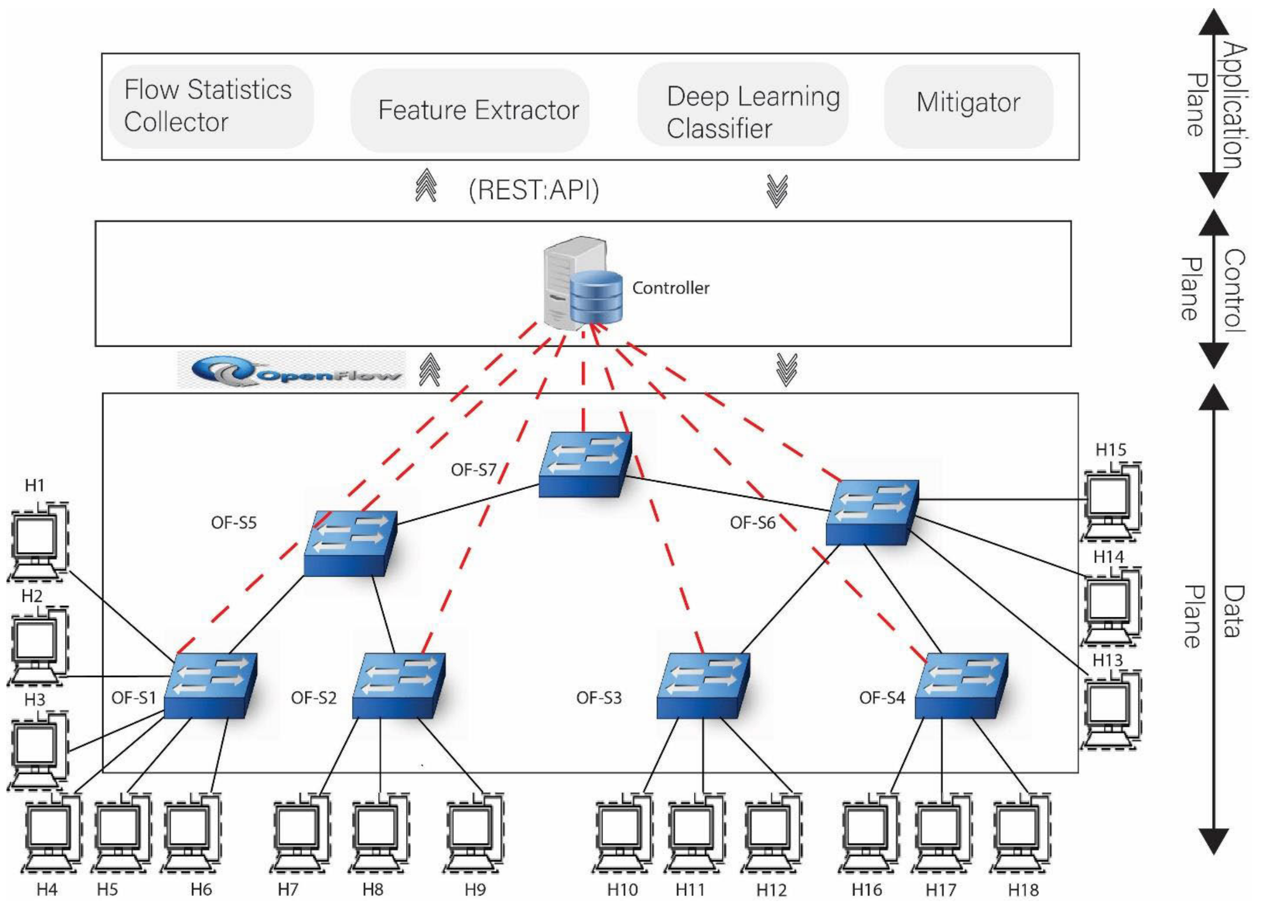


FIGURE 2. SDN-based experimental network topology.

mitigator. Furthermore, the network topology consists of a controller and seven OpenFlow switches. The switch S1 is connected to six hosts, and all the other switches with three hosts. The experimental network topology is shown in Fig. 2.

After the successful development of the topology, we execute the "ping" command on all hosts to verify the access of all hosts to each other. We select H2 as a bot master, and H3, H4, H5, and H6 are bots, while H13 is our target server. The other hosts are used to generate legitimate or background traffic. The controller controls the whole network and detects the attack, while the switches are used to forward the traffic. The relevant settings related to topology are given in TABLE 4.

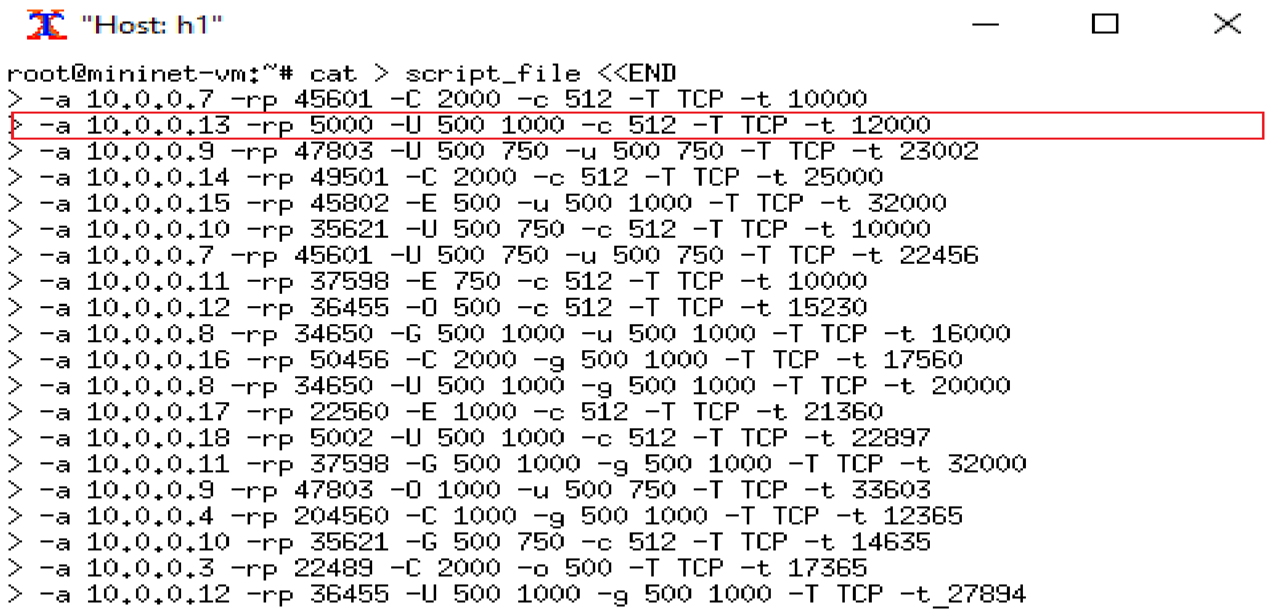
A. DESIGN OF ATTACK AND BACKGROUND TRAFFIC

Our experiments used Python scripts to launch botnet-based DDoS attacks in an SDN environment. Once the topology is set up, we execute "target.py" on the host H13 to set it as a target server. After that, "botmaster.py" is executed on H2, and "bot.py" is run on H3, H4, H5, and H6. Specific ports for the bots were created using the concept of socket programming. The bots wait on their designated ports for the instructions of the botmaster. The botmaster sends the date and time for the attack to all the bots and instructs them to be ready. Once the date and time of all bots are matched with the botmaster's instructed time, they send the attack traffic to the target server. The duration of the attack was 14.26 minutes.

We used a Distributed Internet Traffic Generator (D-ITG) [44] to generate normal or background traffic. The ITGSend command on each host is executed to send traffic and ITGRecv to accept traffic. We used the D-ITG-2.8.1-r1023 version of D-ITG in our experiments. To make the background traffic near to real traffic, we inject more than 200 flows into the network as background flows. The transmission rate of each flow follows constant, uniform, exponential, Poisson, and gamma distributions with TCP protocol. We also variate the packet size for each flow through constant, uniform, exponential, Poisson, and gamma distributions. An example of flow rules to generate normal traffic by the D-ITG is shown in Fig. 3. In Fig. 3, each row has a flow rule to generate the background traffic, and these rules are executed on different hosts to generate and accept traffic. For instance, taking the red box, the third background flow follows the second background flow, and so on. The destination host is H13 (IP address 10.0.0.13 with port 5000).

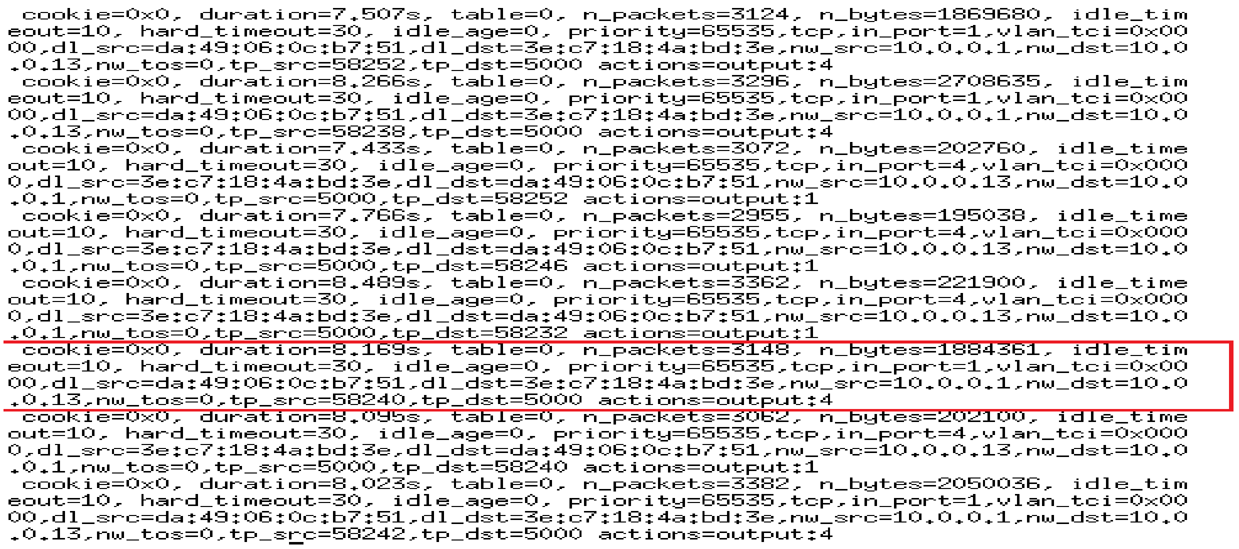
TABLE 4. The setting of addresses and device ports.

Device	Networks Port	Address	State	Device	Networks Port	Address	State
Controller	eth0	127.0.0.1	-	H10	eth0	10.0.0.10	Normal
H1	eth0	10.0.0.1	Normal	H11	eth0	10.0.0.11	Normal
H2	eth0	10.0.0.2	Bot-master	H12	eth0	10.0.0.12	Normal
H3	eth0	10.0.0.3	Bot	H13	eth0	10.0.0.13	Target Server
H4	eth0	10.0.0.4	Bot	H14	eth0	10.0.0.14	Normal
H5	eth0	10.0.0.5	Bot	H15	eth0	10.0.0.15	Normal
H6	eth0	10.0.0.6	Bot	H16	eth0	10.0.0.16	Normal
H7	eth0	10.0.0.7	Normal	H17	eth0	10.0.0.17	Normal
H8	eth0	10.0.0.8	Normal	H18	eth0	10.0.0.18	Normal
H9	eth0	10.0.0.9	Normal				



```
"Host: h1"
root@mininet-vm:~# cat > script_file <<END
> -a 10.0.0.7 -rp 45601 -C 2000 -c 512 -T TCP -t 10000
> -a 10.0.0.13 -rp 5000 -U 500 1000 -c 512 -T TCP -t 12000
> -a 10.0.0.9 -rp 47803 -U 500 750 -u 500 750 -T TCP -t 23002
> -a 10.0.0.14 -rp 49501 -C 2000 -c 512 -T TCP -t 25000
> -a 10.0.0.15 -rp 45802 -E 500 -u 500 1000 -T TCP -t 32000
> -a 10.0.0.10 -rp 35621 -U 500 750 -c 512 -T TCP -t 10000
> -a 10.0.0.7 -rp 45601 -U 500 750 -u 500 750 -T TCP -t 22456
> -a 10.0.0.11 -rp 37598 -E 750 -c 512 -T TCP -t 10000
> -a 10.0.0.12 -rp 36455 -O 500 -c 512 -T TCP -t 15230
> -a 10.0.0.8 -rp 34650 -G 500 1000 -u 500 1000 -T TCP -t 16000
> -a 10.0.0.16 -rp 50456 -C 2000 -g 500 1000 -T TCP -t 17560
> -a 10.0.0.8 -rp 34650 -U 500 1000 -g 500 1000 -T TCP -t 20000
> -a 10.0.0.17 -rp 22560 -E 1000 -c 512 -T TCP -t 21360
> -a 10.0.0.18 -rp 5002 -U 500 1000 -c 512 -T TCP -t 22897
> -a 10.0.0.11 -rp 37598 -G 500 1000 -g 500 1000 -T TCP -t 32000
> -a 10.0.0.9 -rp 47803 -O 1000 -u 500 750 -T TCP -t 33603
> -a 10.0.0.4 -rp 204560 -C 1000 -g 500 1000 -T TCP -t 12365
> -a 10.0.0.10 -rp 35621 -G 500 750 -c 512 -T TCP -t 14635
> -a 10.0.0.3 -rp 22489 -C 2000 -o 500 -T TCP -t 17365
> -a 10.0.0.12 -rp 36455 -U 500 1000 -g 500 1000 -T TCP -t_27894
```

FIGURE 3. An example of flow rules to generate normal traffic.



```
 cookie=0x0, duration=7.507s, table=0, n_packets=3124, n_bytes=1869680, idle_tim
eout=10, hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=1,vlan_tci=0x00
00,dl_src=da:49:06:0c:b7:51,dl_dst=3e:c7:18:4a:bd:3e,nw_src=10.0.0.1,nw_dst=10.0
.0.13,nw_tos=0,tp_src=58252,tp_dst=5000 actions=output:4
 cookie=0x0, duration=8.266s, table=0, n_packets=3296, n_bytes=2708635, idle_tim
eout=10, hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=1,vlan_tci=0x00
00,dl_src=da:49:06:0c:b7:51,dl_dst=3e:c7:18:4a:bd:3e,nw_src=10.0.0.1,nw_dst=10.0
.0.13,nw_tos=0,tp_src=58238,tp_dst=5000 actions=output:4
 cookie=0x0, duration=7.433s, table=0, n_packets=3072, n_bytes=202760, idle_time
out=10, hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=4,vlan_tci=0x000
0,dl_src=3e:c7:18:4a:bd:3e,dl_dst=da:49:06:0c:b7:51,nw_src=10.0.0.13,nw_dst=10.0
.0.1,nw_tos=0,tp_src=5000,tp_dst=58252 actions=output:1
 cookie=0x0, duration=7.766s, table=0, n_packets=2955, n_bytes=195038, idle_time
out=10, hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=4,vlan_tci=0x000
0,dl_src=3e:c7:18:4a:bd:3e,dl_dst=da:49:06:0c:b7:51,nw_src=10.0.0.13,nw_dst=10.0
.0.1,nw_tos=0,tp_src=5000,tp_dst=58246 actions=output:1
 cookie=0x0, duration=8.489s, table=0, n_packets=3362, n_bytes=221900, idle_time
out=10, hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=4,vlan_tci=0x000
0,dl_src=3e:c7:18:4a:bd:3e,dl_dst=da:49:06:0c:b7:51,nw_src=10.0.0.13,nw_dst=10.0
.0.1,nw_tos=0,tp_src=5000,tp_dst=58232 actions=output:1
 cookie=0x0, duration=8.169s, table=0, n_packets=3148, n_bytes=1884361, idle_tim
eout=10, hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=1,vlan_tci=0x00
00,dl_src=da:49:06:0c:b7:51,dl_dst=3e:c7:18:4a:bd:3e,nw_src=10.0.0.1,nw_dst=10.0
.0.13,nw_tos=0,tp_src=58240,tp_dst=5000 actions=output:4
 cookie=0x0, duration=8.095s, table=0, n_packets=3062, n_bytes=202100, idle_time
out=10, hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=4,vlan_tci=0x000
0,dl_src=3e:c7:18:4a:bd:3e,dl_dst=da:49:06:0c:b7:51,nw_src=10.0.0.13,nw_dst=10.0
.0.1,nw_tos=0,tp_src=5000,tp_dst=58240 actions=output:1
 cookie=0x0, duration=8.023s, table=0, n_packets=3382, n_bytes=2050036, idle_tim
eout=10, hard_timeout=30, idle_age=0, priority=65535,tcp,in_port=1,vlan_tci=0x00
00,dl_src=da:49:06:0c:b7:51,dl_dst=3e:c7:18:4a:bd:3e,nw_src=10.0.0.1,nw_dst=10.0
.0.13,nw_tos=0,tp_src=58242,tp_dst=5000 actions=output:4
```

FIGURE 4. Partial screenshot of the flow table of an OpenFlow switch during normal traffic.

The number of packets sent is uniformly distributed from 500 to 1000, and the size of transmitted packets is 512 bytes. This is a TCP protocol-based flow, and the duration of the traffic is 12000 milliseconds. A partial snapshot of the flow table in the OpenFlow switch is shown in Fig. 4. Fig. 4 shows the screenshot when the background traffic runs and highlights it with the red box. In Fig. 4, the red box shows a flow entry of H1 (IP address 10.0.0.1) to H13 (IP address 10.0.0.13) with destination port 5000. It is observed that the background traffic has been successfully inoculated into the network.

B. MITIGATION STRATEGY

The controller is responsible for regularly analyzing the traffic flows to protect itself from malicious attacks. For instance, when the controller detects any attack, it quickly needs to activate the defense shield to reduce the possible impact of the attack and ensure the network's normal operations. In previous studies [1], [6], the authors commonly use modify or block attack traffic concepts in their mitigation methods. However, many malicious flow entries remain in the switches after blocking the attack flows, which affects the normal forwarding process of the network and the resources of the controller and switches. So, we adopted a graph theory and dynamic flow deletion-based mitigation strategy. In our mitigation strategy, first, the controller requests corresponding OpenFlow switches for the flow statistics in every ΔT (e.g., 5 sec). After receiving the flow statistics, the controller passes the features information shown in Fig.5 to the already trained DL classifier (e.g., CNN) to predict whether the flow is normal or attacked. Once the classifier inside the controller detects any attack flow, it informs the controller. Then the controller creates a gray list $\mathrm{S_g}$ in the database and puts the incoming flows of the switches with attack flows for further analysis to reduce the chance of killing the normal packet and continue the normal operation of the remaining network. After that, the controller redirects the $\mathrm{S_g}$ flows to the classifier to detect the attack. Here we set a counter that starts calculating the number of attack flows labeled by the classifier and set a limit for attack flows (e.g., c$\geq$10). Simultaneously, the controller creates two new lists: delete $\mathrm{S_d}$ and block $\mathrm{S_b}$ in the database. The delete list contains the attack flows that need to be deleted from the switches with the same attack flows, and the block list maintains a list of hosts involved in the attack for future use. The controller uses its host tracker features to fetch the attacking hosts' information (e.g., MAC or IP address, TCP or UDP port number, entry port, etc.). When the counter reaches its set limit, we use the graph theory concept derived from [45] to discover the attacking path in the network. Here, finding the network path that passes the attack flow and locating the switches through which the attack flows enter the network is important. The attacking path using the concept of graph theory can be formulated as follows:

$$E_{i,j} = \sum (s_i, r_i) \rightarrow \left(s_j, r_j\right), where\ s_i, s_j \varepsilon\ S_{attack} \tag{5}$$

In the above equation, the $E_{i,j}$ is the edge (e.g., the attacking path), and S_{attack} is a set of switches through botnet-based DDoS attack pass. When the botnet-based DDoS attack passes through both s_i and s_j switches and the forwarding rules are met, we can say that the edge between the (si, sj) hop in the attack path. So, the main aim of our mitigation strategy is to find the attack path to adopt a more targeted dropping strategy for malicious flows and not kill the normal flows by mistake.

According to the network traffic characteristics, we make an assumption (e.g., the closer the hop is to the attack source on an attack path, the greater the proportion of the attack traffic in the link traffic). After successfully finding the attack path, there is now a need to adopt an effective dropping strategy for attack flows. Generally, most attack flows exit in that switches from where the attack flows enter, so their dropping rate must be larger to mitigate the attack effectively. For instance, we can be called these edge switches and other intermediate switches. The dropping rate at intermediate switches keeps smaller than at edge switches to avoid killing normal flows. Different indicators can be used to calculate dropping rates for switches. As the switch has only normal flows, there is no need to set the dropping rate. So, we can use the following formula derived from [45] to calculate the dropping rate for a switch:

$$r_{edge} = k(\Delta H, \Delta N) \tag{6}$$

where ΔH is the difference in entropy of source IP addresses of packets per unit of time, and ΔN is the difference in the number of packets passing through a switch per unit of time. Finally, after finding the switch closer to the attacking source, the controller sends the "OFPFC_ADD" message to that switch and inserts new flow entries in the switch flow table. The corresponding starts to drop the attack flows according to the delete list $\mathrm{S_d}$ based on the calculated drop rates. If the dropping rate for any host reaches 100%, then the information of that host is placed in the backlist, and the host is blocked.

V. EXPERIMENTAL RESULTS

A. EVALUATION METRICS

The following evaluation metrics are commonly used to demonstrate the effectiveness and performance of the DL-based IDS. The performance measures include accuracy, Detection Rate (DR), False Positive Rate (FPR), F1 score, specificity, and precision. The confusion matrix of network anomaly classification is used to compute these metrics. The efficiency of any implemented method can be assessed using these performance measures. In the confusion matrix, True Negatives (TN) are the number of normal records that are accurately detected as normal, True Positives (TP) are the number of attack records that are accurately detected as attacks, False Positives (FP) are normal records which are inaccurately detected as attacks, and False Negatives (FN) are attack records which are inaccurately detected as normal.

All these evaluation metrics can be quantified as follows:

Accuracy: Accuracy is the proportion of the precisely detected number of normal and attack records to all records, which is formulated as follows:

$$Accuracy = \frac{TN + TP}{TN + FP + FN + TP} \tag{7}$$

Detection Rate: DR is a ratio of accurately detected true attack records. It is also known as Sensitivity or recall and

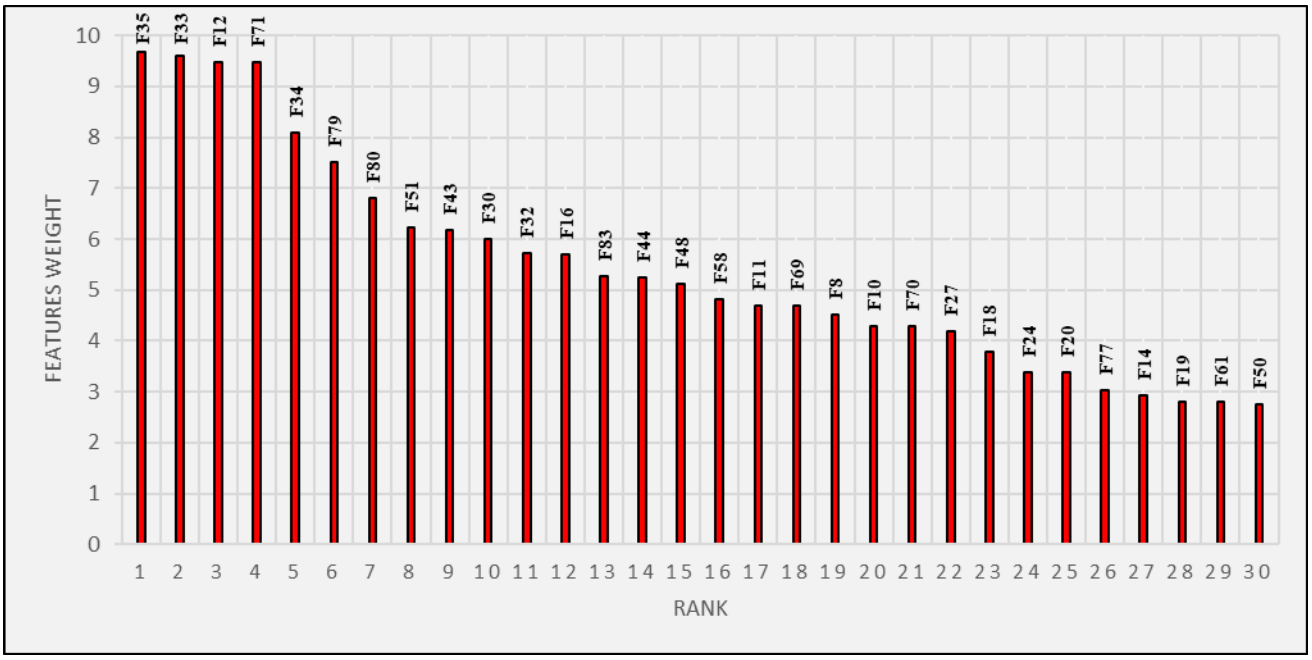


FIGURE 5. List of optimal features in subset-3.

True Positive Rate (TPR). It is computed as:

$$DetectionRate = recall = TPR = \frac{TP}{FN + TP} \tag{8}$$

False Positive Rate: FPR is a ratio of inaccurately identified number of normal records as attack records.

$$FPR = \frac{FP}{TN + FP} \tag{9}$$

Precision: Precision is a ratio of all accurately detected attack records that are really attack records.

$$Precision = \frac{TP}{FP + TP} \tag{10}$$

F1 Score: It is a harmonic mean of both precision and recall. It is considered more efficient than accuracy when the method is trained on an imbalanced dataset. The F1 score for a DL method is computed as follows:

$$F1score = 2 \times \frac{Recall \times Precision}{Recall \times Precision} \tag{11}$$

If the accuracy, DR, precision, and F1 score of any implemented DL method is high, and the same way, If the FPR is low, then that method is regarded as the best method.

B. FEATURE SELECTION RESULTS

We calculated the importance of each feature and split them into five subsets based on feature weighting and threshold tuning methods. The threshold tuning method used the weight values of all features to determine an optimal threshold value. Features with equal or higher weights than the threshold value was selected and placed in a subset. Subset-1 includes all features. For subset-2, the tuning method returns an optimal value of "1.8", selecting 43 features with weights $\alpha \geq 1.80$ to be included in the subset. Subset-3 consists of 30 features with weights $\alpha \geq 2.70$, which was determined as the optimal threshold value. Similarly, for subset-4, 23 features were selected based on $\alpha \geq 3.15$. Lastly, subset-5 includes 15 selected features based on a threshold value of $\alpha \geq 4.90$. For example, Fig. 5 shows the selected features for subset-3.

C. EFFICIENCY OF DL METHODS FOR ATTACK DETECTION

This section aims to verify the efficiency of the DL methods for botnet-based DDoS attack detection in an SDN environment. We simulated and overserved the different classification algorithms' performance with almost identical method structures. We used a confusion matrix to calculate the performance indicators explained in the above section. Fig. 6 shows the general structure of the confusion matrix.

D. STRUCTURAL PERFORMANCE OF METHODS

We compared the performance of five different DL methods to detect the attack traffic. The training set is divided into five subsets based on the optimal features. We adopt almost the same structures (number of hidden layers, number of neurons at hidden layer, activation functions on both hidden and output layer, learning rate, optimizer, and batch size) for all the methods. We observed that the methods produced different results with the same structure and on the same subset of features. The main aim of evaluating the structural performance of the methods is to select the best method without increasing the complexity of the methods.

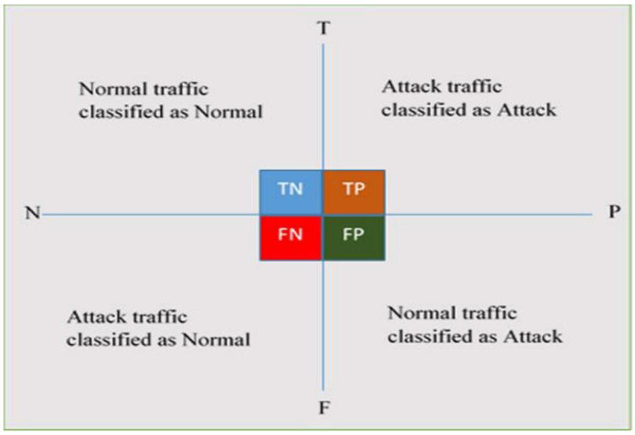

T
Normal traffic classified as Normal
Attack traffic classified as Attack
N
TN
TP
FN
FP
P
Attack traffic classified as Normal
Normal traffic classified as Attack
F

FIGURE 6. The general structure of a confusion matrix for anomaly detection.

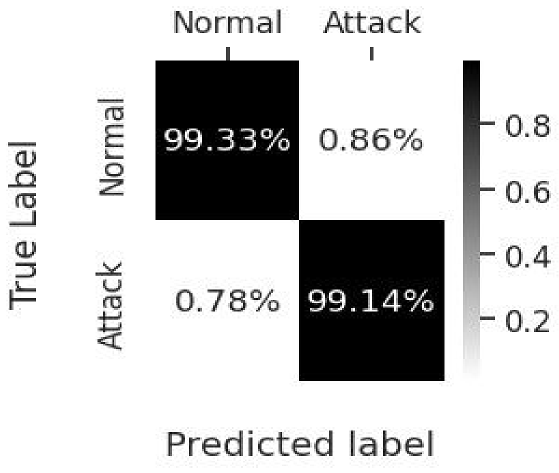
E. CLASSIFICATION PERFORMANCE OF METHODS

The evaluation metrics defined in the above section have been used to successfully measure the prediction rate for both attack and normal states. For most of the DL methods we analyzed, the fluctuations in the output of DL methods are caused by the number of features, which means the prediction becomes effective with optimal features. This paper used five DL algorithms for traffic classification: RNN, CNN, MLP, LSTM, and DNN. These algorithms have different classification capabilities, and we compared their prediction performances. The confusion matrices of all five algorithms for subset-3 features are in Fig. 7. It is observed that all the algorithms RNN, CNN, MLP, LSTM, and DNN shows good abilities for identifying attacks whose underreporting rate (FNR) are 0.78%, 0.40%, 0.59%, 0.87%, and 0.89%, respectively. The recognition ability of all the algorithms to the normal and attack data is about 99%, with a slight difference. The detection methods need high sensitivity because the attack traffic can harm the SDN controller.

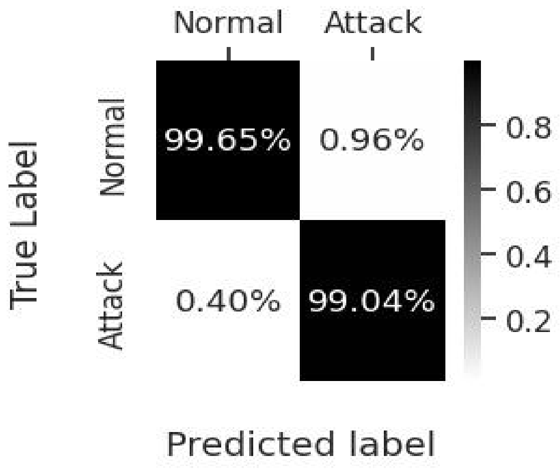
Then, we compare the validation accuracy and loss curves of all the algorithms for subset-3, as shown in Fig. 8. In Fig. 8, it is observed that the accuracy curve of DNN and CNN is more stable and steeper than the other algorithms, which indicates that the performance of DNN and CNN methods is comparatively superior. The validation accuracy of DNN is 99.30%, CNN is 99.37%, MLP is 99.33%, RNN is 99.25%, and LSTM is 99.16%. The Loss curve of all algorithms is shown in Fig. 9. The CNN and DNN have low loss ratio than others. The CNN shows the minimum fluctuation in the loss curve. Comparing the accuracy and loss curves proves that the CNN method is more stable for attack detection than others.

F. COMPARATIVE ANALYSIS OF METHODS PERFORMANCE

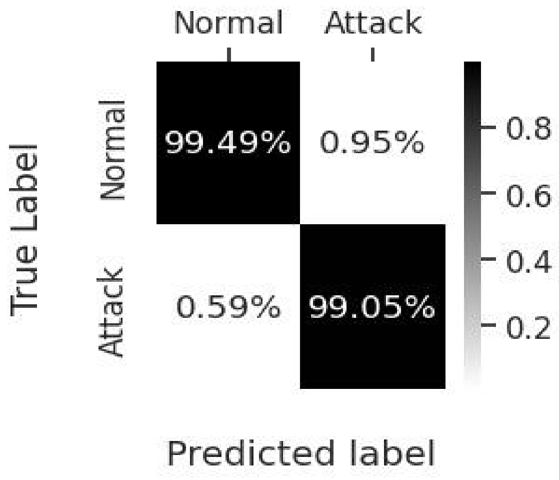
This section compares the performance results of all methods regarding Computational time (training time), accuracy, and detection rate. The performance comparison results are shown in TABLE 5. Regarding the overall trends, the classification performance is improved by reducing the number of features at a certain level. It has been observed in TABLE 5 that the CNN and DNN have the trend to achieve a maximum of 99.43% and 99.53% on subset-2, while the LSTM has minimum accuracy of 97.60% on subset-5. The maximum detection rate achieved by the CNN is 99.60% on subset-3, while the MLP's minimum detection rate is 89.99% on subset-1. For example, we take subset-3 and analyze the accuracy rate, detection rate, and computational time. The accuracy of CNN is 0.12%, 0.04%, 0.07%, and 0.21% higher than that of the other four classifiers (RNN, MLP, DNN, LSTM), respectively. Similarly, the detection rate of CNN is 0.39%, 0.19%, 0.49%, and 0.47% higher than other classifiers. In addition, the computational time of CNN is 39.51s higher than that of MLP, 27.75s higher than that of DNN, 81.29s lower than that of RNN, and 29.85s lower than that of LSTM, but the accuracy is increased by nearly 0.04%-0.21%, and detection rate is improved from 0.19%-0.49%. Since the classifiers are trained offline and are not frequently updated, ensuring an effective detection rate and accuracy, the high training time can be accepted. It means that the detection methods require high accuracy with reasonable training time. We also observed that the CNN method performed better on subset-3 with 30 features than on subset-4 and subset-5. The accuracy of CNN is 99.29% with subset-4 features and 99.26% with subset-5 features. Although these subsets have minimum features compared to subset-3, the training time of CNN with subset-3 is 3.98s and 20.53s lower than others, the accuracy of CNN is 0.08%, 0.11%, and the detection rate is 0.25% and 0.53% higher. Furthermore, the performance results of all classifiers with all five subsets in terms of other parameters such as precision, F1 score, True Positive Rate (TPR), and False Positive Rate (FPR) are shown in Fig. 10. RNN achieves maximum precision rate with all features set which are 99.29%. The minimum is 99.11% on subset-5, CNN attained a maximum of 99.37% with all features, and a minimum is 99.03% with subset-3, MLP achieved a maximum of 99.94% with all features and a minimum is 98.98% with subset-5, DNN achieved maximum 99.75% with all features and minimum is 99.16% with subset-5. LSTM attained a maximum of 99.37% with all features, and a minimum is 97.36% with subset-5. The maximum F1 score of RNN is 99.22% with subset-2 features and 98.57% with all features; CNN achieved a maximum F1 score of 99.37% with subset-2 and a minimum of 98.61% with all features set, MLP attained a maximum of 99.34% with subset-2 and minimum 94.70% with all features, DNN achieved maximum 99.48% with subset-2 and minimum 98.50% with all features, LSTM has maximum 99.09% with subset-3 and minimum 97.40% with subset-5. Hence, the RNN has a maximum TPR of 99.21% with subset-3 features and a minimum of 97.86% with all features, CNN achieved a maximum TPR of 99.60% with subset-3 features and a minimum of 97.87 with all features set, MLP has maximum 99.45% with subset2 and minimum 89.99 with all features, DNN has maximum 99.49% with subset-2 and minimum 97.29% with all features, and LSTM has maximum 99.31% with subset-2 and minimum 97.44% with subset-5. Based on the above results, we can conclude that all the classifiers produce good results using subset3 features. The



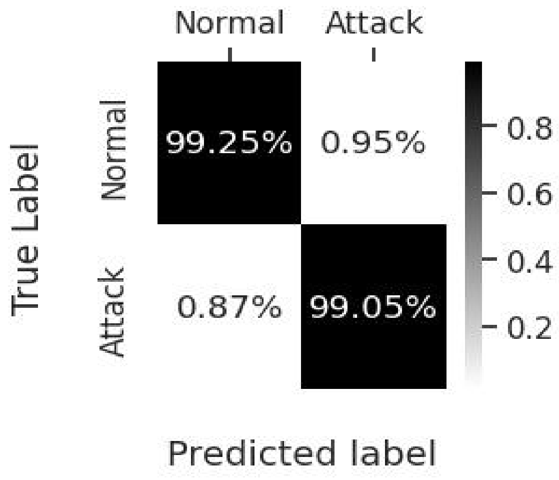
(a) RNN



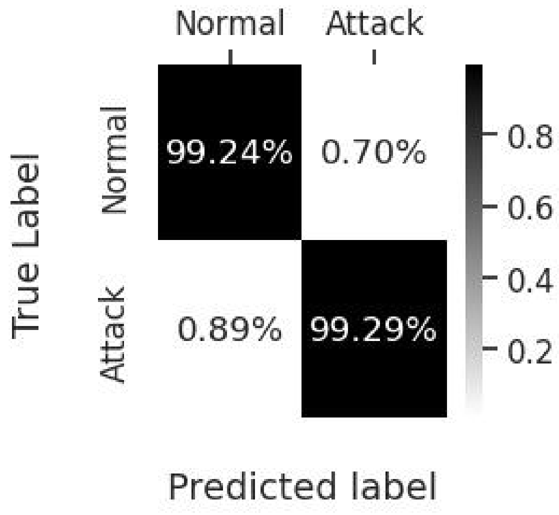
(b) CNN



(c) MLP



(d) LSTM



(e) DNN

FIGURE 7. Comparison of confusion matrix between RNN, CNN, MLP, LSTM, and DNN for subset-3.

TABLE 5. Comparison of performance results with all five sets of features.

Feature Sets	DL Methods	Computational Time (Seconds)	Maximum Accuracy (%)	Detection Rate (%)
	RNN	264.43	98.68	97.86
	CNN	202.39	98.85	97.87
All feature Set-1	MLP	142.94	98.01	89.99
	DNN	202.39	99.00	97.29
	LSTM	323.46	98.94	98.32
	RNN	256.78	99.33	99.20
	CNN	185.92	99.43	99.51
Subset-2	MLP	117.76	99.24	99.45
	DNN	185.92	99.53	99.49
	LSTM	223.06	99.14	99.31
	RNN	263.17	99.25	99.21
	CNN	181.88	99.37	99.60
Subset-3	MLP	142.37	99.33	99.41
	DNN	154.13	99.30	99.11
	LSTM	211.73	99.16	99.13
	RNN	278.09	99.22	99.06
	CNN	185.86	99.29	99.35
Subset-4	MLP	119.37	99.31	99.21
	DNN	155.66	99.25	99.06
	LSTM	231.18	99.06	99.02
	RNN	259.45	99.15	98.90
	CNN	202.41	99.26	99.07
Subset-5	MLP	117.40	98.91	95.24
	DNN	202.42	99.32	99.31
	LSTM	203.40	97.60	97.44

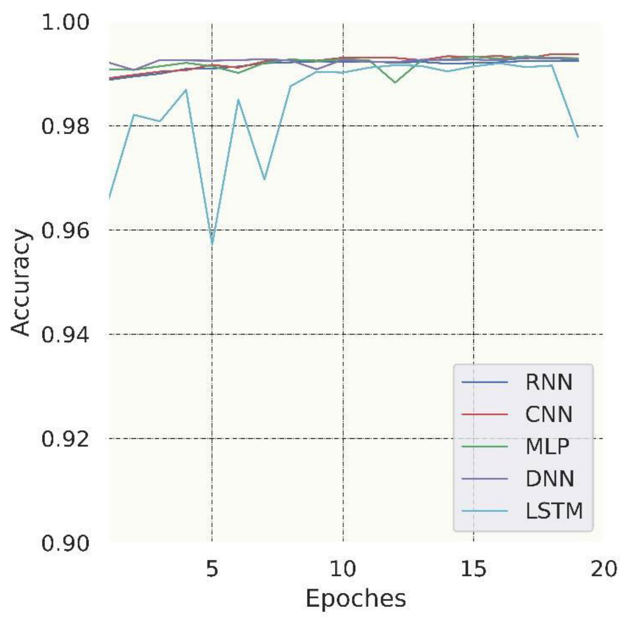


FIGURE 8. Accuracy curve between all algorithms with subset-3.

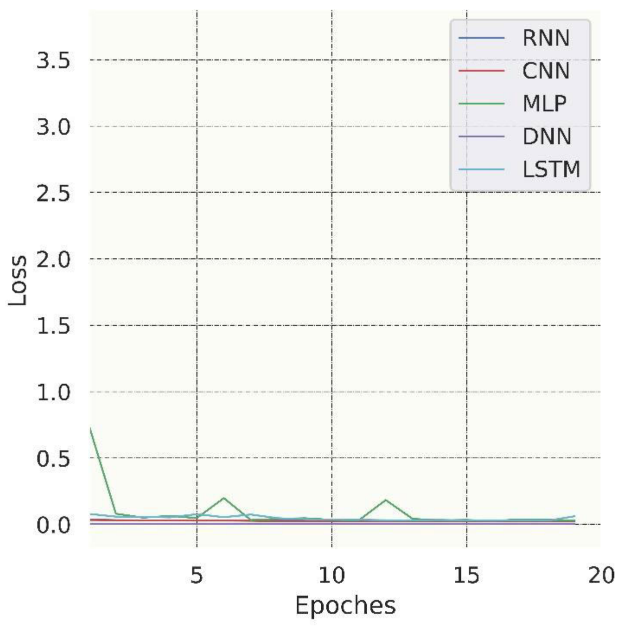
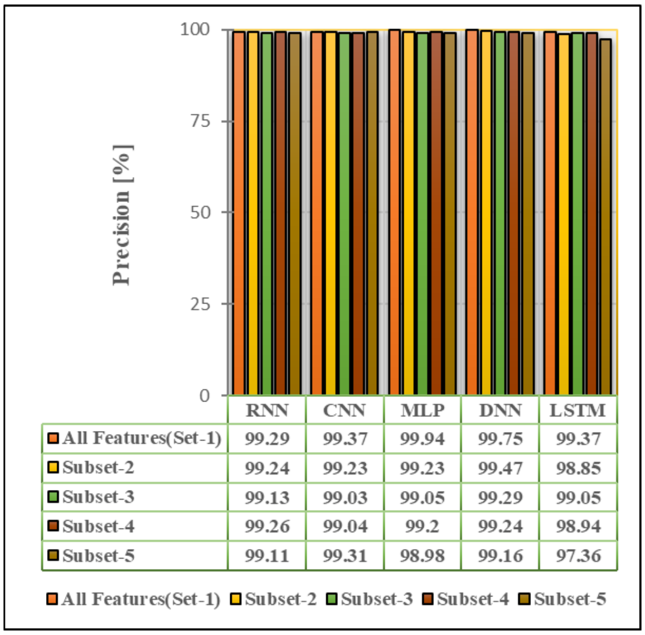


FIGURE 9. Loss curve between all algorithms with subset-3.

CNN method is a top performer with more stable and effective results than other classifiers in a specific scenario adopted in this paper. Thus, there are some advantages of this outcome: (i) the training sets are collected easily without knowing the details of traffic flows; (ii) with optimal features, the training phase becomes simple; (iii) the resource consumption and complexity of the methods is reduced due to training set with optimal features.

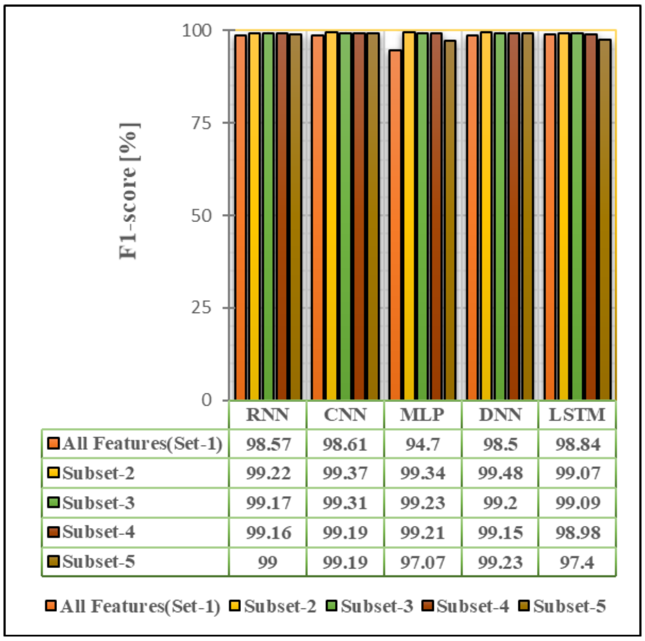
G. IMPLEMENTATION AND EVALUATION IN REAL TESTBED

To evaluate and verify the performance of DL methods on the real testbed, we select the methods that have been trained and validated with subset-3 features. The same network topology shown in Fig. 2. is used for the real testbed. The same way used to generate the training data (e.g., explained in section IV-A) has been followed to create/collect the flow



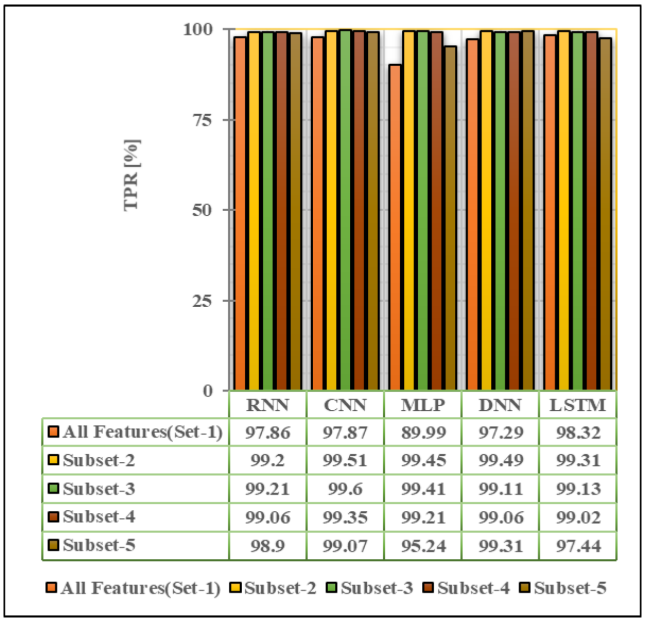
	RNN	CNN	MLP	DNN	LSTM
🟧All Features(Set-1)	99.29	99.37	99.94	99.75	99.37
🟨Subset-2	99.24	99.23	99.23	99.47	98.85
🟩Subset-3	99.13	99.03	99.05	99.29	99.05
🟫Subset-4	99.26	99.04	99.2	99.24	98.94
🟫Subset-5	99.11	99.31	98.98	99.16	97.36

(a) Precision



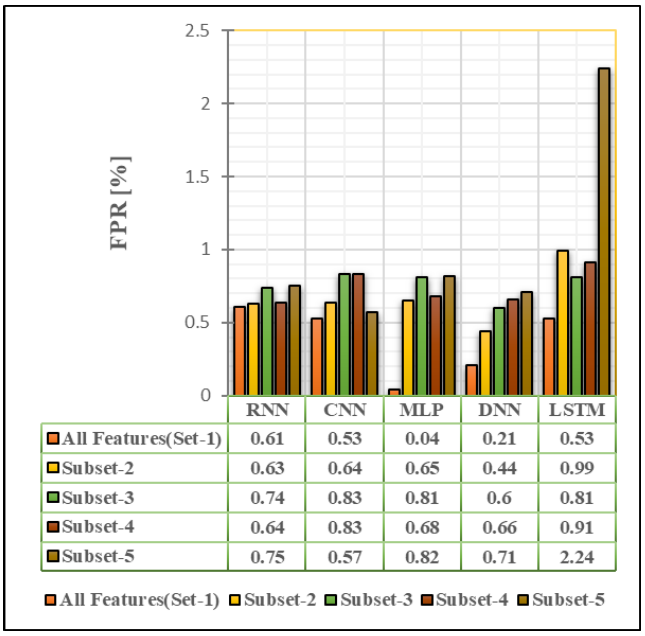
	RNN	CNN	MLP	DNN	LSTM
🟧All Features(Set-1)	98.57	98.61	94.7	98.5	98.84
🟨Subset-2	99.22	99.37	99.34	99.48	99.07
🟩Subset-3	99.17	99.31	99.23	99.2	99.09
🟫Subset-4	99.16	99.19	99.21	99.15	98.98
🟫Subset-5	99	99.19	97.07	99.23	97.4

(b) F1-Score



	RNN	CNN	MLP	DNN	LSTM
🟧All Features(Set-1)	97.86	97.87	89.99	97.29	98.32
🟨Subset-2	99.2	99.51	99.45	99.49	99.31
🟩Subset-3	99.21	99.6	99.41	99.11	99.13
🟫Subset-4	99.06	99.35	99.21	99.06	99.02
🟫Subset-5	98.9	99.07	95.24	99.31	97.44

(c) True positive rate



	RNN	CNN	MLP	DNN	LSTM
🟧All Features(Set-1)	0.61	0.53	0.04	0.21	0.53
🟨Subset-2	0.63	0.64	0.65	0.44	0.99
🟩Subset-3	0.74	0.83	0.81	0.6	0.81
🟫Subset-4	0.64	0.83	0.68	0.66	0.91
🟫Subset-5	0.75	0.57	0.82	0.71	2.24

(d) False positive rate

FIGURE 10. Comparison of performance results between all algorithms in terms of precision, f1-score, true positive rate, and false positive rate.

statistics for predicting normal or attack traffic flows in a real testbed environment. Each trained DL method is individually deployed in the controller. Then the method classifies the incoming flow with the number "0" or"1" (e.g., all methods have only two options in our test, so each method defines the normal flow as "0" and attack flow as "1"). To verify the overall performance during real-time traffic, we used 50 consecutive decisions, which every method individually makes under two network states (normal or attack). The correct detection rate of each method during real-time traffic is shown in Fig. 11. We observed that the output predictions of all the methods for normal flows are better than the attack flows. All methods achieved more than a 90% detection rate, especially with CNN, which reaches 99% in predicting normal flows. Similarly, the detection rate of methods reaches 87%, 97%, 85%, 93%, and 85% for attack flows, respectively.

Comparing the training time (e.g., in seconds) of DL methods for subset-3 features, RNN took more time to get trained than other methods. CNN took a reasonable time to get trained. The detection time (in microseconds (μs)) per flow for CNN is also slightly low than in other methods. In contrast, the detection time of LSTM is significantly higher than other methods, which indicates that the LSTM for attack detection can handle a few flows per second. Taking the detection rate, training, and detection times as evaluation parameters, we can conclude that CNN is the best method for detecting botnet-based DDoS attacks in an SDN environment. A visual representation of training and detection times is shown in Fig. 12 and Fig. 13, respectively.

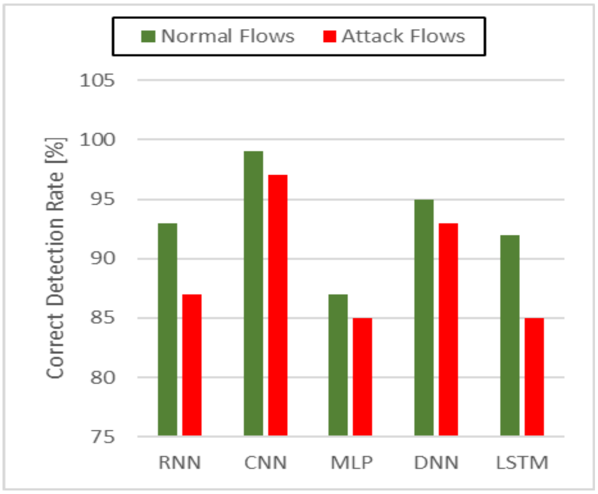


FIGURE 11. The correct detection rate of each algorithm during real-time traffic.

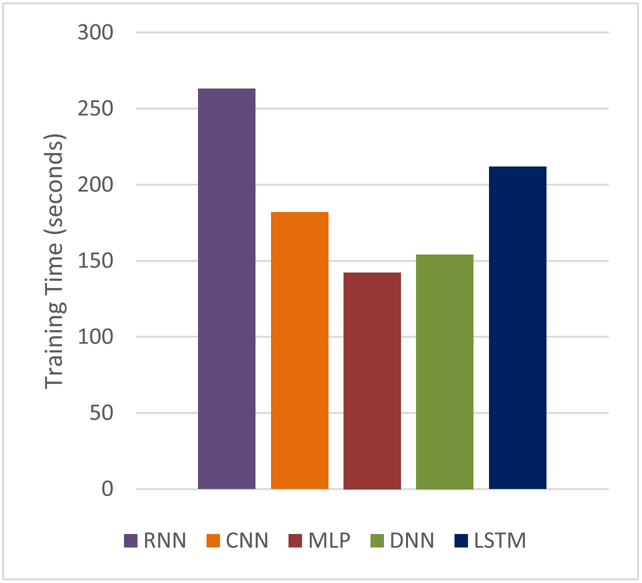


FIGURE 12. Comparison of training time with subset-3.

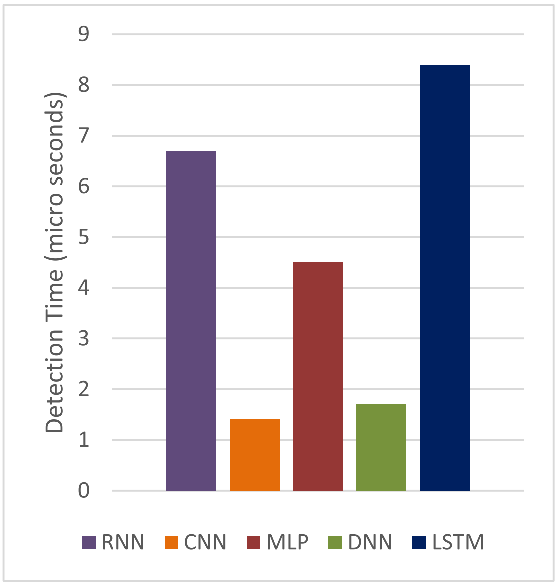


FIGURE 13. Comparison of detection time during real traffic.

VI. DISCUSSION AND FUTURE WORK

In this paper, we study and employ DL methods to contribute to the detection of botnet-based DDoS attacks in an SDN-supported environment. We explore the efficiency of the DL methods with baseline hyper-parameters and optimal features to detect the attack. We evaluate the performance of the DL methods using different evaluation metrics (e.g., accuracy, detection rate, training, detection time, etc.). A core contributing factor of this study is the generation and collection of the training dataset, which is purely developed in an SDN environment instead of relying on old or traditional datasets. In most studies [17], [20], the researchers used traditional datasets (e.g., Na-BaIoT, NSL-KDD, CIC-DDoS 2019, DARPA, etc.), which are not suitable for SDN due its flow based nature, and also these datasets suffer imbalanced problems. In addition, we do not rely on a dataset with many features but divide the whole dataset into subsets (e.g., optimal features) based on the feature's importance; then, these subsets have been tested to observe the impact of optimal features on the method performance. Simulation results show a variation in the performance of each DL method on the same subset of features, so we can conclude that the optimal features could improve the detection rate. Based on the experimental results and discussion above, we conclude that CNN is the best method for the proposed study and the adopted scene. Its accuracy reaches 99.37% with subset-3 features using generated dataset. During real testbed traffic, the detection rate of CNN for normal flows is 99% and 97% for attack flows. We also considered timing metrics (e.g., training and detection times) and observed that CNN took reasonable time during training and detection. Consequently, CNN shows an acceptable detection rate or accuracy during real-time botnet-based DDoS attack detection in an SDN environment. A final key advantage of this study is that the defense method protects the SDN from botnet-based DDoS attacks.

The limitation of this study is that it is only valid for botnet-based flooding DDoS attacks in SDN environments. It cannot accurately detect non-volumetric attacks, such as low-rate DDoS or other malicious attacks. Additionally, we focused on a single SDN controller-based environment.

We plan to extend this study to investigate the low-rate, spoofed DDoS and other malicious attacks with real SDN traffic using hybrid DL techniques. Training the real-time methods to keep the IDS systems updated is also good.

VII. CONCLUSION

SDN redefines the network's management and communications and leads to reforms. Although SDN has vast capabilities, it also introduced new emerging security challenges, which need the great attention of both the network and research community. The botnet attacks target the devices in the data plane or the controller in the control plane. Detecting the botnet and DDoS attacks in SDN becomes more challenging than in traditional networks due to the sophistication of traffic flow features. The SDN's decoupled architecture would help to build and deploy a method that flexibly detects the botnet-based DDoS attack. Nevertheless, the new network environments shall be adopted to provide a reliable definition and behavior of botnet-based DDoS attacks in SDN. Nowadays, deep learning-based network applications are trending and could be utilized to secure the SDN. The DL methods use the historical training sets in predicting real-time network state and inform the controller. In our study, first, we produced a dataset in a pure SDN-supported environment and then used DL techniques to predict the botnet-based DDoS attack to resolve these issues. The DL procedures, from dataset generation to attack detection, are done over a single controller and gratitude to the SDN's centralized control. In real-time, the controller sends a packet-In request to all the OpenFlow switches to collect the flow statistics, and then these flow statistics are used by the DL module inside the controller to detect the attacks. The real-time detection rate of CNN with 30 features reaches 99% for normal flows and 97% for attack flows, a remarkable contribution of this study to secure the SDN from botnet-based DDoS attacks. We named the CNN-based method "DepBot," simplifying the data preparation in the DL training phase. Although the DL methods achieved reasonable detection rates and accuracies with 30 features in the adopted network scenario, it is suggested to use advanced feature selection methods to better the prediction accuracy further.

DATA AVAILABILITY

The source code to generate the botnet attack and the normal and attack traffic dataset utilized to conduct this study is available at: https://github.com/Waqas-Nadeem/Botnet-based-DDoS-attack-in-an-SDN-Environment.

REFERENCES

[1] L. Tan, Y. Pan, J. Wu, J. Zhou, H. Jiang, and Y. Deng, "A new framework for DDoS attack detection and defense in SDN environment," *IEEE Access*, vol. 8, pp. 161908–161919, 2020.

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

[3] Y. Cui, Q. Qian, C. Guo, G. Shen, Y. Tian, H. Xing, and L. Yan, "Towards DDoS detection mechanisms in software-defined networking," *J. Netw. Comput. Appl.*, vol. 190, Sep. 2021, Art. no. 103156.

[4] 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, pp. 1–8, Apr. 2018.

[5] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for Internet of Things," *Future Gener. Comput. Syst.*, vol. 82, pp. 761–768, May 2018.

[6] J. A. Pérez-Díaz, I. A. Valdovinos, 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.

[7] R. K. Chouhan, M. Atulkar, and N. K. Nagwani, "A framework to detect DDoS attack in Ryu controller based software defined networks using feature extraction and classification," *Appl. Intell.*, pp. 1–21, 2022.

[8] Y. Liu, T. Zhi, M. Shen, L. Wang, Y. Li, and M. Wan, "Software-defined DDoS detection with information entropy analysis and optimized deep learning," *Future Gener. Comput. Syst.*, vol. 129, pp. 99–114, Apr. 2022.

[9] O. Habibi, M. Chemmakha, and M. Lazaar, "Imbalanced tabular data modelization using CTGAN and machine learning to improve IoT botnet attacks detection," *Eng. Appl. Artif. Intell.*, vol. 118, Feb. 2023, Art. no. 105669.

[10] H. S. Ilango, M. Ma, and R. Su, "A FeedForward–Convolutional neural network to detect low-rate DoS in IoT," *Eng. Appl. Artif. Intell.*, vol. 114, Sep. 2022, Art. no. 105059.

[11] M. W. Nadeem, H. G. Goh, V. Ponnusamy, and Y. Aun, "DDoS detection in SDN using machine learning techniques," *Comput., Mater. Continua*, vol. 71, no. 1, pp. 771–789, 2022, doi: 10.32604/cmc.2022.021669.

[12] K. N. Rao, K. V. Rao, and P. V. G. D. P. Reddy, "A hybrid intrusion detection system based on sparse autoencoder and deep neural network," *Comput. Commun.*, vol. 180, pp. 77–88, Dec. 2021.

[13] M. W. Nadeem, H. G. Goh, Y. Aun, and V. Ponnusamy, "A recurrent neural network based method for low-rate DDoS attack detection in SDN," in *Proc. 3rd Int. Conf. Artif. Intell. Data Sci. (AiDAS)*, Sep. 2022, pp. 13–18.

[14] P. L. S. Jayalaxmi, G. Kumar, R. Saha, M. Conti, T.-H. Kim, and R. Thomas, "DeBot: A deep learning-based model for bot detection in industrial Internet-of-Things," *Comput. Electr. Eng.*, vol. 102, Sep. 2022, Art. no. 108214.

[15] H.-T. Nguyen, Q.-D. Ngo, D.-H. Nguyen, and V.-H. Le, "PSI-rooted subgraph: A novel feature for IoT botnet detection using classifier algorithms," *ICT Exp.*, vol. 6, no. 2, pp. 128–138, Jun. 2020.

[16] A. Al Shorman, H. Faris, and I. Aljarah, "Unsupervised intelligent system based on one class support vector machine and grey wolf optimization for IoT botnet detection," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 7, pp. 2809–2825, Jul. 2020.

[17] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders," *IEEE Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, Jul. 2018.

[18] M. Asadi, M. A. Jabraeil Jamali, S. Parsa, and V. Majidnezhad, "Detecting botnet by using particle swarm optimization algorithm based on voting system," *Future Gener. Comput. Syst.*, vol. 107, pp. 95–111, Jun. 2020.

[19] I. Idrissi, M. Boukabous, M. Azizi, O. Moussaoui, and H. El Fadili, "Toward a deep learning-based intrusion detection system for IoT against botnet attacks," *IAES Int. J. Artif. Intell. (IJ-AI)*, vol. 10, no. 1, p. 110, Mar. 2021.

[20] T. Saba, A. Rehman, T. Sadad, H. Kolivand, and S. A. Bahaj, "Anomaly-based intrusion detection system for IoT networks through deep learning model," *Comput. Electr. Eng.*, vol. 99, Apr. 2022, Art. no. 107810.

[21] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, Oct. 2016, pp. 258–263.

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

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

[24] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del-Rincón, and D. Siracusa, "Lucid: A practical, lightweight deep learning solution for DDoS attack detection," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 2, pp. 876–889, Jun. 2020.

[25] K. S. Sahoo, B. K. Tripathy, K. Naik, S. Ramasubbareddy, B. Balusamy, M. Khari, and D. Burgos, "An evolutionary SVM model for DDOS attack detection in software defined networks," *IEEE Access*, vol. 8, pp. 132502–132513, 2020.

[26] Z. Liu, Y. He, W. Wang, and B. Zhang, "DDoS attack detection scheme based on entropy and PSO-BP neural network in SDN," *China Commun.*, vol. 16, no. 7, pp. 144–155, Jul. 2019.

[27] Y. Cui, L. Yan, S. Li, H. Xing, W. Pan, J. Zhu, and X. Zheng, "SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks," *J. Netw. Comput. Appl.*, vol. 68, pp. 65–79, Jun. 2016.

[28] A. A. Ahmed, W. A. Jabbar, A. S. Sadiq, and H. Patel, "Deep learning-based classification method for botnet attack detection," *J. Ambient Intell. Humaniz. Comput.*, vol. 143, pp. 1–10, Mar. 2020.

[29] W.-C. Shi and H.-M. Sun, "DeepBot: A time-based botnet detection with deep learning," *Soft Comput.*, vol. 24, no. 21, pp. 16605–16616, Nov. 2020.

[30] H. Alkahtani and T. H. H. Aldhyani, "Botnet attack detection by using CNN-LSTM model for Internet of Things applications," *Secur. Commun. Netw.*, vol. 2021, pp. 1–23, Sep. 2021.

[31] G. De La Torre Parra, P. Rad, K.-K.-R. Choo, and N. Beebe, "Detecting Internet of Things attacks using distributed deep learning," *J. Netw. Comput. Appl.*, vol. 163, Aug. 2020, Art. no. 102662.

[32] S. Velliangiri and H. M. Pandey, "Fuzzy-Taylor-elephant herd optimization inspired deep belief network for DDoS attack detection and comparison with state-of-the-arts algorithms," *Future Gener. Comput. Syst.*, vol. 110, pp. 80–90, Sep. 2020.

[33] J. Cui, M. Wang, Y. Luo, and H. Zhong, "DDoS detection and defense mechanism based on cognitive-inspired computing in SDN," *Future Gener. Comput. Syst.*, vol. 97, pp. 275–283, Aug. 2019.

[34] A. A. E.-B. Donkol, A. G. Hafez, A. I. Hussein, and M. M. Mabrook, "Optimization of intrusion detection using likely point PSO and enhanced LSTM-RNN hybrid technique in communication networks," *IEEE Access*, vol. 11, pp. 9469–9482, 2023.

[35] A. Aldweesh, A. Derhab, and A. Z. Emam, "Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues," *Knowl.-Based Syst.*, vol. 189, Feb. 2020, Art. no. 105124.

[36] T. Su, H. Sun, J. Zhu, S. Wang, and Y. Li, "BAT: Deep learning methods on network intrusion detection using NSL-KDD dataset," *IEEE Access*, vol. 8, pp. 29575–29585, 2020.

[37] E. Mushtaq, A. Zameer, and A. Khan, "A two-stage stacked ensemble intrusion detection system using five base classifiers and MLP with optimal feature selection," *Microprocessors Microsyst.*, vol. 94, Oct. 2022, Art. no. 104660.

[38] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *J. Netw. Comput. Appl.*, vol. 169, Nov. 2020, Art. no. 102767.

[39] T. H. H. Aldhyani and H. Alkahtani, "Cyber security for detecting distributed denial of service attacks in agriculture 4.0: Deep learning model," *Mathematics*, vol. 11, no. 1, p. 233, Jan. 2023.

[40] S. A. Wagan, J. Koo, I. F. Siddiqui, N. M. F. Qureshi, M. Attique, and D. R. Shin, "A fuzzy-based duo-secure multi-modal framework for IoMT anomaly detection," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 35, no. 1, pp. 131–144, Jan. 2023.

[41] K. Sridevi and M. A. Saifulla, "LBABC: Distributed controller load balancing using artificial bee colony optimization in an SDN," *Peer Peer Netw. Appl.*, vol. 16, pp. 1–11, Feb. 2023.

[42] P. V. Shalini, V. Radha, and S. G. Sanjeevi, "Early detection and mitigation of TCP SYN flood attacks in SDN using chi-square test," *J. Supercomput.*, vol. 79, no. 9, pp. 1–33, 2023.

[43] K. Subratie, S. Aditya, and R. J. Figueiredo, "EdgeVPN: Self-organizing layer-2 virtual edge networks," *Future Gener. Comput. Syst.*, vol. 140, pp. 104–116, Mar. 2023.

[44] W. Zhijun, X. Qing, W. Jingjie, Y. Meng, and L. Liang, "Low-rate DDoS attack detection based on factorization machine in software defined network," *IEEE Access*, vol. 8, pp. 17404–17418, 2020.

[45] Y. Cao, H. Jiang, Y. Deng, J. Wu, P. Zhou, and W. Luo, "Detecting and mitigating DDoS attacks in SDN using spatial–temporal graph convolutional network," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 6, pp. 3855–3872, Nov. 2022.

MUHAMMAD WAQAS NADEEM received the B.S. degree from Lahore Garrison University and the M.Phil. degree in computer science from the University of Management and Technology, Lahore. He is currently pursuing the Ph.D. degree in computer science with Universiti Tunku Abdul Rahman (UTAR), Malaysia. He has published numerous research articles in various international journals and conferences. His potential research interests include network security, FinTech security, SDN security, the IoT security, artificial intelligence, machine learning, and deep learning.

HOCK GUAN GOH received the Bachelor of Information Technology degree in multimedia from the University of Malaya, in 2003, the Master of Science degree in information technology from Multimedia University, in 2006, and the Doctor of Philosophy degree in electronic and electrical engineering from the University of Strathclyde, in 2014. He is currently an Associate Professor with the Faculty of Information and Communication Technology (FICT), Kampar Campus, Universiti Tunku Abdul Rahman (UTAR). His research interests include cognitive wireless sensor-actor networks, agriculture/environmental monitoring systems, the Internet of Things, and data analysis/processing.

YICHIET AUN received the Ph.D. degree in computer science from the Science University of Malaysia. He is currently an Assistant Professor with Universiti Tunku Abdul Rahman, Malaysia. His research interests include computer networking, data analytics, cybersecurity, and sentic computing.

VASAKI PONNUSAMY received the Bachelor of Computer Science degree and the M.Sc. degree in computer science from the Science University of Malaysia and the Ph.D. degree in IT from Universiti Teknologi Petronas (UTP), Malaysia, in 2013. She is currently an Assistant Professor with the Higher Colleges of Technology, United Arab Emirates, and before that, she was an Assistant Professor with Universiti Tunku Abdul Rahman, Malaysia. She is also working on cybersecurity, IT governance, biologically-inspired computing, wireless sensor networks, and energy harvesting.