Future Generation Computer Systems 127 (2022) 276–285

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A lightweight supervised intrusion detection mechanism for IoT networks

Souradip Roy[a], Juan Li[a,*], Bong-Jin Choi[b], Yan Bai[c]

[a] Department of Computer Science, North Dakota State University, United States of America
[b] Department of Statistics, North Dakota State University, United States of America
[c] School Engineering and Technology, University of Washington Tacoma, United States of America

ARTICLE INFO

Article history:
Received 4 December 2020
Received in revised form 13 September 2021
Accepted 17 September 2021
Available online 29 September 2021

Keywords:
Intrusion detection
IoT network
Machine learning
Security

ABSTRACT

As the Internet of Things (IoT) is becoming increasingly popular, we have experienced more security breaches that are associated with the connection of vulnerable IoT devices. Therefore, it is crucial to employ intrusion detection techniques to mitigate attacks that exploit IoT security vulnerabilities. However, due to the limited capabilities of IoT devices and the specific protocols used, conventional intrusion detection mechanisms may not work well for IoT environments. In this paper, we propose a novel intrusion detection model that uses machine learning to effectively detect cyber-attacks and anomalies in resource-constraint IoT networks. Through a set of optimizations including removal of multicollinearity, sampling, and dimensionality reduction, our model can identify the most important features to detect intrusions using much fewer training data and less training time. Extensive experiments were performed on the CICIDS2017 and NSL-KDD datasets respectively to evaluate the proposed approach. The experimental results on two popular datasets show that our model has a high detection rate and a low false alarm rate. It outperforms existing models in multiple performance metrics and is consistent in classifying major cyber-attacks, respectively. Most importantly, unlike traditional resource-intensive intrusion detection systems, the proposed model is lightweight and can be deployed on IoT nodes with limited power and storage capabilities.

1. Introduction

The Internet of Things (IoT) connects uniquely identifiable heterogeneous embedded computing devices in the physical environment to the Internet. It offers pervasive connectivity of devices, systems, and services, and has been widely used in our daily life. We have witnessed the explosion in connected devices and IoT technologies ranging from smart homes, smart hospitals to drones and even autonomous bots. As more homes and businesses adopt IoT devices, a large number of connected IoT devices will revolutionize how data is processed and consumed. IoT continues to enjoy an even greater surge in popularity, but on the other hand, the associated security risks are also surging.

Many efforts have been applied to enhance IoT security, including enforcement of encryption on data transmitted in the network, strict access control mechanisms for data confidentiality, and various privacy and trust policies and management among users and IoT devices. However, even with these mechanisms, IoT networks are still vulnerable to many kinds of cyber-attacks. Over the last few years, there have been a tremendous number of IoT-centric attacks: from more widescale, powerful distributed denial of service (DDoS) attacks, to the hacking of baby monitors. Intrusion Detection Systems (IDS) have, therefore, been used as another layer of defense in order to better protect the legitimate operation of the IoT devices.

IDS are devices or software applications that monitor network or system behaviors for malicious activities or policy violations and send reports to a management station. Many IDS have been proposed to improve Internet hosts and network security. However, we cannot deploy traditional IDS systems directly onto IoT networks because of the special characteristics of IoT networks: nodes in IoT networks are deployed in resource-constrained devices, e.g., with limited power, computing, communication, and storage capabilities. This requires significant simplification, optimization, and adaptation of existing security techniques. In addition, IoT network uses different protocol stacks and standards. These necessities require to devise of security mechanisms accordingly.

To address the challenges faced by IDS in IoT networks, in this paper, we propose a novel machine learning-based intrusion detection mechanism, which has the following advantages over traditional IDS systems:

[*] Correspondence to: Computer Science Department, North Dakota State University, Fargo, ND, 58078, USA.
E-mail address: j.li@ndsu.edu (J. Li).

https://doi.org/10.1016/j.future.2021.09.027
0167-739X/© 2021 Elsevier B.V. All rights reserved.

- Lightweight. Through a set of optimization mechanisms including removal of multicollinearity, data scaling, dimensionality reduction, and sampling, our mechanism can quickly identify the most important feature and dramatically reduce the size of the features and data points needed for abnormality detection, and consequently, lower computational complexity. The lightweight feature makes the algorithm appropriate for IoT networks with resource constraints.
- Less training time. The proposed algorithm needs a much smaller training set. It, thus, dramatically reduces the training time compared to the existing techniques.
- Higher detection rate for less popular cyber-attacks with infrequent observations, such as the User to Root attack.
- Lower false positive and false negative rates. Our algorithm reduces the false positive rate while keeping or improving the accuracy compared to the existing techniques.

The rest of the paper is organized as follows. Section 2 surveys related work on various IDS for IoT networks and systems. Section 3 describes our proposed methodology in detail. Section 4 presents our evaluation results. Finally, in Section 5, we provide conclusions and future work directions.

2. Related work

According to Zarpelão et al. [1], intrusion detection in IoT can be classified into four categories depending upon the detection mechanism used in the system: signature-based, anomaly-based, specification-based, and hybrid. Signature-based and anomaly-based are the most widely used two approaches.

Signature-based IDS detects cyber-attacks the same way as virus scanners, by searching for specific patterns or signatures stored in the IDS internal databases. If any system or network activity matches with stored patterns, it will be detected as an intrusion. Sheikh et al. [2] proposed a lightweight signature-based IDS for IoT networks. The IDS system consists of four parts including signature generator, pattern generator, intrusion detection engine, and output engine. They tested their system using the NSL-KDD dataset. Liu et al. [3] proposed another signature-based IDS for IoT. It uses an Artificial Immune System to detect attacks. Attack signatures are stored in immune cells which can be further classified. The computation overhead of this approach is high. However, the authors did not explain how this approach can be applied to resource-limited IoT environments. Rebbah et al. [4] proposed a signature-based IDS, named IoTSecurity, for IoT systems using Cloud. It calculates the temporary and spatial profile of each client based on the data of its request. Attacks are detected based on matching the profile with the signature. While signature-based IDS is very efficient at detecting known cyber-attacks, it is not effective in detecting new cyber-attacks.

Anomaly-based IDS can identify an unknown activity by comparing it with a normal behavior profile and then classifying it as either normal or anomalous. Anomaly-based IDS are different from signature-based IDS that only detect attacks by matching previously created signatures: Anomaly-based IDS are effective in identifying new intrusions. Many approaches have been proposed to implement anomaly-based intrusion detection systems, among which deep learning has been widely used. Larijani et al. [5] proposed a random neural network-based intrusion detection system (RNN-IDS) for IoTs. After selecting features, RNN-IDS trains and tests neurons at different learning rates with the NSL-KDD dataset. The performance is evaluated with the benchmark NSL-KDD dataset for binary classification. The system can obtain an accuracy of 94.50%. Similarly, Yin et al. [6] also proposed a deep learning approach that uses a recurrent neural network (RNN) for anomaly-based intrusion detection. Diro and Chilamkurti [7] applied deep learning for attack detection in social IoT networks. Their experiments demonstrated that the deep model is more effective in attack detection than its shallow counterparts. Alom et al. [8] used deep belief neural (DBN) networks to create IDS and tested the system with the NSL-KDD dataset. The proposed method achieved a detection accuracy of about 97.5%. Ahsan and Nygard [9] proposed an approach using a hybrid algorithm of Convolutional Neural Network (CNN) and Long Short Term Memory (LSTM). Ieracitano et al. [10] proposed a statistical analysis-driven optimized deep learning system for intrusion detection. The system extracts features using big data visualization and statistical analysis methods, followed by a deep autoencoder (AE) for potential threat detection.

Different strategies have been proposed to reduce the false prediction rate and improves accuracy. For example, Song et al. [11] propose a multiple decision-based classification method to identify misclassified data, and classify them into three different classes, called a malicious, benign, and ambiguous dataset. They evaluated their approach with the recent real-world network traffic data, Kyoto2006+ datasets. Khan et al. [12] proposed a hybrid-multilevel anomaly prediction approach to deal with unbalanced intrusion data to improve the accuracy of the detection system.

Many machine learning-based algorithms have been applied for IDS based on the CICIDS2017 dataset [13]. A deep neural network (DNN)-based IDS is proposed to detect and classify cyberattacks [14]. Researchers have performed a comprehensive evaluation of DNNs and other classical machine learning classifiers using different datasets including KDDCup 99, NSL-KDD [15], UNSW-NB15 [16], Kyoto [17], WSN-DS [18], and CICIDS2017 [13]. They confirmed that DNNs perform well in comparison with the classical machine learning classifiers. Sethi et al. proposed a reinforcement learning-based IDS that employs Deep Q-Network logic in multiple distributed agents and uses attention mechanisms to classify network attacks [19]. They also tested their data using NSL-KDD and CICIDS2017 and demonstrated improved performance. Yulianto et al. apply a set of mechanisms including oversampling, Principal Component Analysis (PCA), and Ensemble Feature Selection (EFS) to improve the performance of AdaBoost-based IDS on CICIDS 2017 Dataset [20]. Another work also tries to solve the data imbalance problem by using the Generative Adversarial Networks (GAN) model [21]. It uses a deep learning approach to address data imbalances and Random Forest to classify attacks.

Other machine learning algorithms were also explored for IDS based on the NSL-KDD dataset. Li et al. [15] proposed K-Nearest Neighbor (KNN) classification algorithm in wireless sensor networks for intrusion detection. Shapoorifard and Shamsinejad [22] worked on combining the KNN classifier with K-MEANS clustering for intrusion detection. Ingre et al. [23] proposed a Decision Tree-based IDS and tested it with the NSL-KDD dataset. Farnaaz et al. [24] used Random Forest for detection anomaly. As specified by Sasha, "Anomaly-based Intrusion Detection at both the network and host levels have a few shortcomings; namely a high false-positive rate and the ability to be fooled by a correctly delivered attack" [25].

3. Methodology

The goal of this research is to construct an efficient classifier model from limited information and resources to detect various attacks as accurately as possible. This is achieved by a series of mechanisms including (A) removal of multicollinearity, which also reduces the dimensionality of data; (B) sampling data to reduce the required training data size and improve detection rate; (C) further dimensionality removal to reduce overfitting, and (D) effective classification algorithm. In the following subsections, we present the details of these four mechanisms.

3.1. Data segmentation and feature ranking

Feature ranking is the process of assigning a score to each of the independent variables. The information contained in these independent variables helps the model to correctly predict the class of the samples. This ranking method helps us to determine the most important features in the dataset. This technique will help add non-linearity to our reduced dataset which is described in Section 3.3. To correctly identify the important features and rank them based on their importance we have used regression analysis. A regression model may cause problems to correctly interpret the important features if multicollinearity exists within the feature set. Multicollinearity is a condition when there is a significant dependency or association between the independent variables or the predictor variables. Multicollinearity may misleadingly inflate the standard error of a data model in an excessive amount [26]. In the presence of multicollinearity, the coefficient may provide high estimates of changes in the multiple regressions when only small changes can be seen in the model or the data. Therefore, it is important to detect and remove multicollinearity. We use the Variance Inflation Factor (VIF) [27] to identify multicollinearity. As defined in Eq. (1), VIF measures multicollinearity among the independent variables in a feature set by using multiple regression analysis.

$$VIF = \frac{1}{1 - {R_i}^2} \tag{1}$$

In Eq. (1), R_{i}^2, termed as r-squared, is the regression of the ith independent variable from the feature set. r-squared is calculated as:

$${R_i}^2 = 1 - \frac{\Sigma\,(Y_i - \mu)}{\Sigma\,(Y_i - \bar{y})} \tag{2}$$

in which Y_{i} is the actual class of the data point and μ is the predicted value by the logistic regression model. The numerator in the fraction is known as sum squared regression and the denominator is called sum squared total. The sum squared regression is the partial model where only the independent variable is considered. VIF indicates how much the variance (behavior) of an independent variable is influenced by its interaction with others. If there is a significant presence of multicollinearity then the VIF of the independent variable will be significantly high. Once the VIF for all the features is obtained, it is important to determine a threshold value, above which all the features will be called multicollinearity.

Traditionally there are two ways to remove multicollinearity: one is to drop features with high VIF; the other is to apply the Partial Least Squares Regression (PLS) [28] or Principal Component Analysis (PCA) [29] to all features above a threshold to merge them to one feature [26]. Both methods may cause information loss because of removing features directly. We try to keep as much as possible information on the original dataset to avoid information loss. For this purpose, we use data segmentation that will generate new features. These new features represent the original features of the dataset without multicollinearity. We creatively propose to group variables (with VIF beyond the threshold) into n groups according to their VIF. In our experiment, we set n = 5. The 5 groups include least multicollinearity, medium multicollinearity, moderate multicollinearity, high multicollinearity, and very high multicollinearity. We then apply PCA to represent features in a group in a lower dimension. PCA is a mathematical algorithm that reduces data dimensionality while retaining most of the variation in the dataset. This is achieved by identifying directions (i.e., principal components), along which the variation in the data is maximal.

In order to decide on the number of principal components needed for each group, we have used a scree plot [30]. This plotting technique helps us to identify the minimum number of principal components needed to retain the required percentage of variance in the data. In our experiment, we choose the number of principal components for each group to be the minimum number to achieve 90% of the variance explained. The goal is to maximize the percentage of variance while keeping the smallest number of principal components. The 90% of variance defines most of the information contained by the principal component compared to the original features. Once the required number of principal components for each group is obtained, we use the principal components as an independent feature. The linear combination of all the data points obtained from the principal components is treated as new observations. We dropped all features having VIF values higher than the collinearity threshold t from the original dataset and appended these newly generated features. Using this approach, in our experiment on the NSL-KDD dataset, after removing multicollinearity from the dataset, we reduced the features from 41 to 28. After the removal of multicollinearity, we used a regression model to rank the features and understand the importance of each of the independent variables to define the class of the samples.

We apply data normalization to normalize the range of independent variables. The floor and ceiling values of the data points for each feature may not fall in the same range and the distribution of the data is unknown, using standardization allows us to rescale the features which have the properties of a standard normal distribution. We use z-score [27] normalization that is defined in Eq. (3):

$$Z = \frac{X_i - \bar{x}}{S_x} \tag{3}$$

in which X_{i} is the individual value belonging to an observation of a feature, $\bar{x}$ is the mean value of each feature and S_{x} is the standard deviation. Z-score allows the data points to follow a normal distribution curve and ensures each feature's data distribution has a mean value of 0 and a standard deviation of 1. If the dataset has unextreme outliers, using scaling, we do not need to remove them.

3.2. Sampling

Due to the nature of imbalanced network attacks [31], the occurrences of some cyber-attacks are destined to be much more frequent than some others. Therefore, the dataset used to develop the classification model is often unbalanced, i.e., classes are unevenly distributed. For example, compared with other attacks, the number of User to Root attacks is much smaller in our experimental dataset. Class imbalance can give rise to a series of problems, such as a biased model towards predicting the majority class, misclassification of observations, overfitting of the model, etc.

One of our goals is to reduce the training time yet be effective in detecting anomalies. We use sampling techniques to select individuals to draw statistical inferences about the population. Specifically, we apply undersampling on the majority classes and oversampling on the minority ones. Random undersampling with replacement is used where a sample is selected from the dataset in which every data has equal chances of being selected. The probability of each sample being selected from the population is 1/N where N is the size of the population.

The oversampling method we have used is SMOTE [32]. It uses the concept of nearest neighbors to generate synthetic data for a minority class. The new synthetic data are generated between existing minority instances. This method takes three parameters to generate new instances which are T, N, k where T is the number of minority class samples, N is the percentage explaining the minority class to be over-sampled and k is the number of nearest neighbors that need to be considered for the oversampling technique.

Under- and oversampling may cause the cross-class nearest neighbors' problem, a complication in which a single data point may have equidistant neighbors falling in different categories. When this happens, it is difficult to place the data point to its correct class. Therefore, it is important to eliminate these data points; otherwise, they can cause misclassification. We use Tomek Link [33] to clean up overlap between classes. A Tomek Link is defined as follows: given two samples s_i, and s_j, $d(s_i, s_j)$ is the distance between s_i and s_j. s_i, s_j is called a Tomek link if there are no other samples s_o, such that $d(s_i, s_o)< d(s_i, s_j)$ or $d(s_j, s_o)< d(s_i, s_j)$. When two samples form a Tomek link, either one of these samples is noise or both of them are close to the class border. Removing such samples helps us to create well-defined classes, and consequently improving classification performance.

3.3. Dimensionality reduction and addition of non-linearity

After removing multicollinearity and sampling, we reduce the number of features and samples used in the dataset. These approaches dramatically reduce the computation complexity and help fast response in detecting attacks in real-time. However, as the number of observations gets reduced, the sample density of the dataset will decrease. The result of decreased sample density and sparsity in the dataset will cause a model overfitting problem, i.e., corresponding too closely to the training data, but fail to fit additional data or predict the future. To solve this problem, dimensionality reduction is applied. A dimensionality reduction is a statistical procedure where a set of independent variables are projected to a lower dimension using a set of principal variables. We utilize the PCA approach that uses unsupervised learning to detect hidden patterns within the data. Assume that we have a dataset M with dimension N

$$M = \begin{bmatrix} X_{11} & \cdots & X_{1N} \\ \vdots & \ddots & \vdots \\ X_{N1} & \cdots & X_{NN} \end{bmatrix} \tag{4}$$

M is a N×N dimensional Matrix, in which

- X_{ij} are the data points where $1 \leq i \leq N$ and $1 \leq j \leq N$.
- Each column in M is an independent variable F_k where $1 \leq k \leq N$

The covariance score between F_k of M can be calculated as

$$cov\,(X, Y) = \frac{1}{N-1} \Sigma \left(X_p - \overline{X}\right) \left(Y_p - \overline{Y}\right)' \tag{5}$$

where $X, Y \in F, X \neq Y, X_p \in X, Y_p \in Y$. The resulting matrix is the covariance matrix A.

$$A = \begin{bmatrix} M_{11} & \cdots & M_{1N} \\ \vdots & \ddots & \vdots \\ M_{N1} & \cdots & M_{NN} \end{bmatrix} \tag{6}$$

in which M_{ij} represents the covariance score between each independent variable.

After obtaining the covariance matrix A, we need to find the eigenvalues. Each eigenvalue is considered as a principal component. The eigenvalues are used to determine the eigenvector of the matrix. We determine the eigenvalue by the following equation:

$$\det\left(A - \lambda I\right) = 0 \tag{7}$$

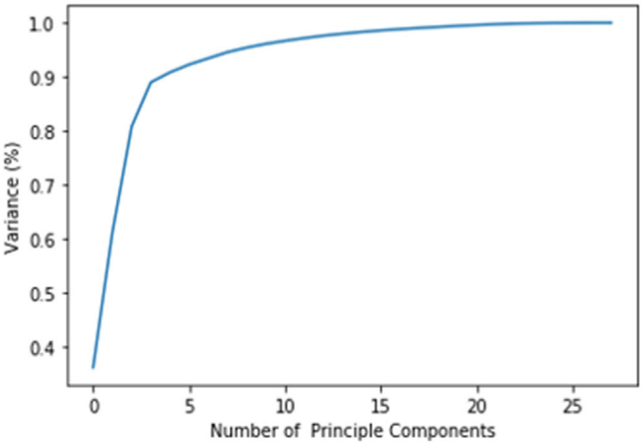


Fig. 1. Number of Principal Components defining the percentage of explained variance by the dataset.

$$I = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix} \tag{8}$$

I is the identity matrix of the same dimension as a covariance matrix. Then I is multiplied with scalar λ, and the result matrix is λI:

$$\lambda I = \begin{bmatrix} \lambda & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda \end{bmatrix} \tag{9}$$

The result of the determinant of Eq. (7) will provide an n-degree polynomial in terms of λ:

$$a\lambda^n + b\lambda^{n-1} + \cdots + c = 0 \tag{10}$$

Solving this equation gives us a scalar of 1×n eigenvalues, and these eigenvalues can be used to produce eigenvectors. Each of the values is treated as a principal component in the following order: the first eigenvalue is the first principal component, the second eigenvalue is the second principal component, and so on. After we obtain all of the eigenvalues, we use the following equation to produce the eigenvector corresponding to each eigenvalue.

$$\left(A - \lambda I\right) x = 0 \tag{11}$$

The process is generally known as the Gaussian Elimination process [34]. The number of eigenvectors will be equal to the number of eigenvalues. Each eigenvector will contain the same number of data points as the original matrix M. Once we have all the eigenvectors, they are sorted in decreasing order. Afterward, k vectors are chosen to form an N × k dimensional matrix. This new matrix will serve as the dataset in the process of model training and testing.

Again, we use a scree plot to determine the number of principal components required for dimensionality reduction. Fig. 1 shows the percentage of variance explained by our NSL-KDD experimental dataset with respect to the number of principal components. As shown in Fig. 1, 10 principal components explain almost 95% of the variance in the original variables. Further adding principal components would not increase variance percentage much. Therefore, we choose 10 principal components. It has obtained the best results considering the least number of principal components. The dataset obtained after using PCA is linear. It is obvious that any kind of model used for classifying a linear dataset will always have satisfying results. To prove that our model is equally trustworthy in classifying a non-linear dataset, we added three independent variables from the original dataset as features. The three features have the least variance influential factor which is added as features besides the principal components. We conducted extensive experiments on adding the different number of independent variables from the original dataset besides the principal components. The experimental results help us to identify that three is the best and least number of independent variables which can be used, as adding more than three independent variables does not improve the classifier's performance

3.4. Ensemble learning

As presented in Section 2 Related Work, various machine learning approaches have been proposed to classify network intrusions. We propose B-Stacking, a new approach of stacking based on an adaptive combination of boosting and stacking algorithms. Boosting and stacking are ensemble learning methods that are normally used separately. A stacking model involves two or more base models, often referred to as level-0 models, and a meta-model that combines the predictions of the base models referred to as a level-1 model. The level-0 classifiers pass on probabilities or predicted values as features to the level-0 classifier, which then uses these values and the original observation vector to make the final classification.

Our B-Stacking algorithm chooses several simple classifiers as the level-0 weak learners. Among the weak learners, one of them should be a boosting algorithm, an ensemble meta-algorithm for primarily reducing bias and variance. For instance, in our experiments, we choose K-Nearest Neighbors, Random Forest, and XGBoost [35] as the level-0 learners. XGBoost is a booting learning algorithm. In XGBoost, weights are the sum of gradients scaled by the sum of Hessians. Choosing a boosting learner in level-0 aims to reduce the bias of the feature set on which the level-1 learner will be trained. In addition, a stacking ensemble model helps to minimize the variance in the data. Therefore, combining the stacking and boosting at level-0 will create a generalized intrusion detection system that can perform unbiased classification across all the classes. Boosting is also used as the level-1 learner in the B-Stacking algorithm with the purpose of reducing the bias of the learning model, as the level-1 boosting learner makes the final prediction. In summary, level-0 boosting helps to minimize the bias of the dataset, while level-1 learner boosting makes the architecture of the model generalized enough to fit various datasets. Therefore, combining boosting and stacking algorithms in both level-0 and level-1 brings us multiple benefits. Another advantage of the B-stacking algorithm is that the boosting leaner can be trained parallelly to reduce computation time.

Algorithm 1: B-Sacking Algorithm

```
/* Input: Sample data of N × k (N is the number of observation, and k is the number of features) */
/*Output: Trained B-stacking Classifier */

/*Training of Base Classifiers. */
Initiate a Stacked Ensemble Model S
Assign Weak classifiers L to S's lower level
Set the Boosting model B to S's top level
For i=1 to N
    train L on k features
    feed output of L to input of B
    extracts n features where $n \neq k$ using B
End For
/*Training of Meta-Classifier. */
Pass n features to meta classifier M
For j=1 to N
    train M on n features
End For
End
```

The advantage of our B-Staking classifier is that it does not require a large amount of training data and training time, as what the Neural Network requires. But the voting process of each learner gives the model the advantages of multiple decision models. The basic procedure for the proposed algorithm is summarized in Algorithm 1.

4. Evaluations

4.1. Datasets and data processing

We tested our approach on two popular and publicly available datasets, CICIDS2017 [13]. NSL-KDD [33]. CICIDS2017 is one of the most recent IDS datasets that contains benign and some common attack network flows, which meet real-world criteria and are publicly available. It is widely used in the most recent studies of cybersecurity for real-world intrusion detection. This dataset also contains network traffic analysis results obtained by using CICFlowMeter. It contains 79 features for each record, with 78 of the features refer to the network traffic and the remaining one defines whether it is a particular kind of intrusion or normal. There are 14 different intrusion detection classes present in the dataset. We represent the record count of each class in Fig. 2. As can be seen from Fig. 2, five classes including Infiltration, Web Attack Brute Force, Web Attack XSS, Web Attack SQL Injection and Heartbleed have almost no records compared to other classes of the dataset. We combined these five classes and represent them as a single class. In addition, we find that DDoS, PortScan, and DoS Hulk have significantly more records than other classes. Therefore, we apply different sampling techniques in sequential order mentioned in Section 3.2 to create a balanced dataset. After combining the five classes in CICIDS2017 we have 11 classes where 0 represents normal traffic and all the anomaly classes are represented by each number ranging from 1 to 10. We have removed some records with missing values.

Because of its popularity, we also used NSL-KDD as another dataset to evaluate our system. The NSL-KDD dataset was a revised, cleaned-up version of the KDD'99 dataset [36] from the University of New Brunswick Canadian Institute for Cybersecurity. These datasets contain the records of the Internet traffic with ghosts of the traffic encountered by real IDS. The dataset contains 43 features per record, with 41 of the features referring to the traffic input itself and the last two are labels as normal traffic or malicious traffic and Score indicating the severity of the traffic input itself. The 41 features include 7 discrete and 34 continuous variables. There are 39 different types of cyber-attacks in the training and testing dataset which can be categorized into 5 major

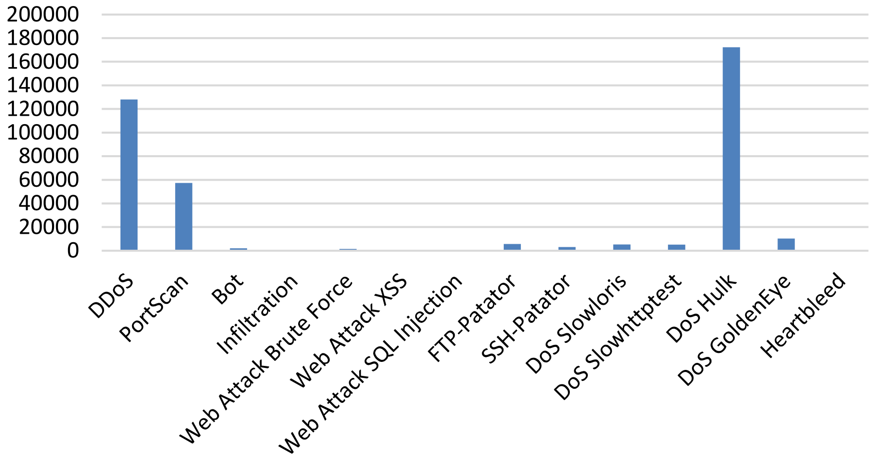


Fig. 2. Record Count of Each Class in CICIDS2017.

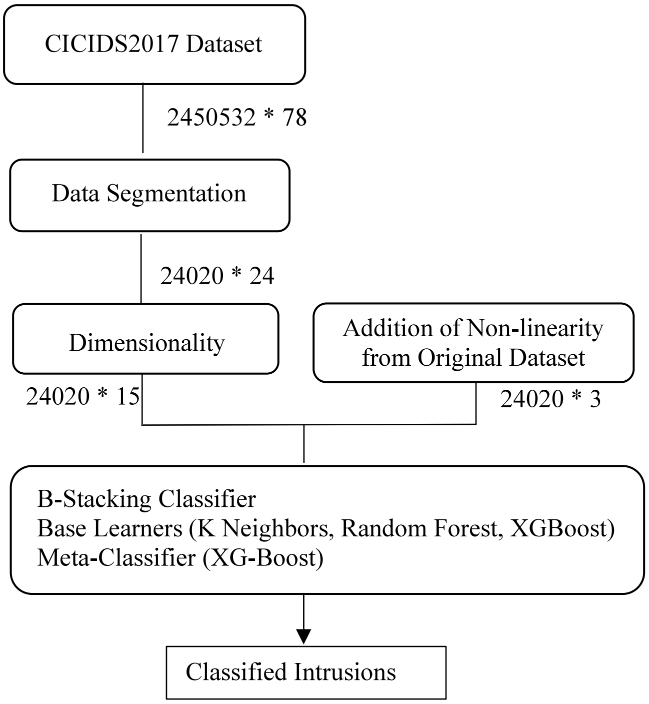


Fig. 3. Experimental Model Setup.

classes, and they are termed as Normal, Denial of Services (DoS), Root to Local (R2L), User to Root (U2R), and Probe. We classify the four major types of attack categories for the NSL-KDD dataset. We define class labels as multivariate where each class is represented by a number ranging from 0 to 4 (the value 0 represents normal traffic and other values represent an attack type).

Fig. 3 Shows the setup procedure of the experimental model. We applied the procedures presented in Section 3 on the CICIDS2017 datasets to execute our methodology. The figure also shows the size of the data after each operation. We applied the same procedure to the NSL-KDD dataset.

4.2. Performance metrics

- In order to evaluate the proposed detection mechanism, we have compared our mechanism with other states of the arts. Our performance metrics, such as Recall, Precision, Accuracy, and AUC-ROC Curve, are defined based on the confusion matrix. The confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. From the confusion matrix, we get the following derivations: *True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN).* Base on the confusion matrix, we have used the following metrics to evaluate our proposed mechanism: Accuracy, Area Under the Curve (AUC): True Positive Rate (TPR), False Positive Rate (FPR), Precision, Recall, F1-Score, and Precision–Recall Curve (PRC).

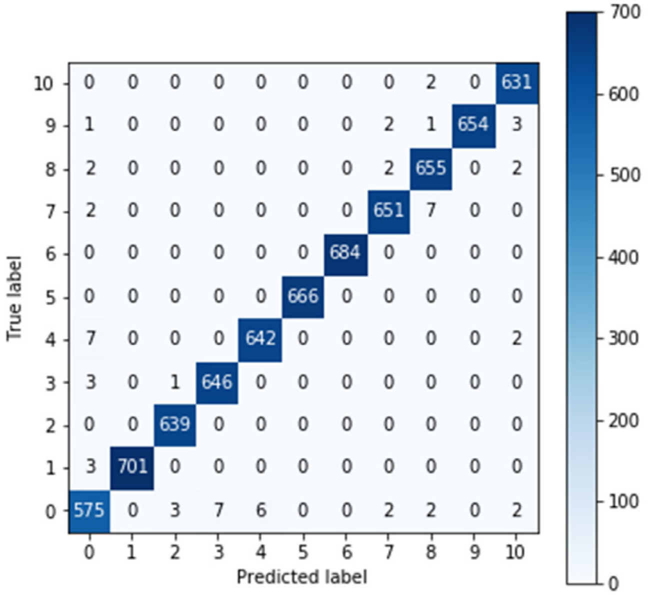
These metrics were used for binary classification and do not natively support our classification task with more than two classes. We apply the One-vs-Rest strategy [37] to use these binary classification algorithms for the multi-classification problem. The basic idea of this strategy is to split the multi-class classification dataset into multiple binary classification datasets and fit a binary classification model on each.

4.3. Result and discussion

To better understand the performance of our proposed intrusion detection mechanism, we compare its performance with the state of the arts in terms of the metrics presented in the previous section. All these systems used the same CICIDS2017 or NSL-KDD datasets.

4.3.1. Evaluation of the CICIDS2017 dataset

Fig. 4 shows the confusion matrix of our model for the CICIDS2017 dataset. We had a 7:3 split on our sample data of the size 24,020, i.e., 70% for training and 30% for testing. The diagonal elements in Fig. represent the number of instances from each class that has been correctly classified by our algorithm in this dataset. The total number of observations in our test data are 597, 704, 639, 650, 651, 666, 684, 660, 661, 661, and 633 for Benign, DDoS, PortScan, Bot, the combined class (Infiltration, Web Attack Brute Force, Web Attack XSS, Web Attack SQL Injection and Heartbleed), FTP-Patator, SSH-Patator, DoS Slowloris, DoS Slowhttptest, DoS Hulk, DoS GolenEye respectively. As shown in the confusion matrix, the false-positive rates for each of the classes in the CICIDS2017 dataset are very low which should be the main purpose of an anomaly-based IDS. Our B-Stacking algorithm only misclassified 60 samples among 7,206 in the test data. It has a very high detection rate in classifying the anomaly classes correctly.



True label	0	1	2	3	4	5	6	7	8	9	10
10	0	0	0	0	0	0	0	0	2	0	631
9	1	0	0	0	0	0	0	2	1	654	3
8	2	0	0	0	0	0	0	2	655	0	2
7	2	0	0	0	0	0	0	651	7	0	0
6	0	0	0	0	0	0	684	0	0	0	0
5	0	0	0	0	0	666	0	0	0	0	0
4	7	0	0	0	642	0	0	0	0	0	2
3	3	0	1	646	0	0	0	0	0	0	0
2	0	0	639	0	0	0	0	0	0	0	0
1	3	701	0	0	0	0	0	0	0	0	0
0	575	0	3	7	6	0	0	2	2	0	2

Fig. 4. Confusion matrix of our detection model on CICIDS2017 dataset.

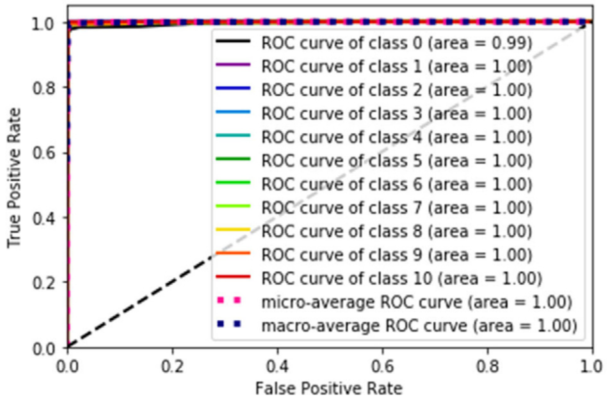


Fig. 5. ROC curve of each class on CICIDS2017 dataset.

Based on the confusion matrix, we plot the ROC curve and Precision–Recall curve using our detection model. ROC Curves summarize the trade-off between the TPR and FPR for a predictive model using different probability thresholds. While Precision–Recall curves summarize the trade-off between the TPR and the positive predictive value for a predictive model using different probability thresholds. We used the One-vs-Rest strategy to extend the precision–recall curve to our multi-class problem. Specifically, we converted the output for each class to binary and use those outputs to draw one curve for each label. The same process was executed for producing the ROC curve for each class. As illustrated in Figs. 5 and 6, both curves cover 100% or near 100% of the area while detecting the classes. We evaluated the performance of our model on each class independently. The micro-average value of the ROC and Precision–Recall curve that our model shows excellent performance.

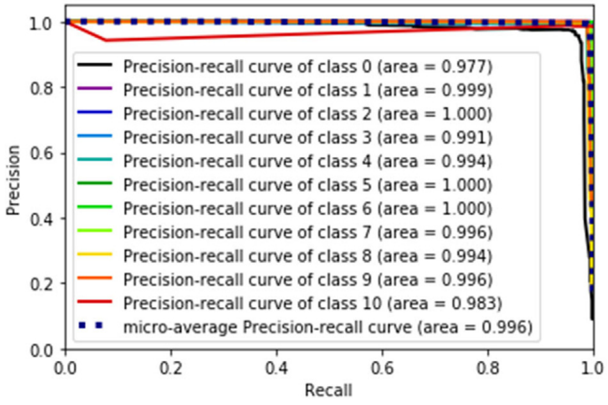


Fig. 6. Precision–recall curve of each class on CICIDS2017 dataset.

Table 1
Performance comparison on CICIDS2017 dataset.

Model name	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
DNN	95.6	96.2	95.6	95.7
GAN	99.83	98.69	92.76	95.04
A-DQN	98.7	98.6	99.4	98.9
ADABOOST	91.83	91.15	100	90.01
B-STACKING	**99.11**	**99.08**	**99.11**	**99.08**

To further understand the performance of our proposed detection model, we compare its performance with state-of-the-art techniques including DNN [14], A-DQN [19], Adaboost [20], and GAN [21]. In Table 1, we examined the performance of our B-Stacking algorithm in terms of accuracy, precision, recall, and F1 score with the four other representative intrusion detection systems that have been explained in Section 2 Related Work. As can be seen from the table that our B-stacking algorithm achieves good performance on all four metrics. It is comparable to the state-of-the-art. Moreover, as can be seen in the latter part of this section, we have compared the overhead of our B-stacking model with other algorithms, and it incurs much less overhead compared with others, such as DNN_kNN [38], which is a hybrid model comprising of a neural network and k-nearest neighbor.

We also studied the system overhead in terms of memory overhead and CPU overhead on the CICIDS2017 dataset. The computing node which has been used in performing the analysis of memory overhead and CPU overhead for the B-stacking model is an Intel®Core TM i5-9400F CPU 2.90 GHz $*$ 6 notebook with 8GB of RAM. Fig. 7 plots the memory overhead of the B-Stacking model while performing the classification task of the CICIDS2017 dataset. It can be noticed from the figure that during the 60 s evaluation time of packet flows to detect anomalies, our model used a constant memory of 3.4% of 8GB RAM. Compared with a recently proposed lightweight IDS system [38] that used a similar testing environment, they allocate 10% of the memory before their DNN–kNN algorithm started the analysis. During the analysis phase, DNN–kNN occupied around 15% of the memory which determines that around 5% of the memory is required by DNN–kNN to operate. Therefore, our system performs similarly to theirs in terms of memory overhead.

Fig. 8 shows the CPU consumption for processing the network traffics. It can be noticed that B-stacking used CPU with a range of 1.5% to 2.9%. As a comparison, the CPU usage for the DNN–kNN model [38] was around 12%. We conclude that compared to DNN–kNN, our model is extremely lightweight and can be easily used in fog nodes or IoT nodes where the computational resource is limited.

4.3.2. Evaluation of the NSL-KDD dataset

Similar experiments have been performed on the NSL-KDD dataset. Figs. 9 and 10 plot the ROC curve and Precision–Recall curve using our detection model. Again, both curves cover 100% or near 100% of the area while detecting the classes. We compared B-stacking with other state-of-the-art techniques. In Table 2, we examine the accuracy of our B-Stacking algorithm

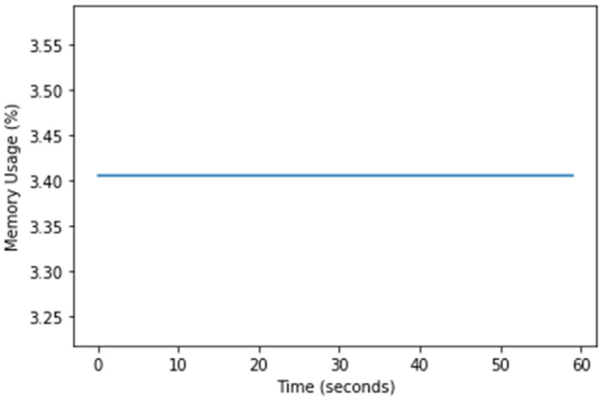


Fig. 7. Memory overhead on CICIDS2017 dataset.

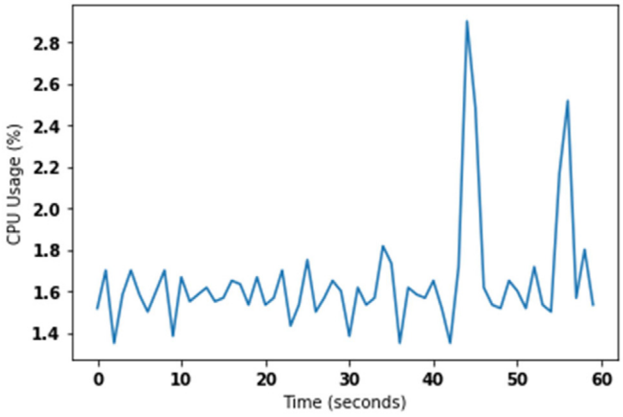


Fig. 8. CPU overhead on CICIDS2017 dataset.

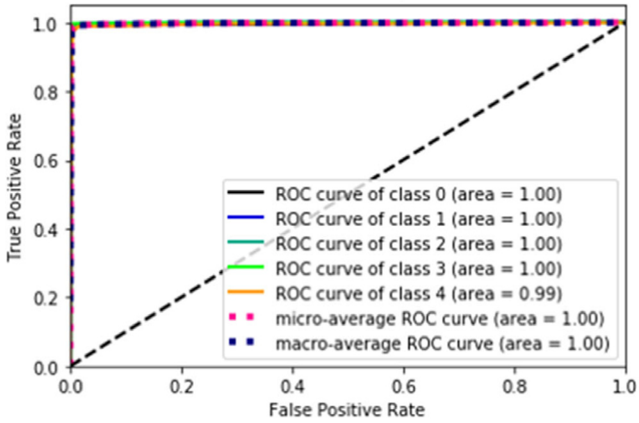


Fig. 9. ROC curve of each class on the NSL-KDD dataset.

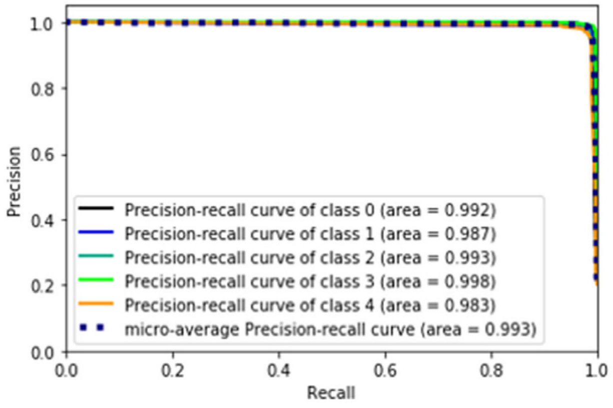


Fig. 10. Precision–recall curve of each class on the NSL-KDD dataset.

Table 2
Comparison of accuracy on the NSL-KDD dataset.

Models	Accuracy (%)
Deep Model (Diro and Chilamkurti, [7])	98.27
Shallow Model (Diro and Chilamkurti, [7])	96.75
kFN-KNN (Shapoorifard et al. [12])	99
CFS-DT (Ingre et al. [13])	90.3
DBN (Alom et al. [8])	97.5
RNN (Yin et. al., [6])	83.28
CNN–LSTM (Ahsan and Nygard, [9])	99
AE (Ieracitano et al. [10])	87
B-Stacking (Our Algorithm)	**98.5 ± 0.3**

with 8 other representative intrusion detection systems that have been explained in Section 2 Related Work. The accuracy value of our model exceeds most of these approaches, except CNN–LSTM (Ahsan and Nygard, [9]) and kFN–KNN (Shapoorifard et al. [12]). However, researchers of kFN–KNN reported accuracy alone without any other metrics in their paper. This is problematic as NSL-KDD is an imbalanced dataset, high accuracy does not mean the system can classify minority classes well. Moreover, they classified the dataset as a binary classification. Our model, instead, is a multi-class classifier that can track the specific type of attacks and allows users to take measures accordingly. Similarly, although CNN–LSTM slightly outperformed our model in accuracy, this model cannot identify the probe attack at all. It is also possible that their test data did not contain any observations falling into the Probe class. Therefore, it is not clear about the performance of the model regarding the classification of all types of attacks. This can be demonstrated by the following tests shown in Figs. 11–13.

Figs. 11–13 compare our B-Stacking model with 5 other IDS, namely AE (Ieracitano et al. [10]), CNN–LSTM (Ahsan and Nygard, [9]), RNN (Yin et al. [6]), Shallow Model, and Deep Model (Diro and Chilamkurti, [7]). As kFN–KNN (Shapoorifard et al. [12]), CFS-DT (Ingre et al. [13]), and DBN (Alom et al. [8]) only provides accuracy and we cannot find these metrics for comparison). From these figures, we can see as Deep Model and Shallow Model combined the Root to Local and User to Root class as one class and both models have not performed very well in classifying the different attack types. RNN model also performed similarly and there lies an inconsistency in detecting each of the attack types.

Comparing with these models our B-Stacking algorithm is consistent across all the classes and can classify most of the instances correctly. Our model involves a sample set of 10,904 instances which is 0.07% of the whole dataset. The use of a stacking classifier helps our meta-classifier to train on the most important features in the whole dataset. Since we used 3 base learners as the process of feature extraction, our final meta-classifier is trained on 3 features. These improvements make the B-Stacking model very lightweight and take much less amount of time in terms of training and prediction. The total time taken to train the meta-classifier after extracting features from the base-classifier is 0.28 s and the prediction time is 0.02 s. The deep learning-based approaches including CNN–LSTM, RNN, AE, Deep Model, and Shallow model are time-consuming procedures, as deep neural networks take much more computation time when a large number of records are involved.

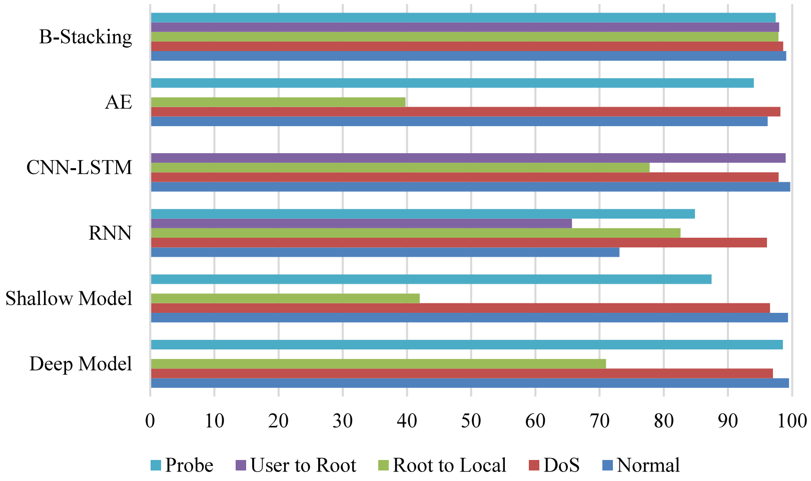


Fig. 11. Comparison of Recall on the NSL-KDD dataset.

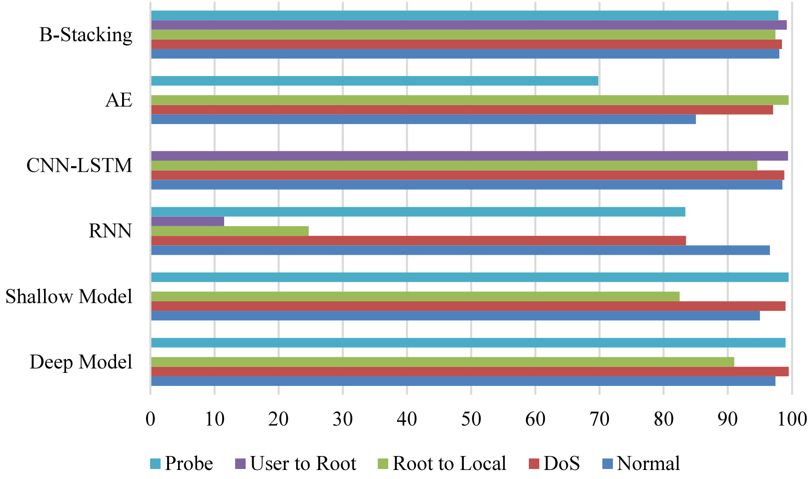


Fig. 12. Comparison of Precision on the NSL-KDD dataset.

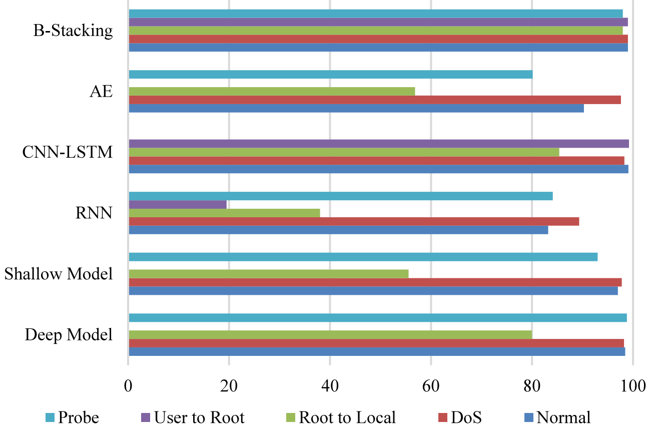


Fig. 13. Comparison of F1 on the NSL-KDD dataset.

5. Conclusion

One of the most important technological progress over the past decade was the widespread adoption of IoT devices across industries. Improving the security of IoT infrastructure is crucial to ensure the security and safety of industries and societies. In this paper, we present a novel IoT intrusion detection model B-Stacking, which uses optimized machine learning approaches to effectively detect cyber-attacks in an IoT network. We have performed extensive experiments to evaluate the performance of our system. The result shows that B-Stacking has a high detection rate and a very low false alarm rate. It overperforms most of the state of the art techniques. In the future, we plan to test our model with different IoT datasets and apply it to a real IoT network.

CRediT authorship contribution statement

Souradip Roy: Conceptualization, Methodology, Software. **Juan Li:** Supervision, Writing – original draft preparation. **Bong-Jin Choi:** Resources, Writing – review & editing. **Yan Bai:** Conceptualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This work was supported in part by the National Science Foundation (NSF), USA with award numbers: 1722913 and 1921576.

References

[1] B.B. Zarpelão, R.S. Miani, C.T. Kawakani, S.C. de Alvarenga, A survey of intrusion detection in internet of things, J. Netw. Comput. Appl. (2017) http://dx.doi.org/10.1016/j.jnca.2017.02.009.

[2] N.U. Sheikh, H. Rahman, S. Vikram, H. AlQahtani, A lightweight signature-based IDS for IoT environment, 2018, arXiv Preprint arXiv:1811.04582.

[3] R.T. Liu, C.H. Chen, C.N. Kao, N.F. Huang, A fast string-matching algorithm for network processor-based intrusion detection system, ACM Trans. Embed. Comput. Syst (2004) http://dx.doi.org/10.1145/1015047.1015055.

[4] M. Rebbah, D.E.H. Rebbah, O. Smail, Intrusion detection in Cloud Internet of Things environment. In: 2017 International Conference on Mathematics and Information Technology (ICMIT), 2017, pp. 65–70.

[5] H. Larijani, J. Ahmad, N. Mtetwa, et al., A novel random neural network based approach for intrusion detection systems. In: 2018 10th Computer Science and Electronic Engineering (CEEC), 2018. pp. 50–55.

[6] C. Yin, Y. Zhu, J. Fei, X. He, A deep learning approach for intrusion detection using recurrent neural networks, IEEE Access (2017) http://dx.doi.org/10.1109/ACCESS.2017.2762418.

[7] A.A. Diro, N. Chilamkurti, Distributed attack detection scheme using deep learning approach for Internet of Things, Futur. Gener. Comput. Syst (2018) http://dx.doi.org/10.1016/j.future.2017.08.043.

[8] M.Z. Alom, V. Bontupalli, T.M. Taha, Intrusion detection using deep belief networks, 2016, http://dx.doi.org/10.1109/NAECON.2015.7443094.

[9] M. Ahsan, K. Nygard, Convolutional neural networks with LSTM for intrusion detection, Proc. 35th Int. Confer 69 (2020) 69–79.

[10] C. Ieracitano, et al., Statistical Analysis Driven Optimized Deep Learning System for Intrusion Detection, 2018, http://dx.doi.org/10.1007/978-3-030-00563-4_74.

[11] C. Song, W. Fan, S.Y. Chang, Y. Park, Reconstructing Classification to Enhance Machine-Learning Based Network Intrusion Detection by Embracing Ambiguity, 2021, http://dx.doi.org/10.1007/978-3-030-72725-3_13.

[12] I.A. Khan, D. Pi, Z.U. Khan, Y. Hussain, A. Nawaz, Hml-ids: A hybrid-multilevel anomaly prediction approach for intrusion detection in scada systems, IEEE Access (2019) http://dx.doi.org/10.1109/ACCESS.2019.2925838.

[13] A. Boukhamla, J.C. Gaviro, CICIDS2017 dataset: Performance improvements and validation as a robust intrusion detection system testbed, Int. J. Inf. Comput. Secur (2018).

[14] R. Vinayakumar, M. Alazab, K.P. Soman, P. Poornachandran, A. Al-Nemrat, S. Venkatraman, Deep learning approach for intelligent intrusion detection system, IEEE Access (2019) http://dx.doi.org/10.1109/ACCESS.2019.2895334.

[15] W. Li, P. Yi, Y. Wu, L. Pan, J. Li, A new intrusion detection system based on KNN classification algorithm in wireless sensor network, J. Electr. Comput. Eng. (2014) http://dx.doi.org/10.1155/2014/240217.

[16] N. Moustafa, J. Slay, UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set), 2015, http://dx.doi.org/10.1109/MilCIS.2015.7348942.

[17] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, K. Nakao, Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation, 2011, http://dx.doi.org/10.1145/1978672.1978676.

[18] I. Almomani, B. Al-Kasasbeh, M. Al-Akhras, WSN-DS: A dataset for intrusion detection systems in wireless sensor networks, J. Sensors (2016) http://dx.doi.org/10.1155/2016/4731953.

[19] K. Sethi, Y.V. Madhav, R. Kumar, P. Bera, Attention based multi-agent intrusion detection systems using reinforcement learning, J. Inf. Secur. Appl (2021) http://dx.doi.org/10.1016/j.jisa.2021.102923.

[20] A. Yulianto, P. Sukarno, N.A. Suwastika, Improving adaboost-based intrusion detection system (IDS) performance on CIC IDS 2017 dataset. In: Journal of Physics: Conference Series, vol. 1192, no. 1, 2019, pp. 12018.

[21] J.H. Lee, K.H. Park, GAN-Based imbalanced data intrusion detection system, Pers. Ubiquitous Comput (2021) http://dx.doi.org/10.1007/s00779-019-01332-y.

[22] H. Shapoorifard, P. Shamsinejad, Intrusion detection using a novel hybrid method incorporating an improved KNN, Int. J. Comput. Appl (2017) http://dx.doi.org/10.5120/ijca2017914340.

[23] B. Ingre, A. Yadav, A.K. Soni, Decision tree based intrusion detection system for NSL-KDD dataset. In: International Conference on Information and Communication Technology for Intelligent Systems, 2017, 207–218.

[24] N. Farnaaz, M.A. Jabbar, Random Forest Modeling for Network Intrusion Detection System, 2016, http://dx.doi.org/10.1016/j.procs.2016.06.047.

[25] B. Sasha, A strict anomaly detection model for IDS, Phrack Mag. 0xa Issue 0x38, May1, vol. 138, 2000.

[26] K.P. Vatcheva, M. Lee, Multicollinearity in regression analyses conducted in epidemiologic studies, Epidemiol. Open Access (2016) http://dx.doi.org/10.4172/2161-1165.1000227.

[27] R.M. O'Brien, A caution regarding rules of thumb for variance inflation factors, Qual. Quant (2007) http://dx.doi.org/10.1007/s11135-006-9018-6.

[28] J.M. Andrade-Garda, A. Carlosena-Zubieta, R. Boqué-Marti, J. Ferré-Baldrich, Partial least-squares regression. In: RSC Analytical Spectroscopy Series, 2013.

[29] H. Abdi, L.J. Williams, Principal component analysis, in: Wiley Interdisciplinary Reviews: Computational Statistics, 2010, http://dx.doi.org/10.1002/wics.101.

[30] M. Zhu, A. Ghodsi, Automatic dimensionality selection from the scree plot via the use of profile likelihood, Comput. Stat. Data Anal (2006) http://dx.doi.org/10.1016/j.csda.2005.09.010.

[31] D.A. Cieslak, N.V. Chawla, A. Striegel, Combating imbalance in network intrusion datasets, 2006, http://dx.doi.org/10.1109/grc.2006.1635905.

[32] R. Blagus, L. Lusa, SMOTE For high-dimensional class-imbalanced data, BMC Bioinformatics (2013) http://dx.doi.org/10.1186/1471-2105-14-106.

[33] E. AT, A. M, A.-M. F, S. M, Classification of imbalance data using tomek link (T-Link) combined with random under-sampling (RUS) as a data reduction method, Glob. J. Technol. Optim (2016) http://dx.doi.org/10.4172/2229-8711.s1111.

[34] J.F. Grcar, How ordinary elimination became Gaussian elimination, Hist. Math (2011) http://dx.doi.org/10.1016/j.hm.2010.06.003.

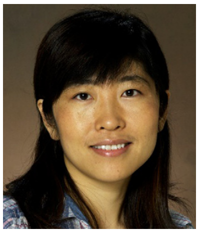
[35] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, 2016, http://dx.doi.org/10.1145/2939672.2939785.

[36] M. Tavallaee, E. Bagheri, W. Lu, A.A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, 2009, http://dx.doi.org/10.1109/CISDA.2009.5356528.

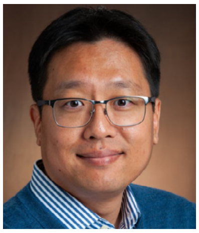
[37] Pattern recognition and machine learning, J. Electron. Imaging (2007) http://dx.doi.org/10.1117/1.2819119.

[38] C.A. de Souza, C.B. Westphall, R.B. Machado, J.B.M. Sobral, G. dos S. Vieira, Hybrid approach to intrusion detection in fog-based IoT environments, Comput. Networks (2020) http://dx.doi.org/10.1016/j.comnet.2020.107417.

Souradip Roy received the B.Tech degree in Computer Science and Engineering from Maulana Abul Kalam Azad University of Technology, Kolkata, India in 2015. He completed his M.S degree from the North Dakota State University in Computer Science in 2019. He is currently pursuing the Ph.D. degree in Machine Learning and Deep Learning at the North Dakota State University, Fargo, USA.



Juan Li is a professor at the Computer Science Department of North Dakota State University. Dr. Li received her Ph.D. in Computer Science from the University of British Columbia, her M.S. degree in Computer Science from the Chinese Academy of Sciences, and her BS in Computer Science from the Beijing Jiaotong University. Dr. Li's major research interest lies in distributed systems, intelligent systems, social networking, and semantic web technologies. Dr. Li has published more than 100 peer-reviewed articles and has served as a reviewer for many different journal and conferences.



Bong-Jin Choi is an assistant professor at the Statistics and Public Health Department of North Dakota State University. Dr. Choi received his B.S. Degree in Computer Science from Kyungnam University, South Korea and received Master and Ph.D. degree in Statistics from University of South Florida, Tampa, FL. After earning Ph.D., Dr. Choi joined the UNC Comprehensive Cancer Center, University of North Carolina, Chapel Hill as Big Health Care Data Manager. His current research interests include Statistical Machine Learning and Big Health Care Data Analysis. Dr. Choi is serving Managing Editor in Global Journal of Mathematical Sciences and Director of Journals in International Federation of Nonlinear Analysts.

Yan Bai is a Professor in the School of Engineering and Technology, University of Washington Tacoma, USA. Dr. Bai received her Ph.D. in Electrical and Computer Engineering from the University of British Columbia, Vancouver, BC, Canada. Her research interests include computer networking, multimedia communications, cybersecurity and privacy, eHealth, Internet of Things, blockchain, cloud, and edge computing. She has published over 80 refereed papers in these areas. She has served as a General Chair/Program Chair/ Technical Program Committee Member for numerous IEEE conferences and workshops, and as a Reviewer for a wide range of high impact research journals and ACM/IEEE flagship conferences.