IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 20 May 2022, accepted 4 June 2022, date of publication 13 June 2022, date of current version 21 June 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3182818

Application Layer DDoS Attack Detection Using Cuckoo Search Algorithm-Trained Radial Basis Function

HAKEM BEITOLLAHI[1], DYARI MOHAMMED SHARIF[2], AND MAHDI FAZELI[3]
[1]School of Computer Engineering, Iran University of Science and Technology, Tehran 1684613114, Iran
[2]Computer Science Department, Soran University, Kurdistan Region, Soran 44008, Iraq
[3]School of Information Technology, Halmstad University, 301 18 Halmstad, Sweden

Corresponding author: Hakem Beitollahi (beitollahi@iust.ac.ir)

ABSTRACT In an application-layer distributed denial of service (App-DDoS) attack, zombie computers bring down the victim server with valid requests. Intrusion detection systems (IDS) cannot identify these requests since they have legal forms of standard TCP connections. Researchers have suggested several techniques for detecting App-DDoS traffic. There is, however, no clear distinction between legitimate and attack traffic. In this paper, we go a step further and propose a Machine Learning (ML) solution by combining the Radial Basis Function (RBF) neural network with the cuckoo search algorithm to detect App-DDoS traffic. We begin by collecting training data and cleaning them, then applying data normalizing and finding an optimal subset of features using the Genetic Algorithm (GA). Next, an RBF neural network is trained by the optimal subset of features and the optimizer algorithm of cuckoo search. Finally, we compare our proposed technique to the well-known k-nearest neighbor (k-NN), Bootstrap Aggregation (Bagging), Support Vector Machine (SVM), Multi-layer Perceptron) MLP, and (Recurrent Neural Network) RNN methods. Our technique outperforms previous standard and well-known ML techniques as it has the lowest error rate according to error metrics. Moreover, according to standard performance metrics, the results of the experiments demonstrate that our proposed technique detects App-DDoS traffic more accurately than previous techniques.

INDEX TERMS Application layer DDoS, machine learning, radial basis function, cuckoo search algorithm, genetic algorithm.

I. INTRODUCTION

In application-layer DDoS attacks (App-DDoS), the attackers send legitimate packets toward the victim server to bring down the server [5], [11]. As the malicious packets mimic the behavior of legitimate users and contain the genuine source IP addresses, neither the victim server nor IDS (Intrusion Detection System) can distinguish the packet of attackers from legitimate users [5], [11]. The main goal of DDoS attackers regardless of their types is to force victim servers to respond so slowly as to be unusable or shut down completely. To achieve this goal, they overwhelm the bottleneck resources of the victim server. The desired bottleneck resources for application-layer DDoS attacks are TCP/IP stacks, CPU cycles, memory, I/O bandwidth, and disk/database bandwidth. In App-DDoS attacks, the adversary overwhelms the bottleneck resources via legitimate requests. To launch such an attack, every bot machine that wants to participate in the attack establishes a TCP connection with the victim server, which requires a genuine IP address.

The main aim of a defense technique against App-DDoS attacks is to detect and distinguish attack traffic from legitimate ones. This issue is difficult as attackers purposely fabricate App-DDoS traffic to look like legitimate traffic. On the other hand, professional attackers continuously change their toolkits and develop more sophisticated App-DDoS traffic; hence detecting attack traffic becomes more difficult. However, once the attack traffic is detected, the victim server can block any traffic coming from attack sources [5], [11].

Several classic, heuristic, and more recently machine-learning (ML) based techniques have been proposed to

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

detect and distinguish App-DDoS traffic from legitimate traffic during the last decade. Previous classic and heuristic techniques, which we call traditional techniques, detect App-DDoS traffic by rules (e.g., statistical, challenge/response, and time series) programmed by traffic engineers. Traditional approaches have not achieved much to catch App-DDoS traffic, as they suffer from low accuracy due to the dynamic and evolving natures of App-DDoS attacks. Some well-known traditional techniques are reviewed in Section II.

Recently, academics and industries are exploring artificial intelligence techniques, especially ML techniques to detect App-DDoS traffic. ML techniques such as KNN, SVM, random forest, logistic regression, and Naïve Bayesian can accurately classify binary data (the data belonging to class YES or class NO) if we have a large volume of labeled samples (the samples with labels YES or NO). The mechanism is that human experts select the classification features. In deep-learning (DL) approaches such as CNN and RNN, even feature selection can be made by machine without human intervention through a series of nonlinear processing layers. Next, an ML model is customized and trained with the labeled samples via the selected features. The trained ML model is used to predict the class of upcoming data. Therefore, ML techniques can be a powerful tool for detecting App-DDoS traffic. Section II discusses some recent ML techniques that detect App-DDoS traffic. Section II discusses some recent ML techniques that detect App-DDoS traffic.

This paper proposes a novel technique based on machine learning to detect the traffic of App-DDoS attacks. A Radial Basis Function (RBF) neural network is used at the heart of the technique. The Cuckoo Search optimizer Algorithm (CSA) is applied to the RBF network to enhance the network power detection. Through the Genetic Algorithm (GA), the most valuable features of network traffic that have the main role in detecting App-DDoS traffic are determined and applied to the RBF network to train the network. The trained RBF significantly detects and distinguishes the attack traffic from legitimate ones. Experimental results show that our proposed technique improves accuracy in detection on average by 3% and 6% compared to Bootstrap Aggregation (Bagging) technique and k-nearest neighbor (KNN), respectively. The proposed technique also performs better than well-known machine learning techniques such as support vector machine (SVM), multi-layer perceptron (MLP), and recurrent neural network (RNN).

The main contributions of the proposed technique are as follows.

1) Utilizing Radial Basis Function (RBF) neural network to detect App-DDoS traffic from legitimate traffic.
2) Utilizing the Genetic Algorithm (GA) to distinguish the most valuable features of the dataset in order to increase the accuracy of attack detection. Utilizing Radial Basis Function (RBF) neural network to detect App-DDoS traffic from legitimate traffic.
3) Applying the Cuckoo search algorithm (CSA) to RBF neural network to better train the network.
4) Experimental results show that the combination of CSA + GA + RBF outperforms other well-known machine learning-based techniques such as KNN, Bagging, SVM, RNN, and MLP.

The rest of the paper is organized as follows: Sections II, III, IV and V present the related work, the proposed technique, experimental results and the conclusion, respectively.

II. RELATED WORK

Several App-DDoS traffic detection techniques have been proposed in the last decade. In this section, a few traditional techniques are discussed first, then the techniques based on machine learning and deep learning are discussed.

A. TRADITIONAL TECHNIQUES

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is an elegant mechanism to separate human-based users from automated software tools [9], [25]. CAPTCHA is the most trusted technique against App-DDoS attacks; however, it suffers various challenges such as annoying users, technical difficulties with some browsers, not being robust against image recognition techniques, and third-party users.

In Whitelisting Technique [19], a list of source IP addresses called very important IP addresses (VIPs) is prepared before the attack. When the victim server is under an App-DDoS attack, priority is given to traffic that belongs to the VIP list. The main challenges with this technique are (1) it is not user transparent, (2) it has the problem with mobile users (3) The zombie machines and royal clients that have resided behind a NAT (network address translator) have the same privilege to access the server.

Trust Management Helmet (TMH) [20] is a mechanism that distinguishes legitimate users from malicious users using trust. During an App-DDoS attack, the priority for establishing sessions with the victim server is given to users with higher trust values instead of identifying all bot machines. TMH depends on the browser's cookie; however, in many companies and institutes, the cookies are deleted due to privacy and security issues.

In the ConnectionScore technique [4], every session is scored based on its history and statistical analysis that has been done during normal connection. Those sessions that take lower scores are blocked, and bottleneck resources are retaken from them. However, the computational overhead imposed on the server can be large during the attack as it measures a proper score for every session.

Authors in [10] proposed a probability-based detection mechanism based on the central limit theorem and the windowing method. Through the probabilistic model, [10] trains an engine offline using distributions of six TCP flags of incoming SYN packets of the DARPA dataset. The trained engine detects malicious traffic, mimicking normal packets

with spoofed information in the online mode. [10] shows that their probabilistic technique is better than the entropy-based strategy in terms of false-negative rate (FNR) in various attack circumstances. The main problem with the technique is that the distribution of six TCP flags is not a robust criterion for App-DDoS traffic detection. Due to this limitation, the authors only considered the SYN flood attack in their experiments.

B. MACHINE-LEARNING BASED TECHNIQUES

Authors in [21] use machine learning and deep learning to identify transport and application layer DDoS attacks in software-defined networks (SDNs). [21] includes four modules: Flow collector, preprocessing, detection, and flow Manager. The flow collector module gathers the data packets from the network, creates the flows, and then forwards them to the preprocessing module. In the preprocessing, data is cleaned. The detection module utilizes a trained model based on various machine learning and deep learning techniques to classify the input as suspicious or benign. The model is trained based on KNN, SVM, RF (Random Forest), MLP, CNN (Convolutional Neural Network), GRU (Gated Recurrent Units), and LSTM (Long Short-Term Memory) classifiers. Two recent security datasets, CICDoS2017, and CICDoS2019 are used to train the model. Finally, the flow manager module informs the controller about the suspicious traffic.

Authors in [15] trained a model based on the decision tree (DT) approach for an imbalanced dataset in which the normal traffic dominates attack traffic. Most classification techniques are normally biased toward the majority class (i.e., normal traffic). [15] proposed an ensemble-based method by using K-Means, RUSBoost, and DT approaches to mitigate the class imbalance problem. The approach selected DT as the best classifier after optimizing the hyper-parameters in terms of accuracy, fast training time, and improved average prediction rate.

Feng *et al.* [8] propose a reinforcement-learning-based model to detect and mitigate App-DDoS attacks. The model is continuously trained with various metrics related to the server's load, clients' dynamic behaviors, and the victim's network load. The model utilizes the Markov decision process to construct the attack classification model. The reward function of the reinforcement-learning model is a multi-objective function to guide a reinforcement-learning agent to learn the most suitable action in mitigating App-DDoS attacks.

Banerjee *et al.* [3] propose two modules that work sequentially to detect and mitigate DDoS attacks. The first module, called the Signature IDS, uses machine learning algorithms like Naive Bayes, KNN, and K-Means to classify incoming packets as normal or anomalous. Hence, all the possible intruders in the incoming traffic are predicted beforehand. Among various ML techniques, the one giving the best possible outcome is implemented in the Signature IDS. The second module performs a three-way handshake to identify the exact host, which is an intruder.

Authors in [16] propose a DDoS attack detection architecture that integrates Bi-Directional Long Short-Term Memory (BI-LSTM), a Gaussian Mixture Model (GMM), and incremental learning. Unknown traffic is captured by the GMM unit and then is labeled by data engineers for discrimination. The labeled traffic is then fed back to the BI-LSTM and the GMM for incremental learning. The Bi-LSTM is used to discriminate malicious traffic from legitimate ones.

Alghazzawi *et al.* [2] investigate a hybrid model of CNN and Bi-LSTM for DDoS attacks classification. The chi-squared (x^2) is used to identify highly related features. Next, a CNN network is used to extract the high-rated features. These features are fed to the Bi-LSTM model to distinguish attack packets from normal packets.

Zeeshan *et al.* [23] propose a Protocol Based Deep Intrusion Detection (PB-DID) architecture in which a data-set of packets from IoT traffic is created based on flow and Transmission Control Protocol (TCP). Next, an LSTM-based unsupervised deep learning model is utilized to detect DDoS attack traffic. The model is trained on all the data available in two famous benchmarks, namely UNSW-NB15 and BotIoT data sets, to cover the maximum possible packet types. In this model, the number of features for training is reduced to almost half.

A hybrid model of random forest (RF) and XGBoost is presented in [12] to predict the DDoS attack traffic. The data of the benchmark is pre-processed to handle irrelevant data. Next, features are extracted from the data, and data is labeled. The hybrid model is trained with the extracted features and labeled data.

Table 1 summarize and compare previous machine-learning based techniques via various parameters. In the table, '+' operator is used when two (or more) techniques are used in parallel, and '→' is used when two (or more) techniques are used in series (output of a technique becomes the input of the next one).

III. THE PROPOSED TECHNIQUE

The proposed algorithm includes data gathering, data cleaning, data normalization, feature selection based on genetic algorithm (GA), and training of the Radial Basis Function (RBF) neural network based on the optimizer algorithm of cuckoo search. Fig. 1 shows the general flowchart of the proposed technique. The details of each step are explained as follows.

A. DATA GATHERING

The data is gathered via NSL-KDD dataset [17]. The NSL-KDD dataset is derived from the KDD Cup 99 dataset, one of the most available datasets for the research community. The NSL-KDD dataset includes legitimate traffic (48%), DDoS traffic (35%), Probe attack (10.55%), U2R attack (0.32%), and R2L (6.13%). Each dataset record is made up of 41 qualitative and quantitative features.

TABLE 1. Summarizing and comparing previous machine-learning based techniques.

Ref	Dataset	Technique(s) for Feature section	Number of features for training the model	Technique(s) for classification	Technique for model optimizing	Standard performance metrics	Contributions	Drawbacks
[15]	KDD NSL-KDD TUIDS Kyoto CAIDA NITRIDS	No	Not reported	Decision Tree(DT)	No	Not reported	Handling the Imbalanced dataset	No mechanism to find the best feature for the root of DT
[8]	Simulated traffic	No	8 out of 8	Deep Reinforcement learning	No	Accuracy = 93% Recall= 98% Precision = 95%	Reinforcement learning based on Markov decision process within multi-objective reward function	Some features are dependent
[3]	Kaggle	No	Not reported	(KNN + Naive-Bayes) → K-means	No	Accuracy = 94%	Identifing intruders	Utilizing dependent features/sensitive to irrelevant and noisy data, time consuming with big data set
[16]	CIC-IDS2017 CIC-IDS2019	No	80 out of 80	Bi-LSTM → GMM	No	Recall = 94% Precision =94% F1-score = 94%	High accuracy + handling unknown traffic + utilizing time domain correlation of data in learning	Utilizing dependent features /very slow tech.
[2]	CIC-IDS2019	Chi-squared (x^2)	10 out of 86	CNN →Bi-LSTM	No	Accuracy = 94% Recall= 92% Precision = 94%	Utilize a hybrid model of CNN+BiLSTM enhanced with feature selection using chi-squared	Very slow tech./feature selection is not efficient with chi-squared
[23]	UNSWNB15 Bot-IoT	Select common features between two dataset by hand	26 out of (46,43)	LSTM	No	Accuracy=96%	Use common features of two different datasets for training the model + handling imbalanced dataset	Retain only previous contextual knowledge and discard upcoming contextual information which is inaccurate technique/slow tech.
[12]	UNWS-np-15	No	45 out of 45	RF (Random Forest) + XGBoost	No	Recall = 89% Precision =89% F1-score = 89%	Utilizing different machine learning classifiers via latest dataset	Utilizing dependent features/low accuracy/ susceptible to overfitting in the presence of noisy data/unreliable for categorical data
[21]	CIC-IDS2017 CIC-IDS2019	Principal Component Analysis (PCA)	15 out of (49 +50)	RF + SVM +KNN + MLP + CNN + GRU + LSTM	No	Accuracy = 99% Recall= 99% Precision = 9%	Exploring diverse ML/DL techniques in terms of accuracy under different attack types and compare time and space complexity	No new technique

In our study, we excluded all categories except DoS and legitimate. After excluding all other types of attacks, our dataset has 57% normal traffic and 43% DDoS traffic. The dataset was then reduced to a small number of 3495 samples which keeps the normal traffic rates and Dos traffic unchanged (i.e., 1992 samples are normal and 1503 samples are DoS traffic). We utilized 2796 samples for the training set and 699 samples for the testing set, which constitute 80% and 20%, respectively. Finally, 233 extra samples are randomly selected in the dataset for the holdout dataset while keeping the rate of normal data and attack data unchanged. The holdout dataset is used to check the overfitting problem of the model.

B. DATA CLEANING

In many real-world data mining applications, even with large amounts of data and adequate storage space, some data may be lost in existing samples. However, the issue arises when large data sets cannot ignore individual values due to their efficacy. Organizations have access to vast volumes of data that affect their business decisions. The data gathered from multiple sources is dirty, which will impact the accuracy of

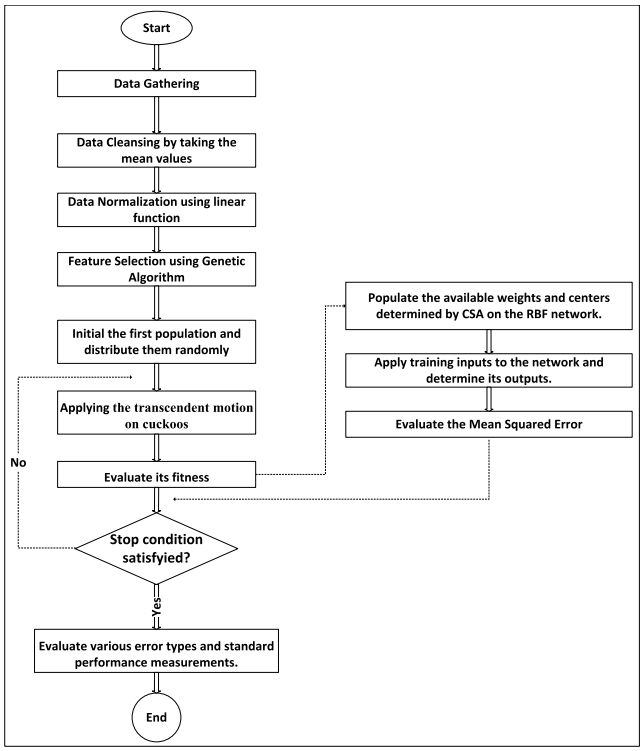

FIGURE 1. The flowchart of the proposed technique.

the prediction result. Data cleaning improves data quality, allowing organizations to ensure that their data is suitable for analysis. One solution is to use fixed values to replace and clear the lost data [13]. In this work, the average value of each feature's lost values is utilized and substituted in missing samples.

C. DATA NORMALIZATION

Larger values have a higher influence on the model due to the non-uniform range of features and their various units, but this does not always imply that they are more significant. Feature normalization is a technique for keeping all values within predefined ranges. However, choosing the right normalizing technique is crucial because applying normalization to the input might affect the data's structure. The goal of the feature normalization technique is to compensate for the impacts of mismatch in the environment [1]. In this work, data normalization is done in order to overcome this problem. The data of every feature are adjusted to a range of [−1,1] using linear normalization [22].

$$X = 2 \times (x - min(x))/(max(x) - min(x)) - 1 \qquad (1)$$

where $min(x)$ is the minimum of the input x, $max(x)$ is the maximum of the input x and X is the normalized x.

D. FEATURE SELECTION

Feature selection is a fundamental topic in ML that significantly influences the model's performance. The data features used to train the ML models significantly impact the results that may be gotten. Model performance can be harmed by features that are irrelevant or just partially relevant [7]. Since CSA is a nature-inspired algorithm, we tend to use also a nature-inspired algorithm like Genetic Algorithms (GAs) for feature selection. GA is a stochastic function in natural genetics and evolution, a heuristic optimization method based on natural evolution's rules. GA finds the best answer after a series of repeated calculations based on Darwin's "Survival of the Fittest" theory. We use GA for feature selection for the following reasons: (1) GAs commonly outperform conventional feature selection algorithms [24]. (2) For the mortality prediction problem, the GA-selected feature subset for one classifier can be used for others while still obtaining high performance [24]. (3) GAs are capable of managing data sets with a variety of features [24]. (4) GA achieves comparable or even superior predictive outcomes with far fewer features, saving time and cost, making it more advantageous [18], [24]. (5) GAs do not require specific information about the problem under study [18].

Our procedure of feature selection based on GA is as follows. The GA technique creates an ideal binary vector, with each bit corresponding to a feature. If the i^{th} bit of this vector equals 1, the i^{th} feature is permitted to participate in classification; if the bit equals 0, the feature is not permitted to participate. The starting population is produced at random from the sample space of feature sets. A score is assigned to each member in the starting population. The performance of the provided estimator is measured in this manner. The estimator used in this study is the Mean Squared Error (MSE).

A tournament selection is conducted to decide which members will continue to the following generation. We set the tournament size at three, which determines the number of members who compete against one another based on a score criterion. The tournament winner is chosen as a parent for the following generation. After then, the child has both parents' genetic material. This attribute is called crossover, and we set it to be 0.5, representing the probability of crossing over from one generation to the next. Next, a random mutation is added to each generation in addition to the crossover. We set the probability parameter that a mutation will happen to 0.2. Finally, to set the population size to 20 and the maximum iteration to 100, we implemented the GA so that in case the population's best member has not improved over numerous generations, even before reaching the maximum iteration, the search has yielded an optimal result.

Using the above procedure, we select nine features out of 41 features. The features selected by GA are illustrated in table 2.

E. RADIAL BASIS FUNCTION NEURAL NETWORK ARCHITECTURE

Radial Basis Function (RBF) neural network is widely used for the nonparametric estimation of a multidimensional function through a set of limited features. As Fig. 2 shows, these networks include three layers: Input layer, hidden layer, and output layer. The duty of the input layer is to assign

TABLE 2. The selected features via GA and their descriptions.

Feature	Description	Type
protocol_type	Type of the protocol, e.g. tcp, udp, etc	Discrete
service	Network service on the destination, e.g., http, telnet, etc	Discrete
num_access_files	Number of operations on access control files	Continuous
serror_rate	The percentage of connections that have "SYN" errors	Continuous
rerror_rate	The percentage of connections that have "REJ" errors	Continuous
same_srv_rate	The percentage of connections to the same service	Continuous
diff_srv_rate	The percentage of connections to different services	Continuous
srv_serror_rate	The percentage of connections that have "SYN" errors	Continuous
srv_rerror_rate	The percentage of connections that have "REJ" errors	Continuous

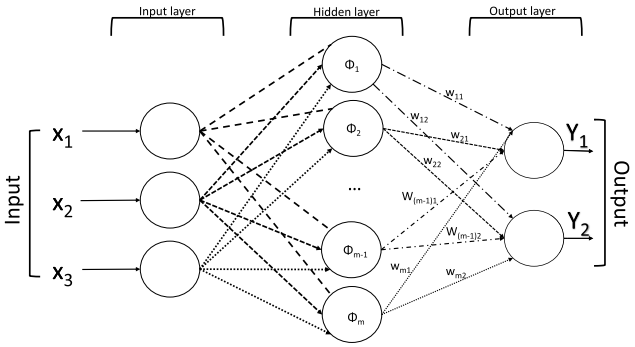

FIGURE 2. Radial basis function (RBF) neural network architecture.

a neuron for each input feature of the traffic and then feed the hidden layer with the features of the input layer without any change (i.e., there is no weight between the input layer and hidden layer). The hidden layer establishes a nonlinear correspondence between the input space and a space with usually a larger dimension (i.e., the nonlinear RBF transfer functions). Finally, the output layer generates a weighted sum with a linear output. In the case of classification such as our work, the activation function of the output layer becomes the sigmoid activation function.

The neurons of the hidden layer use a Gaussian function as RBF, as illustrated in equation 2.

$$\varphi_j(X) = e^{(-\frac{\|X-X_{C_j}\|}{2\sigma_j})} \tag{2}$$

where X_{C_j}, σ_j and $\varphi_j(X)$ denote its well-pointed center (centroid), spread width (stretch constant) and the response of the j^{th} hidden neuron corresponding to input X, respectively. In this equation, the term of $\|X - X_{C_j}\|$ illustrates the Euclidean distance between the elements of the input vector X and the corresponding centroid of Gaussian X_{C_j} that can be calculated as follows

$$\|X - X_{C_j}\| = \sqrt{\sum_{i=1}^{n}(x_i - X_{C_j})} \tag{3}$$

where $X = [x_1, x_2, \ldots, x_n]$, x_i is feature i of the input layer and n is the number of features in the input layer. The Important point in designing RBF networks is that the nonlinear function of the hidden layer neurons should cover all significant areas of the input vector space.

The outcome values of neurons of the hidden layer are multiplied by the corresponding weights of each neuron and passed to the output neurons. Each output neuron adds up the weighted values. The final summation passes through the sigmoid activation function to classify the input data as DDoS or normal traffic. Equation 4 shows the output function.

$$f(X) = \phi(\sum_{j=1}^{m}(w_j \times \varphi_j(X))) \tag{4}$$

In equation 4, ϕ is the sigmoid activation function. The RBF neural network is used in the proposed technique because (1) the RBF neural network is quick. This issue assists us in detecting DDoS traffic fast. (2) RBF neural network is suitable for the cases where data is in the form of clusters. As App-DDoS requests are in the form of clusters from big to small [4], RBF neural network is a good choice.

F. TRAINING RBF NEURAL NETWORK USING CUCKOO SEARCH ALGORITHM

In our RBF neural network, centroid, spread width, and weight of neurons of the hidden layer are the parameters that should be well tuned during the training procedure of the network. To well train the network, improve the network's performance in terms of accuracy, and converge the network fast, the Cuckoo Search Algorithm (CSA) is utilized. More clearly, (1) CSA can find the global optimum solution with higher success rates [6]. (2) CSA improves performance (accuracy) by utilizing Levy flight. (3) CSA leads to a faster convergence rate.

CSA, which was created by Xin-She Yang and Suash Deb in 2009, is based on some cuckoo species' aggressive brood parasitism and egg-laying technique [14]. CSA has inspired the behavior of the cuckoo birds that lays their eggs in the nests of birds of other species. With a probability of P_a, the host bird discovers the alien eggs and either throws the alien eggs away or abandons the nest. In CSA, each egg placed in the nest is a solution. A better solution would be to place the eggs in safer nests (i.e., the host bird does not notice the cuckoo's eggs). The goal of cuckoos over different generations is to find better solutions. Each nest represents a set of solutions to the problem in which each egg is a solution. In general, CSA is based on three rules:

1) Each cuckoo lays exactly one egg at a time and places it randomly in one of the nests.

2) The nests having the high quality eggs (i.e., the best solutions) are used for the next generation.
3) The number of nests available is constant and the probability of each cuckoo egg being detected by the host bird is P_a.

CSA starts its work by initializing a random population of n host nests. Each host nest includes one egg of a cuckoo bird. Some of these eggs will grow and become adult cuckoos. Other eggs with the probability of P_a are detected and destroyed by the host bird. The amount of eggs grown indicates the suitability of the nests in that area; hence, cuckoos are interested in migrating to those areas. The situation where the largest number of eggs is saved will be a parameter that the cuckoo algorithm intends to optimize. To migrate to the area including best nests, cuckoos could balance the global random walk and local random walk to promote searchability and find a better solution. The global random walk is modeled mathematically as equation 5.

$$X_i^{t+1} = X_i^t + \alpha \otimes Levy(\lambda) \tag{5}$$

where X_i^{t+1} and X_i^t indicate the next and current positions of cuckoo i, respectively. $\alpha > 0$ denotes the step size that is normally considered one. The $\otimes$ is an entry-wise multiplication. $Levy(\lambda)$ is the Levy distribution with rate of λ. In our algorithm, equation 6 models equation 5.

$$X_i^{t+1} = X_i^t + r \otimes (X_{best}^t - X_i^t) \tag{6}$$

where r is the deviation parameter, it is a vector with random values a uniform distribution in the range of [0, 1]. To model the local random walk of cuckoo i, two cuckoos with indices of j and k are selected randomly between all cuckoos, and the next position of cuckoo i is calculated as equation 7.

$$X_i^{t+1} = X_i^t + \alpha \otimes H(P_a - \epsilon) \otimes (X_j^t - X_k^t) \tag{7}$$

In equation 7, ϵ is a random value generated based on the normal distribution.

G. THE PSEUDOCODE OF THE PROPOSED ALGORITHM

Algorithm 1 shows the pseudocode of the proposed algorithm. Let us explain the algorithm with a motivational example. Consider an RBF network that has two input neurons (i.e., two features), three neurons in the hidden layer (the centroid (X_c) and the spread width (σ) of each neuron should be trained), and two output neurons. Fig. 3 illustrates the position vector of a cuckoo. The vector includes $X_{11}, X_{21}, \sigma_1, X_{12}, X_{22}, \sigma_2, X_{13}, X_{23}, \sigma_3, w_1, w_2$, and w_3. $X_{ij} = \|x_i - X_{C_j}\|$ distance vector between neuron x_i of the input layer and centroid X_{c_j} of neuron j of hidden layer.

1) THE INITIALIZATION STEP

In the initialization step, the position vector of all cuckoos is randomly assigned. The RBF function for each neuron of the hidden layer is calculated according to equation 2. Next, the output function is calculated as equation 4. Next, the Mean square error (MSE) as the cost function is calculated for the

Algorithm 1 The Pseudocode of the Proposed Algorithm

Initialization phase:
Create an initial population of N solutions
for i=1 to N **do** ▷ N is the number of cuckoos
  for j=1 to m **do** ▷ m is the number of neurons of hidden layer
    $X_{C_j} \leftarrow$ *random value in* [0, 1]
    $\sigma_j \leftarrow$ *random value in* [0, 1]
    $w_j \leftarrow$ *random value in* [0, 1]
  end for
end for
Initialize P_a, ϵ, t^{max}
for i=1 to N **do**
  for j=1 to m **do**
    Calculate RBF function for neuron i based on equation 2
  end for
  Calculate output function for the output neuron based on equation 4
  Calculate MSE for the output neuron
end for
Find minimum MSE
$t \leftarrow 0$
Iteration phase:
while ($t < t^{max}$ or $MSE > \epsilon$) **do**
  Create new population using Levy flight equation 6
  for i=1 to N **do**
    for j=1 to m **do**
      Calculate RBF function for neuron i based on equation 2
    end for
    Calculate output function for the output neuron based on equation 4
    Calculate MSE for the output neuron
  end for
  Create new population using local random walk equation 7
  for i=1 to N **do**
    for j=1 to m **do**
      Calculate RBF function for neuron i based on equation 2
    end for
    Calculate output function for the output neuron based on equation 4
    Calculate MSE for the output neuron
  end for
  Find minimum MSE
  Best solution ← the solution with minimum MSE
end while
return Best solution

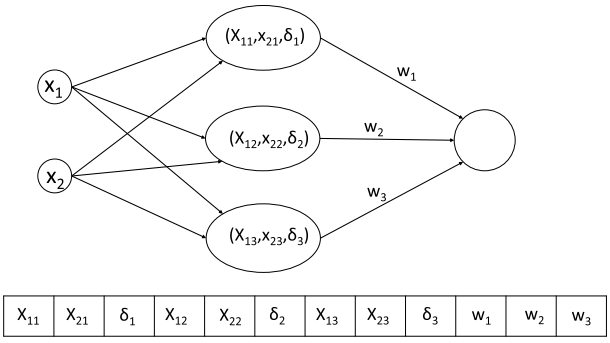

FIGURE 3. The position of a cuckoo in the presented example.

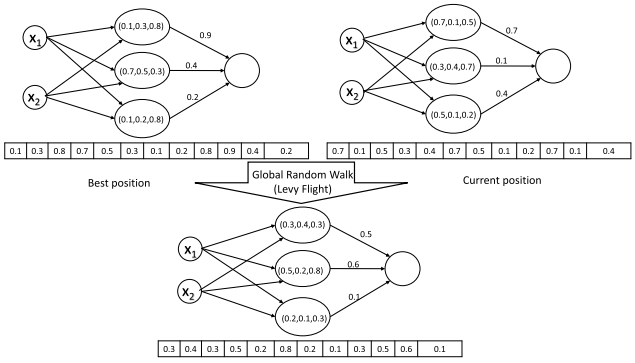

FIGURE 4. Global random walk (Levy flight) for the presented example.

output. All cuckoos are sorted based on their MSE value in descending order. The cuckoo with the minimum value of MSE is considered the best cuckoo.

2) THE ITERATION STEP

In this step, the new position of the cuckoo is calculated with both global random walk (Levy flight) and local random walk according to equations 6 and 7, respectively. Fig. 4 illustrates the global random walk for this example.

We measure MSE for three positions: The current location, the new position after the global random walk, and the new position after the local random walk. The position, which results in the minimum MSE, is selected as the best position of the cuckoo for the current time t. The best position is saved in X^t_{best} for the next iterations.

3) ALGORITHM STOP CONDITION

The algorithm is continued until the number of iterations either reaches t_{max} (the upper bound limitation of iterations) or the achieved MSE in the previous step becomes smaller than epsilon. The algorithm's output is a cuckoo population member with the best values of centroids X_c, σ, and weights of neurons of the hidden layer in the RBF neural network so that MSE is the lowest possible. The RBF neural network is trained with the best solution. Next, the trained RBF neural network is used to classify traffic packets as App-DDoS or legitimate.

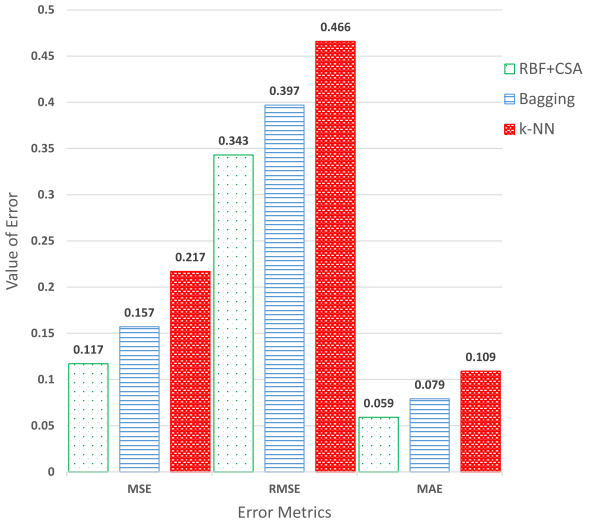

FIGURE 5. The comparison between CSA-trained RBF, Bagging and k-NN based on error metric, per training set.

H. MITIGATION TECHNIQUE

The victim server can shut down all connections belonging to attack traffic when the attack traffic is detected. The victim server also can send CAPTCHA puzzles to attack sources. The sources who could not solve the CAPTCHA are bot machines, and their connection is terminated.

IV. EXPERIMENT AND RESULTS

A. SIMULATION SETUP

We use the MATLAB 2020 software environment to implement our experiment's simulation stages. In the following experiments, the population size of the CSA algorithm is set to 50 in the proposed method. The maximum iterations in the proposed method are set to 500, and P_a is set to 0.25.

B. PERFORMANCE METRICS

In order to evaluate the effectiveness of the proposed technique, we look at the outcomes of the proposed technique and other methods. The results of the proposed technique (RBF + CSA) are compared with the Bagging algorithm and k-NN classifiers using four error metrics: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Sum of Squares Error (SSE). We further compare the performance of the proposed technique with Bagging and k-NN by calculating five more performance metrics which are Sensitivity (Sn), Specificity (Sp), Positive Predictive Value (PPV), Negative Predictive Value (NPV), and Precision. Finally, we compare the precision metric of our proposed approach to many other well-known ML techniques. All experiments are run for training, testing, and the whole dataset.

C. RESULTS

Figures 5, 6 and 7 show the comparison results between the proposed technique, Bagging, and k-NN based on error

TABLE 3. The comparison between CSA-trained RBF, Bagging, and K-NN based on error metrics.

	Training Set			Testing Set			Whole set		
	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE
RBF + CSA	0.117	0.343	0.059	0.16	0.4	0.08	0.126	0.355	0.063
Bagging	0.157	0.397	0.079	0.172	0.414	0.086	0.16	0.4	0.08
K-NN	0.217	0.466	0.109	0.189	0.435	0.094	0.212	0.46	0.106

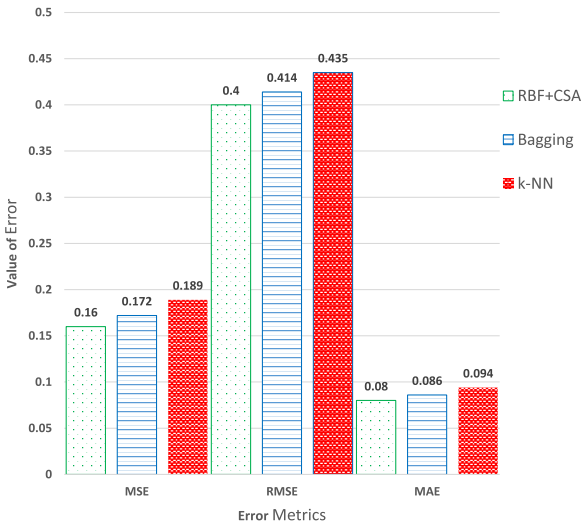

FIGURE 6. The comparison between CSA-trained RBF, Bagging and k-NN based on error metric, per testing set.

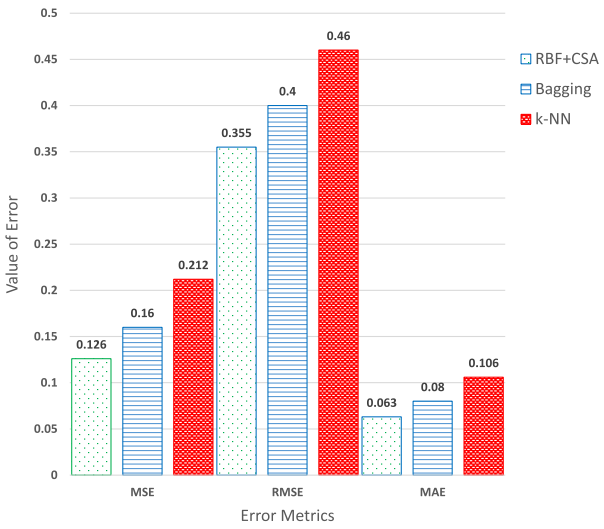

FIGURE 7. The comparison between CSA-trained RBF, Bagging and k-NN based on error metric, per the whole set.

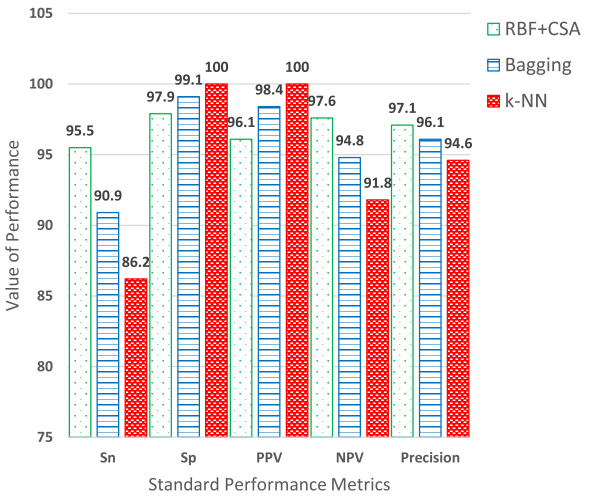

FIGURE 8. The comparison between CSA-trained RBF, Bagging and k-NN based on standard performance metrics, per training set.

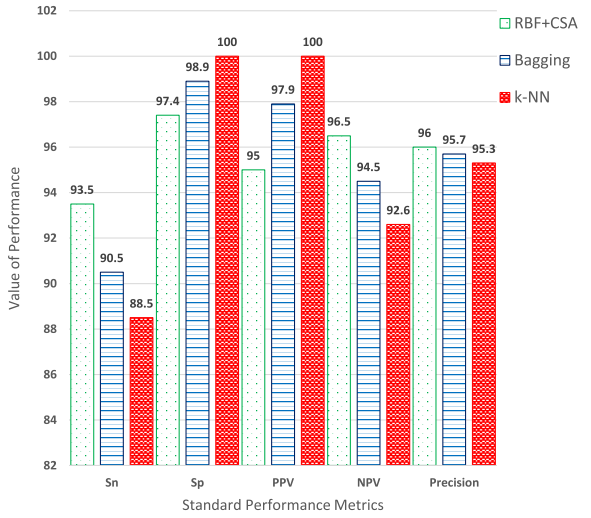

FIGURE 9. The comparison between CSA-trained RBF, Bagging and k-NN based on standard performance metrics, per testing set.

metrics for the training dataset, testing dataset, and the whole dataset, respectively. Table 3 shows the same result, but via a table. As can be seen, the proposed technique improves the MSE error metric on average by 22% and 57% in comparison with Bagging and k-NN, respectively. The improvement for the RMSE error metric is 10% and 24% in compared to Bagging and k-NN, respectively. The proposed technique enhances the MAE error metric on average by 22% and 56% compared to Bagging and k-NN.

Figures 8 (training dataset), 9 (testing dataset), and 10 (the whole dataset) compare the proposed technique with Bagging and k-NN for the performance metrics of Sensitivity (Sn), Specificity (Sp), Positive Predictive Value (PPV), Negative Predictive Value (NPV), and Precision. Table 4 shows standard performance metrics with a table. As can be seen, the proposed technique outperforms the Bagging algorithm and the k-NN technique except in SP and PPV.

TABLE 4. The comparison between CSA-trained RBF, Bagging, and K-NN based on standard performance metrics.

	Training Set					Testing Set					Whole set				
	Sn	Sp	PPV	NPV	Precision	Sn	Sp	PPV	NPV	Precision	Sn	Sp	PPV	NPV	Precision
RBF + CSA	95.5	97.9	96.1	97.6	97.1	93.5	97.4	95	96.5	96	95.1	97.8	95.9	97.4	96.9
Bagging	90.9	99.1	98.4	94.8	96.1	90.5	98.9	97.9	94.5	95.7	90.8	99.1	98.3	94.8	96
K-NN	86.2	100	100	91.8	94.6	88.5	100	100	92.6	95.3	86.7	100	100	91.9	94.7

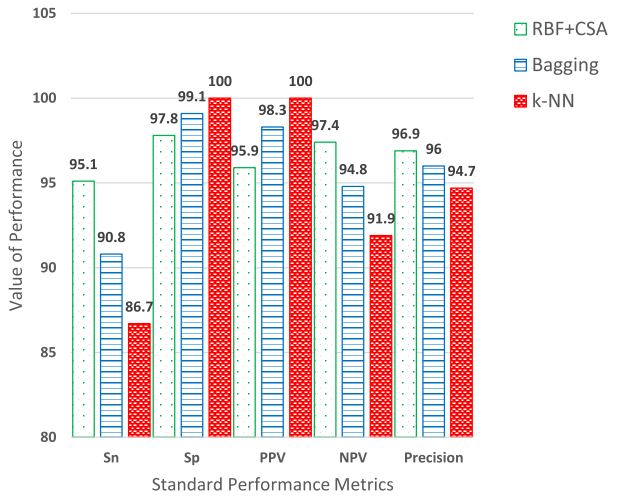

FIGURE 10. The comparison between CSA-trained RBF, Bagging and k-NN based on standard performance metrics, per the whole set.

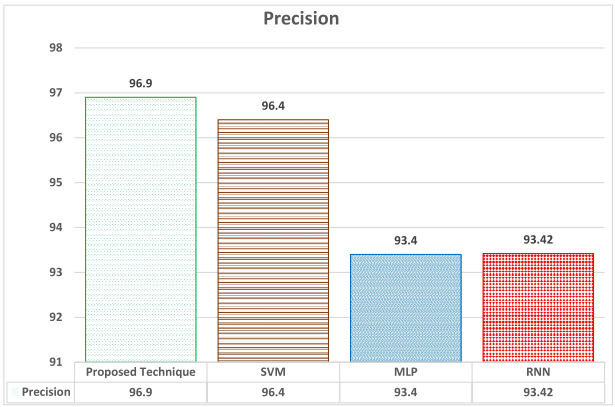

FIGURE 11. The comparison between CSAtrained RBF, SVM, MLP and RNN, per testing set based on Precision metric.

In terms of precision metrics, we compared our proposed technique with Support Vector Machine (SVM), Multilayer Perceptron (MLP), and Recurrent Neural Network(RNN). Figure 11 compares the accuracy between the proposed technique, SVM, MLP, and RNN. As can be seen, our proposed technique enhances the accuracy on average by 0.5%, 3.6%, and 3.5% compared with SVM, MLP, and RNN, respectively.

V. CONCLUSION

This paper proposes a technique based on machine learning to cope with App-DDoS attacks. The proposed technique is a hybrid method of Radial Basis Function (RBF) neural network and Cuckoo Search Algorithm (CSA). RBF neural network is used for classification, while CSA is used to train the RBF network. The feature selection procedure of the dataset (NSL-KDD) is done based on the Genetic Algorithm (GA). Several experiments are conducted to evaluate and compare the proposed technique with the Bagging algorithm, k-NN classifier, SVM, MLP, and RNN. The simulation results, i.e., accuracy = 96.9%, MSE = 0.134, RMSE = 0.366, and MAE = 0.067, clearly indicate the superiority of the proposed technique against standard machine learning techniques that are used to detect App-DDoS attacks. The proposed technique enhances the accuracy on average 2% compared with all mentioned machine learning-based techniques for all sets of datasets (training dataset, testing dataset, and the whole dataset).

REFERENCES

[1] M. J. Alam, P. Ouellet, P. Kenny, and D. O'Shaughnessy, "Comparative evaluation of feature normalization techniques for speaker verification," in *Proc. Int. Conf. Nonlinear Speech Process.* Berlin, Germany: Springer-Verlag, 2011, pp. 246–253.

[2] D. Alghazzawi, O. Bamasag, H. Ullah, and M. Z. Asghar, "Efficient detection of DDoS attacks using a hybrid deep learning model with improved feature selection," *Appl. Sci.*, vol. 11, no. 24, p. 11634, Dec. 2021.

[3] S. Banerjee and P. S. Chakraborty, "To detect the distributed denial-of-service attacks in SDN using machine learning algorithms," in *Proc. Int. Conf. Comput., Commun., Intell. Syst. (ICCCIS)*, Feb. 2021, pp. 966–971.

[4] H. Beitollahi and G. Deconinck, "ConnectionScore: A statistical technique to resist application-layer DDoS attacks," *J. Ambient Intell. Humanized Comput.*, vol. 5, no. 3, pp. 425–442, Jun. 2014.

[5] H. Beitollahi and G. Deconinck, "A four-step technique for tackling DDoS attacks," *Proc. Comput. Sci.*, vol. 10, pp. 507–516, Jan. 2012.

[6] P. Civicioglu and E. Besdok, "A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms," *Artif. Intell. Rev.*, vol. 39, no. 4, pp. 315–346, Apr. 2013.

[7] M. Dash and H. Liu, "Feature selection for classification," *Intell. Data Anal.*, vol. 1, nos. 1–4, pp. 131–156, 1997.

[8] Y. Feng, J. Li, and T. Nguyen, "Application-layer DDoS defense with reinforcement learning," in *Proc. IEEE/ACM 28th Int. Symp. Quality Service (IWQoS)*, Jun. 2020, pp. 1–10.

[9] H. Gao, X. Wang, F. Cao, Z. Zhang, L. Lei, J. Qi, and X. Liu, "Robustness of text-based completely automated public Turing test to tell computers and humans apart," *IET Inf. Secur.*, vol. 10, no. 1, pp. 45–52, 2016.

[10] P. Maity, S. Saxena, S. Srivastava, K. S. Sahoo, A. K. Pradhan, and N. Kumar, "An effective probabilistic technique for DDoS detection in OpenFlow controller," *IEEE Syst. J.*, vol. 16, no. 1, pp. 1345–1354, Mar. 2022.

[11] K. N. Mallikarjunan, K. Muthupriya, and S. M. Shalinie, "A survey of distributed denial of service attack," in *Proc. 10th Int. Conf. Intell. Syst. Control (ISCO)*, Coimbatore, India, Jan. 2016, pp. 83–89.

[12] Ismail, M. I. Mohmand, H. Hussain, A. A. Khan, U. Ullah, M. Zakarya, A. Ahmed, M. Raza, I. U. Rahman, and M. Haleem, "A machine learning-based classification and prediction technique for DDoS attacks," *IEEE Access*, vol. 10, pp. 21443–21454, 2022.

[13] F. Ridzuan and W. M. N. W. Zainon, "A review on data cleansing methods for big data," *Proc. Comput. Sci.*, vol. 161, pp. 731–738, Jan. 2019.
[14] X.-S. Yang and S. Deb, "Engineering optimisation by cuckoo search," *Int. J. Math. Model. Numer. Optim.*, vol. 1, no. 4, pp. 330–343, 2010.
[15] S. K. Sahu, D. P. Mohapatra, J. K. Rout, K. S. Sahoo, and A. K. Luhach, "An ensemble-based scalable approach for intrusion detection using big data framework," *Big Data*, vol. 9, no. 4, pp. 303–321, Aug. 2021.
[16] C.-S. Shieh, W.-W. Lin, T.-T. Nguyen, C.-H. Chen, M.-F. Horng, and D. Miu, "Detection of unknown DDoS attacks with deep learning and Gaussian mixture model," *Appl. Sci.*, vol. 11, no. 11, p. 5213, Jun. 2021.
[17] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl.*, Jul. 2009, pp. 1–6.
[18] H. Vafaie and K. De Jong, "Genetic algorithms as a tool for feature selection in machine learning," in *Proc. 4th Int. Conf. Tools With Artif. Intell. (TAI)*, Arlington, VA, USA, Nov. 1992, pp. 200–203.
[19] M. Yoon, "Using whitelisting to mitigate DDoS attacks on critical internet sites," *IEEE Commun. Mag.*, vol. 48, no. 7, pp. 110–115, Jul. 2010.
[20] J. Yu, C. Fang, L. Lu, and Z. Li, "Mitigating application layer distributed denial of service attacks via effective trust management," *IET Commun.*, vol. 4, no. 16, pp. 1952–1962, Nov. 2010.
[21] N. M. Yungaicela-Naula, C. Vargas-Rosales, and J. A. Perez-Diaz, "SDN-based architecture for transport and application layer DDoS attack detection by using machine and deep learning," *IEEE Access*, vol. 9, pp. 108495–108512, 2021.
[22] M. J. Zaki and W. Meira, Jr., *Data Mining and Machine Learning: Fundamental Concepts and Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2019.
[23] M. Zeeshan, Q. Riaz, M. A. Bilal, M. K. Shahzad, H. Jabeen, S. A. Haider, and A. Rahim, "Protocol-based deep intrusion detection for DoS and DDoS attacks using UNSW-NB15 and bot-IoT data-sets," *IEEE Access*, vol. 10, pp. 2269–2283, 2022.
[24] F. Zhang, C. Luo, C. Lan, and J. Zhan, "Benchmarking feature selection methods with different prediction models on large-scale healthcare event data," *BenchCouncil Trans. Benchmarks, Standards Eval.*, vol. 1, no. 1, Oct. 2021, Art. no. 100004.
[25] B. B. Zhu, J. Yan, G. Bao, M. Yang, and N. Xu, "Captcha as graphical passwords—A new security primitive based on hard AI problems," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 6, pp. 891–904, Jun. 2014.

HAKEM BEITOLLAHI received the B.S. degree in computer engineering from the University of Tehran, in 2002, the M.S. degree from the Sharif University of Technology, Tehran, Iran, in 2005, and the Ph.D. degree from the Katholieke Universiteit Leuven, Belgium, in 2012. He is currently an Assistant Professor and the Head of the Hardware and Computer Systems Architecture Branch, School of Computer Engineering, Iran University of Science and Technology. His research interests include network security, real-time systems, fault tolerance, and dependability.

DYARI MOHAMMED SHARIF received the B.Sc. degree in computer science/software from Soran University, Kurdistan Region, Iraq, in 2019, where he is currently pursuing the M.Sc. degree in artificial intelligence. His research interest includes the development of DoS/DDoS detection/prevention techniques. His awards and honors include the Top Student Award at the Department Level, Top Student at the Faculty Level, and Top Ranked Research at the Department.

MAHDI FAZELI received the M.Sc. and Ph.D. degrees in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2005 and 2011, respectively. He is currently an Associate Professor with the School of Information Technology, Halmstad University, Sweden. He has authored or coauthored more than 50 papers in reputable journals and conferences. His current research interests include system level power analysis and management, real-time systems, hardware security, and reliability modeling and evaluation.