IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 24, 2020, accepted April 16, 2020, date of publication April 20, 2020, date of current version May 4, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2988854

Fog-Based Attack Detection Framework for Internet of Things Using Deep Learning

AHMED SAMY[1,2], HAINING YU[1], AND HONGLI ZHANG[1], (Member, IEEE)
[1]School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
[2]Faculty of Computers and Information, Menoufia University, Shebeen El-Kom 32511, Egypt

Corresponding author: Ahmed Samy (ahmed.samy@hit.edu.cn)

This work was supported by the Natural Science Foundation of China under Grant 61732022 and Grant 61601146.

ABSTRACT The number of cyber-attacks and data breaches has immensely increased across different enterprises, companies, and industries as a result of the exploitation of the weaknesses in securing Internet of Things (IoT) devices. The increasing number of various devices connected to IoT and their different protocols has led to growing volume of zero-day attacks. Deep learning (DL) has demonstrated its superiority in big data fields and cyber-security. Recently, DL has been used in cyber-attacks detection because of its capability of extracting and learning deep features of known attacks and detecting unknown attacks without the need for manual feature engineering. However, DL cannot be implemented on IoT devices with limited resources because it requires extensive computation, strong power and storage capabilities. This paper presents a comprehensive attack detection framework of a distributed, robust, and high detection rate to detect several IoT cyber-attacks using DL. The proposed framework implements an attack detector on fog nodes because of its distributed nature, high computational capacity and proximity to edge devices. Six DL models are compared to identify the DL model with the best performance. All DL models are evaluated using five different datasets, each of which involves various attacks. Experiments show that the long short-term memory model outperforms the five other DL models. The proposed framework is effective in terms of response time and detection accuracy and can detect several types of cyber-attacks with 99.97% detection rate and 99.96% detection accuracy in binary classification and 99.65% detection accuracy in multi-class classification.

INDEX TERMS Attack detection, cybersecurity, deep learning, fog computing, long short term memory, Internet of Things.

I. INTRODUCTION

The Internet of Things (IoT) is considered a rapidly developing paradigm in the history of computing. In the past few years, IoT has immensely evolved in different technological fields. It has converged between hundreds of billions of devices from different systems (such as smart vehicles, smart health care, smart grid, smart home, etc.) and the internet [1]. However, this convergence has resulted in many cyber-attacks on IoT systems because IoT integrates the digital world with the physical environment [2]. IoT security has become challenging because of the heterogeneity, large scale, limited hardware resources, and global accessibility of IoT systems. Researchers have used machine learning (ML) algorithms such as decision tree (DT), random forest (RF), support vector machine (SVM), Bayesian network, and K-Means to detect network attacks, as proposed in [3], [4]. However, the process requires manual feature engineering [5]. Obtained features such as number of bytes sent and received, connection time, number of requests, and error count cannot deeply represent the pattern behavior of attacks. Thus, ML is unsuitable for detecting cyberattacks and serving as a practical solution in the industry. The best solution to overcome the limitations of ML is to use deep learning (DL). DL can represent data using the multiple processing layers of computational models. In addition, DL can provide a deep representation of raw data and predict or classify data more accurately than ML because of its multilayer structure [6]. However, the direct implementation of complex DL models on IoT devices is challenging because of the limited computation, storage, and energy capabilities of IoT devices. Therefore, using DL for attack detection in IoT is not a direct way.

DL has been used in many proposed intrusion detection systems (IDSs) for IoT in [7]–[10], and [11]. DL has a high detection rate (DR) in recognizing morphing attacks. On the one hand, DL is a powerful tool used to analyze huge traffic volumes and accurately distinguish the normal and abnormal behavior of different systems by extracting

The associate editor coordinating the review of this manuscript and approving it for publication was Chun-Wei Tsai.

deep complex patterns from raw data (packets). On the other hand, the direct implementation of complex DL models on IoT devices is inappropriate due to the constrained nature of the IoT devices. Many researchers have used DL to detect cyber-attacks in IoT [5], [12]. However, no one has explained the implementation of these computational and power-intensive models on low capacity sensors. Thanks to the development of fog computing that can aid the direct implementation of DL models in IoT devices by processing, analyzing, and storing large volumes of data on fog nodes with low latency and high response time [13]. The idea is to move the implementation of DL from sensors in the edge layer to the nearest place of data sources, at which point data analysis takes little time. Fog computing extends traditional cloud-based services so that they are at the network edge where data are generated. Furthermore, it provides a distributed environment, mobility, and scalability [14]. Accordingly, DL can be implemented on the fog layer nodes where fog computing allows the implementation and execution of attack detection in a distributed, powerful and scalable manner.

In the current work, we present a detailed framework of a distributed and robust attack detection for IoT that is based on fog computing and DL. The proposed fog-based attack detection framework takes a very short time to detect attacks and smaller response time than cloud-based attack detection. We evaluate six supervised DL models in our experiments and select the best one. We use four new datasets and one old dataset to evaluate the performance of the DL models through extensive experiments. The five datasets involving different attacks, such as mirai, DDoS, worms, exploits, sybil, sinkhole, prob, R2L, etc to verify the capability of the proposed framework in detecting several attacks. Based on the results of extensive experiments, the LSTM model achieves the best performance in attack detection and accuracy. IoT traffic generated in the edge layer is routed to the fog layer via smart gateway devices. The detection system implemented on the fog nodes classifies IoT data to detect attacks. The detection system is controlled using a cloud service to confirm its distributivity, scalability, and rapidity. The proposed framework consists of four stages. The first stage aims to train the DL model and tune the hyperparameters to achieve the best performance. The second stage discusses how to implement the proposed attack detection framework on fog nodes and how they communicate and controlled by cloud service. The third stage discusses the attack detection based on traffic analysis using the LSTM model. The fourth stage shows how to monitor, evaluate, and update the LSTM model. The proposed framework is explained in detail in Section IV. The main contributions of this research are summarized as follows

1) Presentation of detailed framework of a robust and distributed attack detection for IoT networks.
2) Comparison of six supervised DL models to select the best model for IoT attack detection based on extensive experiments.
3) Evaluation of the performance of six DL models using four up-to-date IDS datasets and one old dataset.
4) Achievement of the highest DR and lowest false alarm rate (FAR) in comparison with three other IDSs.
5) Offer of proof that DL has better detection capability than ML and that it can detect several types of attacks in IoT through extensive experiments.

The remaining parts of the paper are organized as follows. Section II reviews the related work that used DL in attack detection. Section III discusses the architecture and rules of fog computing in IoT. Section IV shows the stages of the proposed framework in detail. Section V provides an overview of DL models used in the experiments and the details of the five datasets. Section VI explains the experiments and evaluation of results. Section VII presents the conclusion and future work.

II. RELATED WORK

This section discusses the state-of-the-art security approaches that use DL for attack detection in IoT. Furthermore, the capability of extracting latent features and detecting different attacks of different DL models under different network environments is investigated.

Cyber-attacks have immensely increased in the last 10 years with the rapid growth of IoT devices and applications [15]. Attackers have utilized cyber-attacks to compromise thousands of IoT devices that are globally accessible and unsecured. For instance, in 2016, several websites using DNS provider "Dyn" were attacked using a DDoS attack. This attack involved several IoT devices to execute the botnet malware [15]. Researchers have proposed many security approaches and frameworks to mitigate specific cyber-attacks and internal attacks. However, these security approaches are rapidly compromised by new attacks [16]. Thus, IoT requires a distributed security solution that can constantly monitor IoT devices, detect zero-day attacks, and make sound decisions. Many famous organizations such as Facebook, Yahoo!, Twitter, and YouTube, have developed many applications on the basis of DL to monitor and analyze huge volumes of data generated from billions of users [17]. DL has been widely used with the recent improvement of graphical processing units (GPUs), the availability of big data used to train DL Models, and the existence of powerful learning algorithms. To conclude, DL has demonstrated its superiority in big data analysis, and extensive research has focused on IoT attack detection using DL [18].

An IDS for In-vehicle network security based on deep neural networks (DNNs) was proposed in [7]. They used a pre-trained unsupervised deep belief network model to initialize the parameters of the DL network and extract the features from in-vehicular network packets. These packets were generated by simulating In-vehicle network communication. The deep belief network model, combined with a conventional stochastic gradient descent method, was used for classification. The proposed IDS can respond to real-time

TABLE 1. Comparison between different Intrusion detection approaches.

Ref	DL Model Name	Attacks Detected	Datasets Used	Highest Accuracy	Implementation Layer	Strategy
[7]	DBN	CAN network attacks	packets generated using OCTANE	98%	edge layer	centralized
[8]	DBN	eavesdropping and jamming attacks	10 datasets	outperforms other ML models by 6%	mobile edge computing	centralized
[11]	CNN	android malware	three datasets	89.7%	mobile devices	centralized
[19]	RNN	botnet attack	two datasets	96.8%	desktop computers	centralized
[5]	LSTM	vulnerabilities of wireless communications attacks	two datasets	99.91%	fog layer	distributed
[12]	AEs	network malware	two datasets	83.3%	desktop computers	centralized
[21]	ELM	DoS, botnets	one dataset	99%	cloud and edge layers	distributed
[22]	CNN and LSTM	network malware	one dataset	99.83%	desktop with GPU	centralized

attacks with 98% detection accuracy on average, and the DL model outperforms traditional ML models. Another DL-based security model was developed to detect malicious applications at the edge of a cellular network using mobile edge computing [8]. The proposed model consisted of two components, namely, feature preprocessing and malicious application detection engines. For the malicious application detection engine, a deep belief network was used for unsupervised feature learning, and a softmax function was utilized for prediction. The proposed model was implemented on 10 different datasets and the result showed that its detection accuracy was higher than that of softmax regression, SVM, DT and RF. Another study detected Android malware by automatically extracting convolutional neural features from raw opcode sequences [11]. After the conversion of a sequence of opcode instructions into one-hot vectors, it was fed to the embedding layer. The output from the embedding layer was fed to one or more convolutional layers to extract feature vectors. The feature vectors were passed to a multilayer perceptron (MLP) network for classification. The proposed model achieved higher detection performance with small datasets than other state-of-the-art models. Whereas, it achieved a lower detection performance with large datasets.

Reference [19] used a recurrent neural network (RNN) DL model to detect botnet behavior. The authors discussed the ability of the RNN to evaluate network traffic behavior by detecting botnet attacks by using LSTM. The behavior of connections between devices was used by the LSTM detection model to detect botnets. Two different datasets were used for training and testing the detection model, and another unseen dataset was utilized to evaluate the performance of LSTM with different connection states. LSTM was capable of detecting different botnet behavior. Another research [5], [20] conducted attack detection on fog-to-things using an LSTM-based deep network. The authors introduced a distributed approach using fog nodes, each of which received initialization and update parameters from the coordinating node. Each fog node trained the LSTM model on the basis of the parameters received from the coordinating node then sent the weights and bias values back to the coordinating node. Subsequently, the aggregated parameters were calculated and returned to the fog nodes. The proposed model proved that DL models outperform traditional ML models. The authors demonstrated that distributed attack detection is more efficient and scalable than a centralized approach. Deep autoencoders, which is unsupervised DL model that has been used by some researchers to build feature learning for detection models. The authors in [12] used autoencoders for unsupervised feature learning to detect network-based malware. Autoencoders were fed by the features acquired from cybersecurity phenomena. Latent features generated from autoencoders enhanced the DR of different classifiers, such as Gaussian Naive Bayes, SVM, and Xgboost. Some studies have combined different DL models to provide an ensemble learning method for learning and detection enhancement. A distributed attack detection scheme has been proposed in [21]. It used extreme learning machine (ELM) classifier to classify network traffic at the edge computing layer. Furthermore, it moved all extensive resources operations such as model training and construction to the cloud layer using HPC cluster. The proposed scheme carried out model training used anonymized data collected from edge layer devices. Then, the training model used by classifiers on edge servers. The experiments demonstrated that the proposed scheme achieved high accuracy in scanning, communication, and infected hosts scenarios with 99%, 74%, and 95% respectively. A malware detection method using CNN and LSTM was presented in [22]. The proposed method converted the opcode sequence of a malware file into grayscale images, and CNN and LSTM learned from these images. This method outperformed other ML methods, such as SVM, RF and, linear k-nearest neighbor (KNN). All mentioned related works are summarized in Table 1.

III. FOG COMPUTING ARCHITECTURE AND ITS ROLE IN IoT

In typical IoT architecture, smart gateway devices are used to route data from the edge layer to upper layers (e.g., fog and cloud). Data analysis requires considerable time and experiences some delays because the cloud is so far from the

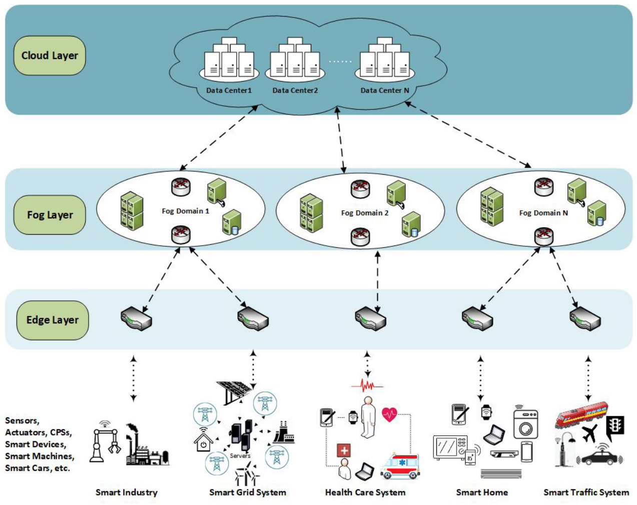
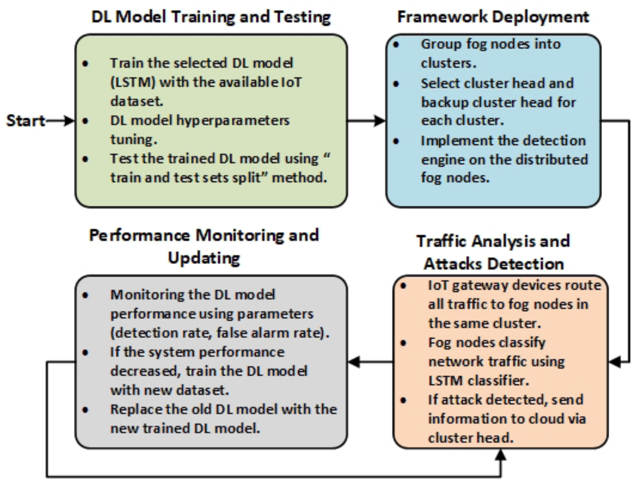


FIGURE 1. Fog computing architecture for IoT systems.


Start
DL Model Training and Testing
Framework Deployment
Performance Monitoring and Updating
Traffic Analysis and Attacks Detection

DL Model Training and Testing

- Train the selected DL model (LSTM) with the available IoT dataset.
- DL model hyperparameters tuning.
- Test the trained DL model using " train and test sets split" method.

Framework Deployment

- Group fog nodes into clusters.
- Select cluster head and backup cluster head for each cluster.
- Implement the detection engine on the distributed fog nodes.

Traffic Analysis and Attacks Detection

- IoT gateway devices route all traffic to fog nodes in the same cluster.
- Fog nodes classify network traffic using LSTM classifier.
- If attack detected, send information to cloud via cluster head.

Performance Monitoring and Updating

- Monitoring the DL model performance using parameters (detection rate, false alarm rate).
- If the system performance decreased, train the DL model with new dataset.
- Replace the old DL model with the new trained DL model.

FIGURE 2. Proposed framework stages for attack detection in IoT networks.

edge layer [23]. Thus, this process is unsuitable for sensitive data that require a fast response. Fog computing is introduced to tackle these challenges by extending the cloud to be close to the data sources. Fog computing was proposed by CISCO in 2012 to solve the challenges of cloud computing [24]. Fig.1 shows the architecture of fog computing in IoT systems, which is divided into three layers, namely, edge, fog, and cloud layers. The edge layer combines billions of heterogeneous IoT devices such as sensors, vehicles, security cameras, smart wearable devices, smart machines and smart home appliances. This layer generates large data size from different places and applications. The fog layer is the intermediate layer between the edge and cloud layer. The fog layer is divided into many connected domains, and each domain contains virtualized fog servers, routers, simple data centers, applications, and services. The upper layer is the cloud layer, which is the core layer comprising high-performance servers and storage devices. The deployment of DL on fog nodes is useful for IoT, in several ways, including the following: sensitive data analysis close to IoT devices that generate data, the latency between data sources and data analysis devices can be reduced, network bandwidth can be minimized, the data to be sent to the cloud and the data to be processed on fog nodes can be processed, and mobility services can be supported [25]. Authors in [26], proposed a fog vehicle computing (FVC), which is a fog model that uses the available unused resources of vehicles to create temporary fog computing resources for data processing. The proposed model used a pool of parking vehicles at a shopping mall for computing power. The authors explained the allocation of appropriate computation and storage resources through a policy management layer. Moreover, a decision-making process was introduced to perform the required services on the available resources. Cisco, IDC FutureScape state that 40% of the data generated by IoT devices are analyzed on devices near the IoT devices [27]. Fog nodes have been widely used in several technologies, such as 5G and many fog-to-things applications, because of the increasing computational and communication capabilities of their hardware [28].

Today, security approaches can be implemented near the edge layer using distributed fog nodes to analyze network traffic and rapidly detect attacks. In the current work, we present a DL-based distributed attack detection framework for IoT with low latency, geographical distributability, scalability, and high-speed response, by using the advantages of fog computing.

IV. PROPOSED ATTACK DETECTION FRAMEWORK

Gartner, Inc. expected that 5.8 billion IoT devices will be in use by 2020 [29]. These numerous IoT devices located at different geographical areas generate massive amounts of data that require rapid analysis to detect attacks. Thus, centralized attack detection unsuitable for IoT security monitoring. The proposed framework based on LSTM DL model implemented on distributed fog nodes, and is controlled and updated via the service located in the cloud computing layer. The proposed framework consists of four main stages, namely, DL model training and testing, framework deployment, traffic analysis and attack detection, and performance monitoring and updating. The stages of the proposed framework are shown in Fig. 2.

1) DL MODEL TRAINING AND TESTING

The selection of an appropriate DL model plays a key role in the proposed framework's detection accuracy and efficiency. The DL model should be as good as the data used to train it. This stage aims to determine the best DL model and train it with IoT data in cloud layer to enhance the performance of the selected DL model in detecting various attacks. To determine the best DL model that fulfills the requirements of the proposed framework, we conduct several experiments using six different supervised DL models with five different datasets, as discussed in details in section V. The LSTM DL model achieves the highest DR as well as small FAR. As we mentioned before, all IoT packets exchanged between IoT devices or routed to upper layers (fog layer and cloud layer) go through smart IoT gateway. A tcpdump is a network sniffer and packet analyzer tool used to capture packets that received or sent over the network in the edge layer. The tcpdump tool run on the smart IoT gateway to collect raw network traffic (packets) from IoT network in Pcap file format. A network traffic flow analyzer tool called ''CICFLOWMETER'' used to convert the raw network traffic to CSV format with more than 80 network traffic features [30]. CICFLOWMETER offers more flexibility in terms of choosing the features you want to calculate.

This stage is composed of several steps, the first of which involves training the LSTM model with the available IoT datasets in the CSV file format that collected from the previous step. The training of LSTM model is carried out in the cloud layer to make it goes faster. We use Amazon Elastic Compute Cloud (Amazon EC2) virtual server instance on which we can run Keras for learning and testing LSTM model. Amazon built the DL Amazon Machine Image (AMI) AmazonLinux-2.0 for DL on EC2 instances with popular DL framework and also contains the anaconda platform. Amazon Web Services DL AMI are built to build, train and debug DL models in EC2 with popular frameworks such as Keras, TensorFlow, PyTorch and more. We can copy the CSV file collected at the edge layer by smart IoT gateways to the AWS instance using scp command by using the keypair and the Ip address of the AWS EC2 instance as follows : **scp -i keras-aws-keypair.pem -r src ec2-user@54.180.78.7:~/**

Then, we run the LSTM model for model training and testing. We use the sigmoid activation function in binary class classification and softmax in multi-class classification. The LSTM network used in the proposed framework has one input layer with 128 cells and one output layer with (m) cells based on the number of classes and has three hidden layers with (256) cells each. We use an adaptive learning rate method called ''Adam'' as an optimizer that computes individual learning rates for different parameters and achieves good results fast. It combines the advantages of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Then, the hyperparameters, which affect the performance of the DL model, are tuned. Hyperparameters values such as epoch number, learning rate, and patch size are set before start model training. We use many values for the DL model's hyperparameters to achieve the best performance. After training, the performance of the trained LSTM model is evaluated on unseen data based on evaluation metrics discussed in section VI. Thus, we use different training and testing datasets using train and test split method. We split the datasets into two parts, namely, training and testing parts. The size of the training dataset is 70% of the entire dataset size, and the remaining 30% is used for testing. If the evaluation result is not good, the hyperparameters are tuned, or the LSTM model becomes deeper until the best performance is achieved. Our experiments conclude that, LSTM has the highest performance that outperform all other DL model in terms of DR, FAR, accuracy, recall, F1-Measure and precision.

2) FRAMEWORK DEPLOYMENT

The second stage of the proposed framework involves implementing the framework in fog nodes. Fig. 3, shows the architecture of the proposed framework, which comprises the cloud, fog, and edge layers. The edge layer contains billions of IoT devices with limited resources, such as sensors, actuators and security cameras. These devices generate huge volumes of unstructured data that are difficult to analyze it at the edge layer. The edge layer includes smart homes, oil rigs, smart power grids, smart cars, and planes. All data generated from edge layer devices are routed to the fog layer, enterprise data center and cloud via a smart IoT gateway. The second layer is the fog layer that contains thousands of servers, routers, and controllers owned by an Internet service provider. These devices are more powerful than edge devices. Fog devices can run processes that require high memory, computational power, storage, and energy. Furthermore, fog nodes are distributed at different geographical areas that exist at service provider (SP) networks and near to the edge layer than to the cloud layer. It has multiple interfaces and services to communicate with different protocols and applications. Distributed data analytics at the fog layer allows data processing before transmitting them into the cloud. Moreover, it minimizes bandwidth, reduces latency, and rapidly response to critical actions that make the system resilient. The top layer is the cloud layer that delivers computing services over the internet and provides flexible, reliable, and scalable resources to cloud users. Cloud computing allows data transfer, storage, and analysis through the internet. IoT experiences high latency during data transfer or data analysis in the cloud, especially on real-time applications, such as smart vehicles.

To implement the proposed framework in IoT networks, we assume that a clustering algorithm such as in [31] is used to group fog nodes into clusters as shown in Fig. 3, the fog layer is divided into N clusters. This clustering algorithm designed for the distributed discovery of clusters of wireless nodes based on physical network topology features. This clustering algorithm work without need any information about the expected number of clusters and it assumes a zero or low mobility for participating nodes that make it appropriate for our framework. It identifies clusters based on some parameters such as the density of the network graph, the preferential attachment, and the interactions among nodes. Clustering fog nodes applied to balance network load, increase network scalability and secure the exchanged traffic between clusters and cloud. One cluster can handle, process and analyze data from different IoT edge networks; for instance, cluster1 in Fig. 3 analyzes data from a wireless sensor network and smart home network. Every cluster has cluster members, one Cluster Head (CH) and one backup cluster head (BCH) that elected by cluster members.

CHs are responsible for process incoming and outgoing traffic that are generated to or from their cluster members. Moreover, CHs are the links between the cloud service and clusters members. All the attack detection updates from cloud service propagated to cluster members via CHs as shown in Fig. 3 by black arrows come from the cloud layer to CH and from CH to cluster members. All data exchanged between CHs and cloud service encrypted using Triple Data Encryption Standard (3DES). We assume that the paths between smart IoT gateway devices and fog nodes are secured which encrypt data before transmission. In case of CH failure, BCH becomes the CH and a cluster member becomes BCH. Smart gateways work as sink nodes, they collect the traffic exchanged in the network and forward it to the nearest fog node. Fog nodes receive network traffic forwarded from several IoT smart gateways and store it in different files. A service in the backgroup on fog nodes work to read and process data from the files. The data processing is extracting the features of each packet in the network traffic, then feed them to the LSTM classifier to detect attacks.

3) TRAFFIC ANALYSIS AND ATTACK DETECTION

After implementing the attack detection on the fog nodes in all clusters, it starts to receive network traffic routed from IoT smart gateway devices from different networks. The proposed attack detection handles raw network data, classifies traffic into two classes (normal or attack), and recognizes the type of attack on the basis of the dataset used in the training stage. Every cluster member saves the ID of its head node, and the CH knows the number of cluster members that form its cluster and their IDs. In the case of attack detection at any fog cluster member, by using the ID of the head node, complete information about the detected attack (attack type, attack source, protocol, duration, etc…) is propagated from cluster members to the CH that work as information aggregator, then it is propagated from the CH to the cloud service as shown in figure 3 by red arrows. The information goes from cluster members to CHs and then to the cloud service. The cloud service raises an alarm and logs all the information from CHs to be used by a network administrator in evaluating and updating the performance of the attack detection and taking the appropriate decisions.

4) PERFORMANCE MONITORING AND UPDATING

The DL model ability to detect attacks diminished over time. So, the performance of the attack detection is tested and updated to verify the capability of the detection engine or the DL model. We measured the capability of the DL model on the basis of two parameters, namely, the DR and FAR. At different times in the month, new data is collected from the edge layer devices and uploaded to the Amazon EC2 server after converting it to CSV file format using CICFLOWMETER tool as mentioned in section IV-.1. The collected data may have a huge volume of data, we can select samples of the collected data to test the performance of the running DL model. The evaluation metrics such as DR, and FAR are calculated in the testing process using samples of the newly collected data. We compare the evaluation metrics values in the testing process with values in the training phase (phase 1). The evaluation metrics show whether the performance of the detection model degrades, improves or becomes stable. If the DR value in the testing process (DR_Test)is less than the DR value in the training stage (DR_Train), or the FAR in the testing process (FAR_Test) is greater than FAR value in the training process (FAR_Train), the performance of the model degraded. If DR and FAR values not changed, the system is stable. For degrading or stable performance, we train the model using new data and tune hyper-parameters to increase the model detection capability. Two methods are used for the new data during model update. The model is trained by using only new data or by combining samples of the new with old training data. Finally, the updated model is evaluated, and the current model is replaced with the updated model. We monitor the updated and current model at appropriate time

Algorithm 1 Performance Monitoring and Updating

```
1:  Input: Detection Rate (DR) and False Alarm Rate (FAR)
    values.
2:  Output: Replace or keep current DL model.
3:  Collect new data from edge layer devices as discussed in
    section IV-.1.
4:  Test the performance of current attack detection every
    month.
5:  Evaluate the current detection model with samples of new
    data.
6:  if (DR_Test < DR_Train) or (FAR_Test > FAR_Train)
    then
7:      Performance degraded
8:      Train current model with new data or combine new and
        old data.
9:      Hyper-parameters tuning.
10:     Evaluate the performance of current model with new
        data.
11:     Deploy updated and current model and monitor them
        in parallel.
12:     if Updated model outperform current model then
13:         Replace current model with updated model
14:     else
15:         Keep current model
16:     end if
17: else
18:     if Performance Improved then
19:         Investigate changes in new data
20:     else
21:         Go to step 8
22:     end if
23: else
24:     Keep monitoring current attack detection perfor-
        mance.
25: end if
```

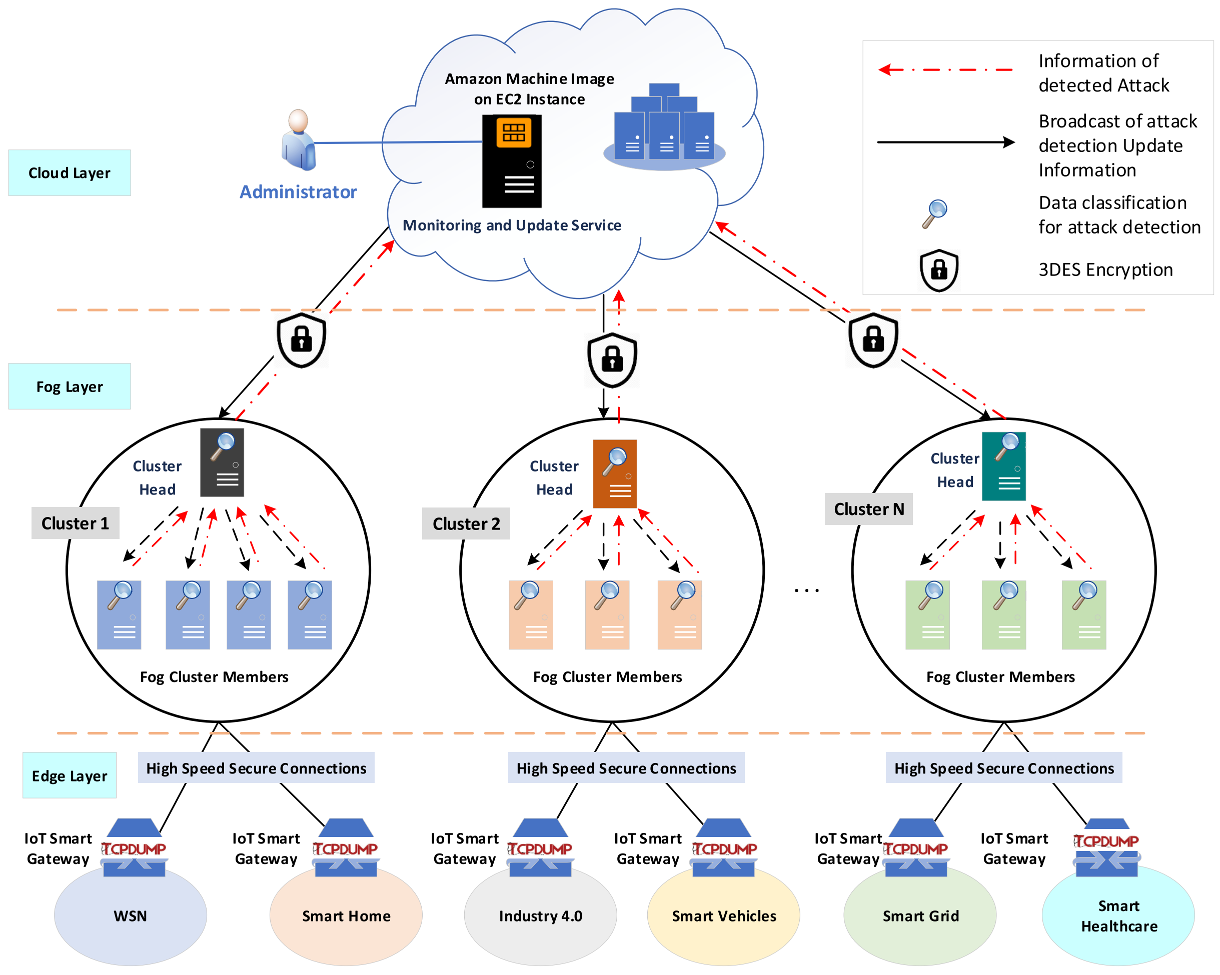


FIGURE 3. **Architecture of the proposed Fog-based attack detection framework.**

intervals. Doing so enables us to completely update the attack detection on all fog nodes or maintain the function of the existing model. Algorithm 1 shows the steps in monitoring and updating the performance of the proposed attack detection.

V. OVERVIEW OF DL MODELS AND DATASETS USED IN THE EXPERIMENTS

DL is a subset of ML, and DL is biologically inspired by the human brain and neurons [18]. DL consists of supervised learning (discriminative learning), unsupervised learning (generative learning), and hybrid learning. In this section, we introduce an overview of different discriminative learning models used in the experiments. Artificial neural networks (ANNs) are divided into three classes, namely, MLP, CNN and RNN. These classes have flexible architectures and have proven their success in various problems. We use LSTM, bidirectional LSTM (BiLSTM), and gated recurrent unit (GRU) as the representative of RNNs. We select DNN as the representative of MLPs. CNN-LSTM is selected to represent hybrid network models.

A. RNN

RNN is a supervised DL model that is designed to handle the sequential data of some applications, such as speech recognition, machine translation, music generation, and sentiment analysis. RNN used to detect botnet behavior in [19] and proposed to build an intelligent network attack detection method in [32] which outperformed SVM. RNN can be considered a group of cells in which each cell performs the same operation on every element in the sequence. In the architecture of unrolled RNN, each cell has input X and output Y and three weight matrices, namely, U, W, and V for the inputs, hidden states, and outputs respectively. U, W and V matrices have the same values in all time steps for different inputs. Thus, the total number of variables is reduced and RNN performs faster than other DL models. We denote the input sequence as $X = (x_0, x_1, x_2, \ldots, x_n)$, the hidden vector as $H = (h_0, h_1, h_2, \ldots, h_n)$, and the output sequence as $Y = (y_0, y_1, y_2, \ldots, y_n)$. The output sequence values are calculated as follows:

$$h_t = \sigma(Ux_t + Wh_{t-1} + b_h) \tag{1}$$

$$y_t = Vh_t + b_y \tag{2}$$

where σ is a nonlinearity activation function, x_t is the input value at time t, h_{t-1} is the hidden state of (t-1), and b_h and b_y are the bias values. RNN uses backpropagation through time to calculate the weights of RNN to minimize the error in the network output compared to expected output. RNN suffers from drawbacks; for example, it cannot memorize long sequences and it is prone to vanishing and exploding gradient problems [33]. Furthermore, it only uses the information that is earlier in the sequence to make a prediction but not use information later in the sequence. Thus, RNN variants such as GRU and LSTM, have been proposed to overcome these problems.

B. LSTM

LSTM is a variant of RNN that designed to deal with vanishing and exploding gradient problems. LSTM was introduced by Hochreiter and Schmidhuber in 1997 [34]. LSTM can learn from long dependencies. The LSTM cell has three gates, namely, forget, input, and output gates that control and protect cell states. The Input gate, forget gate, output gate and state of the memory cell are calculated as follows:

$$i_t = \sigma(W_i h_{t-1} + W_i x_t + b_i) \quad (3)$$
$$f_t = \sigma(W_f h_{t-1} + W_f x_t + b_f) \quad (4)$$
$$o_t = \sigma(W_o X_t + W_o h_{t-1} + b_o) \quad (5)$$
$$h_t = o_t \tanh(c_t) \quad (6)$$
$$c_t = f_t c_{t-1} + i_t \tanh(W_c x_t + W_c h_{t-1}) \quad (7)$$

where i_t, f_t, o_t, c_t are input, forget, output gates, and cell state respectively. σ is the sigmoid function, b is the bias, x_t is the value of input layer at time t, h_t is the hidden state of the cell at time t and W is the weight values. Stochastic gradient descent (SGD) is an optimization method that commonly used in DL to get the minimum loss function that used to update the weights of the neural network through backpropagation using learning rate α. SGD updates the current weight w using gradient $\partial(L)/\partial(W)$ multiplied by α

$$w_{t+1} = w_t - \alpha \frac{\partial(L)}{\partial(w_t)} \quad (8)$$

Authors in [9] used LSTM to detect and classify permission-based android malware which achieved the highest accuracy using real-world Android malware test dataset. In our proposed framework, we use LSTM in attack detection. The performance of the LSTM model discussed in details in section VI.

C. BiLSTM

Bidirectional LSTM proposed in [35] to extract spatial features and bidirectional temporal dependencies from historical data. BiLSTM is developed for speech recognition, handwritten recognition, and protein structure prediction. It obtains the best benefits from an input sequence on the basis of previous and future sequences. It duplicates the first recurrent layer in the network and places them together. The input sequence fed to the first layer as it is and a reverse copy of the input is fed to the second layer. The first and second recurrent layers are connected to the same output layer.

D. GRU

In 2014, Cho presented GRU based on the LSTM network model with few parameters [36]. GRU used in polyphonic music modeling, speech signal modeling, and handwriting recognition. GRU has a simpler structure and fewer cell components than LSTM. It resists the vanishing gradient problem and trains faster because of its small number of computations. GRU has two gates, namely, update (z) and reset gates (r). The update gate and reset gate are calculated as follows:

$$z_t = \sigma(W_z h_{t-1} + W_z x_t + bz) \quad (9)$$
$$r_t = \sigma(W_r h_{t-1} + W_r x_t + br) \quad (10)$$
$$c_t = \tanh(W_h x_t + W_h(h_{t-1} \odot r_t) + bh) \quad (11)$$
$$h_t = (z \otimes c) \oplus ((1 - z) \otimes h_{t-1}) \quad (12)$$

where z_t, r_t, c_t and h_t are update, reset, cell state and hidden state of the cell. x_t represents the input at time t, W represents the weight values, and b represents the bias values. GRU outperforms LSTM in some cases, especially on small datasets [37]. However, LSTM or GRU can't be used for specific tasks. GRU is faster than LSTM in training and can generalize using few data. LSTM requires voluminous data during training and consumes more time than GRU, but it provides better performance in large-scale tasks. GRU used to detect attacks in automated process control systems in [38].

E. CNN

CNNs have shown remarkable performance in object recognition in images [39]. Convolutional, pooling and fully connected layers are stacked with each other to create the CNN architecture. The first layer in CNN is the convolutional layer, which uses multiple equal size filters to convolute input data parameters. If we have a two-dimensional image, I, and a two-dimensional smoothing kernel, K, the convoluted image would be calculated as follows:

$$S(i,j) = \sum_m \sum_n I(m,n) K(i-m, J-n) \quad (13)$$

The pooling layer conducts down sampling for the representation of spatial dimensions (width, height) through max pooling or average pooling operations and reduce the number of parameters and therefore, overfitting. CNN uses "dropout" to reduce overfitting. CNN achieves high performance in learning from raw data but require a high volume of training data. CNNs require high computational resources during network training because they perform huge computations. Thus, the implementation of CNNs on devices with limited resources (sensors, smart devices) is challenging. As mentioned in the related work section, CNN used in [22] to detect malware by converting the opcode to grayscale images.

F. CNN-LSTM

CNN-LSTM is a hybrid DL model designed for visual time series predictions and textual generation from images sequences, such as activity recognition and video description. The CNN-LSTM architecture combines CNN layers for feature extraction from inputs and LSTM layers for time sequence prediction. CNN-LSTM has achieved improvements in speech recognition on DNN. It used in visual recognition and description in [40].

G. DATASETS DESCRIPTIONS

Most researchers have used the KDDCUP99 dataset for training and evaluating their proposed models. We use five datasets to train and evaluate six supervised DL models in binary and multi-class classifications. We use two new IoT datasets (RPL-NIDS-2017 and N_BaIoT-2018) to detect IoT attacks and other datasets (UNSW-NB-2015, CICIDS-2017 and NSL-KDD) to detect conventional attacks. The datasets have been chosen based on the novelty and diversity of attack they have. We select them to prove that the proposed framework able to detect conventional attacks like U2R, R2L, FTP, Worms, Port Scan, etc. besides the IoT attacks. Furthermore, we need to prove that the adoption of DL in attack detection in IoT is a successful idea and powerful tool.

The first data set is **UNSW-NB15** for IDSs [41]. This dataset created at the Cyber Range Lab of the Australian Cyber Security Center. It has nine attack categories, namely, fuzzers, analysis, backdoor, DoS, exploits, generic, reconnaissance, shellcode and worms. The dataset is expressed in CSV format and contains a total of 47 features (state, duration, protocol, service, etc.). We use the dataset file named (UNSW-NB15-1.csv) because DL models require a large volume of data for training. This dataset has 700,000 records, and we split the dataset into 490,000 samples for training and 210,000 samples for validation. The dataset has the following distribution: 677,763 normal traffic, 7,522 generic packets, 5,409 exploits, 5,050 fuzzers, 1,759 reconnaissance, 1,167 DoS, 533 backdoor, 526 analysis, 223 shellcode and 48 worms. All categorical features are encoded into discrete features, and the entire data set is normalized using scikit-learn by rescaling each row to have a length of 1. In our experiment, we train the model for binary and multi-class classifications. For binary classification, we use the dataset in two classes (normal and attack). For multi-class classification, we use the dataset in ten classes (normal, fuzzers, analysis, backdoor, doS, exploits, generic, reconnaissance, shellcode, and worms).

The second dataset is **CICIDS-2017** for ID evaluation [42]. This dataset was created at the Canadian Institute for Cybersecurity (CIC), University of New Brunswick, Canada. The dataset contains benign and up-to-date common attacks such as Web-based, brute force, DoS, DDoS, infiltration, heart-bleed, bot and scan attacks. CICFlowMeter is used for network traffic analysis to generate the CSV files from PCAP traffic files. The CICFlowMeter is a software that available to public on the website of CIC. The CSV files contain 80 network traffic features and labels. The dataset is built on the basis of abstract behavior of 25 users with different protocols, such as the HTTP, FTP, SSH, and email protocols. To train the DL models, we combine four CSV files containing normal traffic, DDoS, portscan, web, SSH, and FTP attacks. The combined dataset has 745,423 records (packets), and we split the dataset into 521,796 samples for training and 223,627 samples for validation. The dataset has the following distribution: 459,199 normal, 123,534 DDoS, 147,329 portscan, 7,935 FTP-Patator, 5,897 SSH-Patator, 1,507 web attack-brute force, and 22 web attack-SQL-injection. In the experiment, we train the model for binary and multi-class classifications. For binary classification, we use the dataset in two classes (normal vs attack). For multi-class classification, we use the dataset in seven classes (normal, DDoS, FTP-patator, SSH-patator, web attack-brute force, exploits and web attack-SQL-injection).

The third dataset is **RPL-NIDS17** for IDS in RPL-based 6LoWPAN networks [43]. This dataset is developed by collecting traces after simulating different routing attacks against RPL routing protocol. The dataset has 21 features such as control_packet_type, source_id, destination_id, app_layer_arrival_time and so on, as well as 1 label. The dataset is created with and without feature encoding, and we use the dataset with feature encoding for training and testing. For binary classification, the training dataset has 116,679 records of normal traffic and 33,337 records of attacks. The testing dataset has 59,560 records of normal traffic and 16,971 records of attacks. For multi-class classification, the dataset has seven classes of attacks (clone-ID, hello flooding, local repair, selective forwarding, sinkhole, sinkhole and blackhole, and sybil) and normal records. The training dataset is distributed as follows: 116,679 normal, 4,405 clone ID attack, 4,822 hello flooding, 4,822 local repair, 4,822 selective forwarding, 4,822 sinkhole, 4,822 sinkhole and blackhole, and 4,822 sybil. The testing dataset is distributed as follows: 59,560 normal, 2,225 clone ID attack, 2,408 hello flooding, 2,450 local repair, 2,424 selective forwarding, 2,495 sinkhole, 2,485 sinkhole and blackhole, and 2,484 of sybil.

The fourth dataset is the **N_BaIoT** dataset developed for detecting IoT botnet attacks using anomaly detection techniques [44]. The dataset contains real traffic collected from nine commercial IoT devices including Danmini doorbell, Ecobee thermostat, and Provision PT-737E security camera. The authors compromised the IoT devices used in their testbed using Mirai and BASHLITE botnet attacks. The dataset contains five attacks of BASHLITE (scan, junk, UDP, TCP, and COMBO) and five attacks of Mirai (scan, Ack, Syn, UDP, UDPplain). To prove that the proposed framework can efficiently detect multiple attacks in several datasets especially in IoT datasets, we evaluate our proposed approach with five attacks of Mirai on a Danmini doorbell. The dataset is expressed in CSV format and has 115 numerical features. The dataset is distributed as follows: 49,548 normal traffic, 102,194 Ack, 107,685 scan, 122,573 syn, 237,665 UDP, and 81,982 UDPplain attacks. The N_BaIoT dataset is used for binary and multi-class classification. For binary classification, we use the dataset in two classes (normal vs attack) with

49,548 records for normal traffic and 652,099 for attacks. For multi-class classification, we use the dataset in six classes (normal, scan, Ack, Syn, UDP, UDPplain).

The fifth dataset is the **NSL-KDD** dataset, which is a refined version of the KDD CUP 99 dataset [45]. The NSL-KDD dataset is widely used in research to evaluate different IDSs. Although, the NSL-KDD dataset has inherent disadvantages, we use it to compare our proposed attack detection framework with state-of-the-art IDSs. The NSL-KDD dataset is available in multiple formats, such as TXT and ARFF formats for training and testing files. We used the CSV file format for training and testing datasets. Each record in the dataset has 41 attributes, including duration, protocol_type, service, flag, src_bytes, and dst_bytes. The dataset has five categories, namely, DoS, user to root (U2R), remote to local (R2L), probe attacks, and normal category. The 42nd column in the dataset contains the data about the five classes categorized as normal or one of the classes of four attacks. The dataset has 125,973 records for training and 22,544 records for testing. The training dataset is distributed as follows: 67,343 normal traffic, 45,927 DoS, 11,656 probe, 995 R2l and 52 U2R. The testing dataset is distributed as follows: 9,711 normal, 7,458 DoS, 2,754 probe, 2,421 R2L, and 200 U2R. In our experiment, we train the model for binary and multi-class classifications. For binary classification, we use the dataset in two classes (normal and attack). For multi-class classification, we use the dataset in five classes (normal, DoS, probe, U2R and R2L).

VI. EXPERIMENTS AND RESULT EVALUATION

All DL models used in our experiments are implemented using Keras on TensorFlow. Keras is a high-level API for building and training DL models. All the experiments are conducted on a personal computer with Intel Core i5-7400 CPU @ 3.00 GHz, 8 GB memory, and CPU-enabled TensorFlow on 64-bit Windows 10. We implement several types of supervised DL models such as GRU, LSTM, CNN, CNN-LSTM, and DNN. All these models can extract deep features from the raw data fed to them. The features extracted by the DL models are compared with the test features in the detection phase. To prove the efficiency of the proposed fog-based attack detection framework in terms of response time, we calculate the response time of the proposed attack detection in fog-based and cloud-based. We calculate DR and FAR as evaluation metrics for evaluating the DL models. Furthermore, precision, recall, F1-Measure and detection time have been used for performance evaluation and comparison. DR refers to the proportion of the total number of correct classifications. FAR refers to the proportion of normal events incorrectly classified as malicious. Accuracy is the percentage of correctly classified samples over the total number of samples. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall calculates the number of positive samples classified as positives. F1-score is the weighted average of precision and recall. Detection time is the time for classifying one packet as normal or an attack. The DL model can be trained offline at a cloud server using GPU to accelerate the training, although the training time is not important. In our experiment, we calculate the average detection time by running the detection 100 times and calculating the average time. The following equations shows the mathematical representation of the evaluation metrics.

$$\begin{aligned} ACC &= \frac{TP+TN}{TP+TN+FP+FN}, \\ FAR &= \frac{FP}{FP+TN}, DR = \frac{TP}{TP+FN} \\ Precision &= \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN}, \\ F1-Measure &= 2 * \frac{Precision * Recall}{Precision + Recall} \end{aligned}$$

To get the best-trained models in binary classification, we use 0.1, 0.01 and 0.001 for learning rate, 32, 64 and 128 for batch size, 100 for epoch number, and Adam optimization algorithm for all DL models. We got the best results using 0.01 for learning rate, 64 for batch size, and Adam as an optimization algorithm. DNN has an input layer with 1024 cells, and five hidden layers with 512 cells each using ReLU activation function, and one output layer with one cell using sigmoid activation function. LSTM has an input layer with 128 cells, and three hidden layers with 256 cells each, and one output layer with one cell using sigmoid activation function. Bi-LSTM has an input layer with 128 cells, and three hidden layers with 128 cells each, and one output layer with one cell using sigmoid activation function. GRU has an input layer with 64 cells, and three hidden layers with 64 cells each, and one output layer with one cell using sigmoid activation function. CNN has three convolutional layers with 64 filters each using ReLU activation function, three pooling layers, and one output layer with one cell using sigmoid activation function. CNN-LSTM has three convolutional layers with 64 filters each using ReLU activation function, three pooling layers, one LSTM layer with 256 cells, and one output layer with one cell using sigmoid activation function. In multi-class classification, the DL models have the same configuration in binary classification but the output layer has number of cells equal to the number of classes, and softmax activation function used instead of the sigmoid. We use dropout to prevent overfitting.

We train and evaluate every DL model for binary and multi-class classifications with five datasets. Each dataset comprises different attacks with different patterns and is used to evaluate the capability of DL models of detecting different patterns from raw data. The DL model with the best performance across all datasets is then identified. Table 2 shows the evaluation metric values for all DL models and their performance in binary classification. For the UNSW-NB15 dataset, LSTM achieves the highest accuracy (99.96%), DR (99.97%), precision (99.98%), recall (99.97%), and F1-Measure (99.98%). The performance of Bi-LSTM is close to that of LSTM, whereas the CNN-LSTM

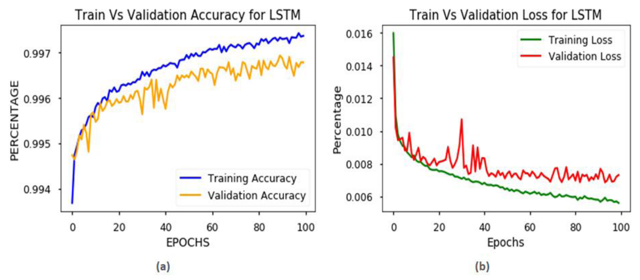


FIGURE 4. **LSTM performance with UNSW-NB15 dataset in binary classification a) accuracy performance b) loss performance.**

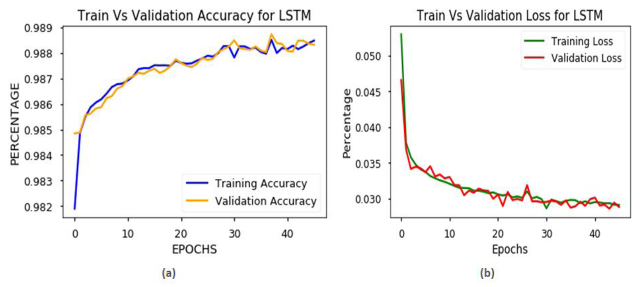


FIGURE 5. **LSTM performance with UNSW-NB15 dataset in multi-class classificationa) a) accuracy performance b) loss performance.**

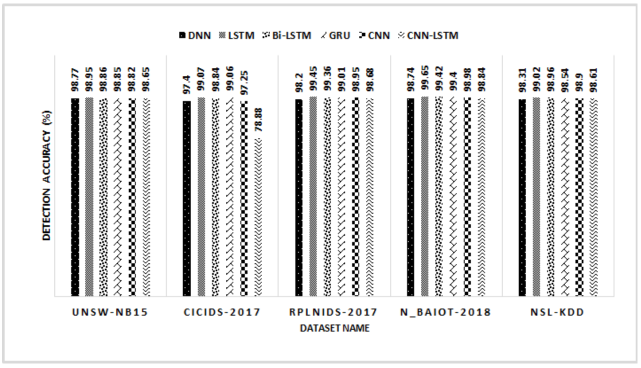


FIGURE 6. **Detection accuracy values of six DL models with five datasets.**

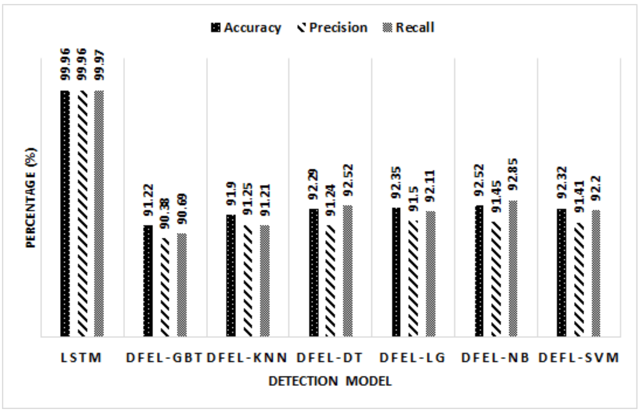


FIGURE 7. **Performance comparison between LSTM and modified ML models proposed in [46] on UNSW-NB15 dataset.**

achieves the lowest FAR and accuracy. The Bi-LSTM outperforms the GRU, CNN, and CNN-LSTM models. For the CICIDS-2017 dataset, the LSTM model achieves the highest accuracy (99.37%), precision (99.28%), F1-Measure (99.49%), and FAR (1.15%), among all the models, and the DNN model achieves the lowest performance. The Bi-LSTM model ranks second after the LSTM model and outperforms DNN, GRU, CNN, and CNN-LSTM. For the RPLNIDS-2017 dataset, LSTM is superior to all other DL models in terms of accuracy (98.15%), DR (99.07%), recall (99.07%), and F1-Measure (99.12%), whereas the CNN model achieves the lowest FAR (11.38%) and highest precision (99.2%). CNN-LSTM achieves the lowest performance and least accuracy, precision, and F1-Measure. N_BaIoT evaluation results show that LSTM is the best in terms of (99.81%), and lowest FAR (0.1), and highest accuracy (99.85%). Whilst, DNN provides the lowest performance. LSTM achieves detection accuracy (99.34%), and FAR (0.1) in the NSL-KDD dataset. LSTM is superior and outperforms all other DL models used in the experiments for binary classifications using the five datasets. GRU consumes less detection time than LSTM because it has fewer parameters and computations but LSTM outperform GRU in DR, FAR and detection accuracy.

Fig.4 and 5 show the performance of LSTM for the UNSWNB15 dataset in the training and validation phases in terms of accuracy and loss values. For binary classification, Fig.4 (a) shows the increase in the training and validation accuracy with the increase of epochs to reach the best accuracy after 100 epochs with a 64 batch size. Fig.4 (b) shows the decrease in training and validation losses that converged after 100 epochs with a 64 batch size. For multi-class classification, Fig.5 (a, b) show the accuracy and loss performance of LSTM with UNSW-NB15. Its accuracy reaches its maximum (98.82%) at 50 epochs, and its loss values to reach its minimum (0.0288) at 50 epochs.

For multi-class classification, every dataset has different numbers and types of attacks. For example, the CICIDS-2017 dataset has seven traffic categories with one normal category and six categories of attacks, and UNSW-NB15 has ten traffic categories one normal and nine categories of attacks. An imbalance of classes that is reflected on the performance of different DL models. Considering space limitation, we present the performance metrics of one dataset (CICIDS-2017) in detail for multi-class classification. Table 3 shows the values of precision, recall, and F1-Measure for every DL model used in the experiments. LSTM is superior to the other DL models in terms of overall performance for all classes. Fig.6 shows the accuracies of the DL models with five datasets in multi-class classification and it illustrates that LSTM is superior to all other DL models.

Fig.7 illustrates the comparison of the LSTM DL model and modified ML models proposed in [46] for the UNSW-NB15 dataset. The proposed ML algorithms are deep feature embedding learning with gradient boosting tree, KNN, DT,

TABLE 2. Evaluation metrics of DL models with the five datasets in binary classification.

DataSet Name	DL Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Measure (%)	FAR (%)	DR (%)	Time (msec)
UNSW-NB15 Dataset	DNN	99.67	99.79	99.87	99.83	6.23	99.87	0.1240
	LSTM	99.96	99.96	99.97	99.98	4.02	99.97	0.2364
	Bi-LSTM	99.67	99.82	99.83	99.83	5.35	99.83	0.8245
	GRU	99.58	99.79	99.77	99.78	6.21	99.77	0.1250
	CNN	99.66	99.86	99.78	99.82	4.24	99.78	0.3212
	CNN-LSTM	98.95	99.96	98.97	99.46	1.20	98.97	0.4866
CICIDS-2017 Dataset	DNN	98.95	98.57	99.73	99.15	2.29	99.73	0.1064
	LSTM	99.37	99.28	99.67	99.49	1.15	99.67	0.0954
	Bi-LSTM	99.35	99.22	99.77	99.48	1.25	99.77	0.4110
	GRU	99.35	99.21	99.73	99.47	1.26	99.73	0.0487
	CNN	99.08	98.61	99.92	99.26	2.25	99.92	0.2380
	CNN-LSTM	98.88	98.41	99.8	99.1	2.57	99.8	0.5494
RPLNIDS-2017 Dataset	DNN	98.01	98.97	98.88	98.93	13.31	98.88	0.0857
	LSTM	98.15	98.99	99.07	99.12	12.25	99.07	0.2746
	Bi-LSTM	98.01	98.97	98.88	98.93	13.31	98.88	0.5915
	GRU	98.01	98.97	98.88	98.92	13.08	98.88	0.1425
	CNN	97.94	99.02	98.74	98.88	11.38	98.74	0.2965
	CNN-LSTM	97.01	97.73	98.92	98.40	29.78	98.92	0.4924
N_BaIoT-2018 Dataset	DNN	98.9	91.88	92.43	92.15	0.6	92.43	0.1570
	LSTM	99.85	98.64	99.81	99.12	0.1	99.81	0.1425
	Bi-LSTM	99.81	97.32	99.8	98.63	0.2	99.8	0.3198
	GRU	99.57	96.36	98.28	97.31	0.31	98.28	0.1230
	CNN	99.41	97.83	94.97	96.38	0.18	94.97	0.1834
	CNN-LSTM	99.39	99.19	93.64	96.34	0.7	93.64	0.3394
NSL-KDD Dataset	DNN	98.24	98.96	98.63	98.29	10.2	98.63	0.1991
	LSTM	99.34	99.18	99.59	99.39	0.1	99.59	0.2531
	Bi-LSTM	99.07	98.61	99.68	99.14	1.6	99.68	0.5510
	GRU	98.85	98.56	99.31	98.93	1.6	99.31	0.1784
	CNN	99.03	99.16	99.51	99.13	0.3	99.51	0.3942
	CNN-LSTM	96.07	94.19	99.77	96.43	7.03	98.77	0.4986

TABLE 3. Performance of DL models with CICIDS-2017 dataset in multi-class classification.

Attacks	DNN			LSTM			Bi-LSTM		
	P (%)	R (%)	F1-M (%)	P (%)	R (%)	F1-M (%)	P (%)	R (%)	F1-M (%)
Benign	98.41	99.71	99.06	99.48	99.15	99.31	99.04	99.24	99.14
DDoS	99.18	57.77	73.01	98.89	98.94	98.91	99.49	97.16	98.31
FTP-Patator	84.38	56.41	67.62	91.05	99.14	94.93	93.84	98.94	96.32
Port-Scan	47.66	74.78	58.21	99.42	99.91	99.66	99.21	99.97	99.59
SSH-Patator	39.31	29.27	33.56	100	70.68	82.82	93.73	79.03	85.75
Brute-Force	30.23	4.12	7.26	44.74	84.65	58.54	44.81	85.77	58.86
SQL-Injection	0	0	0	11.11	2.56	4.28	0	0	0
	GRU			CNN			CNN-LSTM		
	P (%)	R (%)	F1-M (%)	P (%)	R (%)	F1-M (%)	P (%)	R (%)	F1-M (%)
Benign	99.81	98.86	99.33	95.78	99.94	97.82	74.5	99.89	85.35
DDoS	99.06	98.94	99	99.96	98.68	99.32	99.77	98.93	99.35
FTP-Patator	89.13	99.79	94.16	99.76	84.58	91.54	96.42	49.2	65.16
Port-Scan	99.79	99.93	99.86	99.96	91.28	95.42	91	0.57	1.13
SSH-Patator	74.04	98.76	84.63	98.57	50.64	66.91	99.56	50.98	67.43
Brute-Force	43.8	84.65	57.73	26.14	3.16	5.64	0	0	0
SQL-Injection	0	0	0	0	0	0	0	0	0

TABLE 4. Table-comparison.

	Proposed Framework	Proposed attack detection in [5]
Architecture	distributed	distributed
DL model Training	at cloud layer	at fog layer
Layers used for implementintation	fog and cloud layers	fog layer
Number of DL models used	six DL models	one DL model
Number of datasets used	five datasets	one dataset
Detection Accuracy	99.85 % in binary classification 99.65% in multi-class classification	99.2 % in binary classification 98.27% in multiclass classification
Scalability	scalable	limited scalability

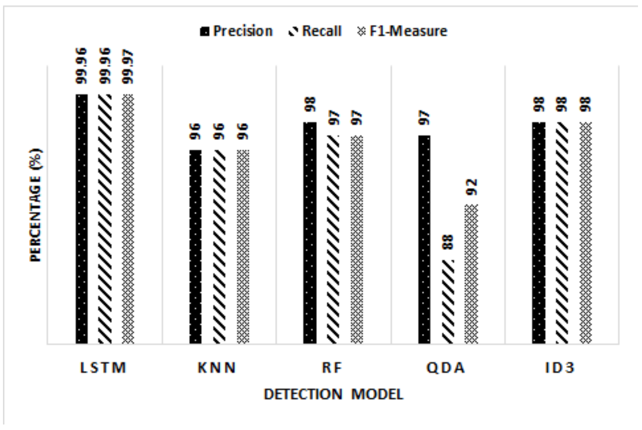


FIGURE 8. Performance comparison between LSTM and ML algorithms used in [42] on CICIDS-2017 dataset.

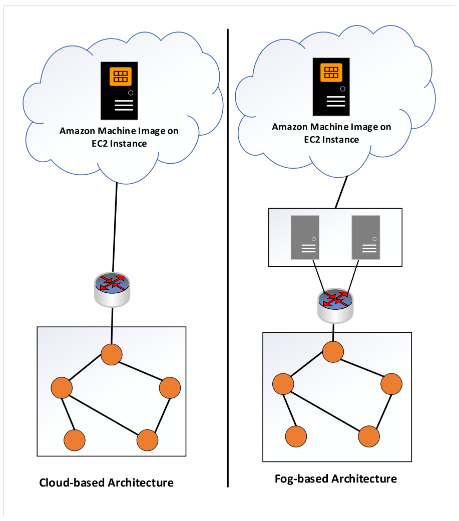

Amazon Machine Image on EC2 Instance
Amazon Machine Image on EC2 Instance
Cloud-based Architecture
Fog-based Architecture

FIGURE 9. Experiment different scenarios.

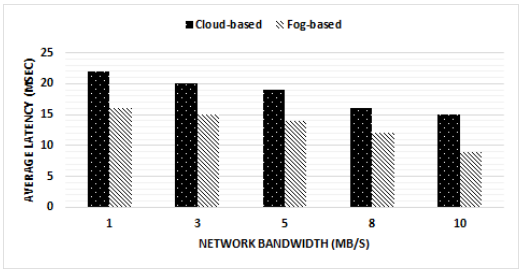


FIGURE 10. Average response time for fog-based and cloud-based detection.

LG, NB and SVM. Fig.7 shows that LSTM is clearly better than all ML algorithms in terms of accuracy, precision, and recall. Fig.8 shows another comparison between LSTM and the KNN, RF, ID3 and quadratic discriminant analysis (QDA) for the CICIDS-2017 dataset used in [42]. The proposed attack detection outperforms all ML algorithms in terms of precision, recall, and F1-score.

The proposed attack detection is similar to the proposed attack detection in [20] but with some differences. Table 4 illustrates the differences and similarities, of both attack detection frameworks. The proposed attack detection is trained at the cloud layer and implemented in the fog layer, whereas the other attack detection is trained and run in the fog layer. We use six DL models, whereas the other attack detection uses only one DL model. We use five datasets in training and testing the DL models, whereas the other model use only one dataset. Our proposed attack detection achieves high DR in binary and multi-class classification. Our proposed framework is also more scalable than other attack detection.

To compare the efficiency of the proposed fog-based attack detection system in terms of response time, we implement the proposed attack detection framework in fog-based and cloud-based architecture as shown in Fig.9. We use Cooja simulator [47] to simulate the wireless sensor network (WSN) motes to represent the IoT edge layer and use CONTIKI-NG [48] platform to implement IoT WSN motes. The fog layer nodes implemented using two computers to represent the fog layer nodes, they connected with the simulated edge sensors. In the cloud-based architecture, we implement the attack detection framework in the cloud using Amazon EC2 virtual server instance. We measure the response time 10 times and calculate the average value for different network speeds. As shown in Fig.10, the response time of the proposed fog-based architecture is less than the cloud-based since the fog nodes is closer to the edge layer and can detect attack with low latency.

LSTM can learn from long sequences and bridge inputs with long time gaps by using the forget gate to store information about the previous state of the network. Thus, LSTM can use historical data from previous network traffic to detect attacks, such as DoS and DDoS, for a long time period. The data generated from IoT devices are unstructured, and LSTM can learn effectively from unstructured data and extract deep insights. Moreover, the performance of LSTM increases with the increase of training data volume because data are the heart and soul of DL. These conditions lead to the superiority of LSTM to the rest of the DL models. The deep structure, feature hierarchy, and the huge number of weights and variables calculated by DL models make them better than ML algorithms.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have proposed an attack detection framework based on LSTM DL model that used for IoT traffic classification. In the proposed framework, the edge layer traffic collected and sent to the cloud layer to train the LSTM model. Then, the trained model installed on the fog layer nodes as a detection engine to detect attacks. Fog computing provides a distributed environment with many fog nodes near to IoT devices in the edge layer. Thus, we implement the detection system on the fog nodes to analyze the data close to the edge layer to minimize latency. We monitor the performance of the LSTM model and update it using a cloud service. The experiments have shown the success of the DL models to be adopted to Cybersecurity to detect several attacks with high detection and accuracy rates. It is also demonstrated that the DL models can detect conventional and Cyber-attacks existed in different datasets. We conclude that the LSTM model is superior to all other supervised DL models used in the experiment because it has forget gate to store the previous state information and can learn from long sequences. This proposed framework overcomes the problems of how to implement the heavy DL detection system directly on limited capacity IoT devices, detect several attacks with high detection rate and high accuracy rates, and how to monitor the detection system and update it to detect new attacks. However, it has a drawback in labeling the data that collected in the edge layer to train the LSTM model in the cloud may be difficult. In the future, we will compare the proposed attack detection with unsupervised DL models and reinforcement learning using a distributed computing environment like Apache Spark with different datasets.

REFERENCES

[1] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in Internet-of-Things," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1250–1258, Oct. 2017.

[2] B. B. Zarpelão, R. S. Miani, C. T. Kawakani, and S. C. de Alvarenga, "A survey of intrusion detection in Internet of Things," *J. Netw. Comput. Appl.*, vol. 84, pp. 25–37, Apr. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517300802

[3] M. E. Karsligel, A. G. Yavuz, M. A. Güvensan, K. Hanifi, and H. Bank, "Network intrusion detection using machine learning anomaly detection algorithms," in *Proc. 25th Signal Process. Commun. Appl. Conf. (SIU)*, May 2017, pp. 1–4.

[4] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 305–316.

[5] A. Diro and N. Chilamkurti, "Leveraging LSTM networks for attack detection in Fog-to-Things communications," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 124–130, Sep. 2018.

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

[7] M.-J. Kang and J.-W. Kang, "Intrusion detection system using deep neural network for in-vehicle network security," *PLoS ONE*, vol. 11, no. 6, Jun. 2016, Art. no. e0155781, doi: 10.1371/journal.pone.0155781.

[8] Y. Chen, Y. Zhang, S. Maharjan, M. Alam, and T. Wu, "Deep learning for secure mobile edge computing in cyber-physical transportation systems," *IEEE Netw.*, vol. 33, no. 4, pp. 36–41, 2019.

[9] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Deep Android malware detection and classification," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Sep. 2017, pp. 1677–1683.

[10] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[11] N. McLaughlin, J. M. del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, and G. J. Ahn, "Deep Android malware detection," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2017, pp. 301–308, doi: 10.1145/3029806.3029823.

[12] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 3854–3861.

[13] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[14] A. Alrawais, A. Alhothaily, C. Hu, and X. Cheng, "Fog computing for the Internet of Things: Security and privacy issues," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 34–42, Mar. 2017.

[15] W. Zhou, Y. Jia, A. Peng, Y. Zhang, and P. Liu, "The effect of IoT new features on security and privacy: New threats, existing solutions, and challenges yet to be solved," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1606–1616, Apr. 2019.

[16] M. Ali Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, and M. Guizani, "A survey of machine and deep learning methods for Internet of Things (IoT) security," *CoRR*, vol. abs/1807.11023, 2018, *arXiv:1807.11023*. [Online]. Available: http://arxiv.org/abs/1807.11023

[17] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," *J. Big Data*, vol. 2, no. 1, Dec. 2015, doi: 10.1186/s40537-014-0007-7.

[18] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, 4th Quart., 2018.

[19] P. Torres, C. Catania, S. Garcia, and C. G. Garino, "An analysis of recurrent neural networks for botnet detection behavior," in *Proc. IEEE Biennial Congr. Argentina (ARGENCON)*, Jun. 2016, pp. 1–6.

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

[21] R. Kozik, M. Choraś, M. Ficco, and F. Palmieri, "A scalable distributed machine learning approach for attack detection in edge computing environments," *J. Parallel Distrib. Comput.*, vol. 119, pp. 18–26, Sep. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731518302004

[22] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Secur. Commun. Netw.*, p. 51964–51974, 2018. [Online]. Available: https://www.hindawi.com/journals/scn/2018/7247095/citations/

[23] P. Zhang, M. Zhou, and G. Fortino, "Security and trust issues in fog computing: A survey," *Future Gener. Comput. Syst.*, vol. 88, pp. 16–27, Nov. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731518302004

[24] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile cloud Comput. (MCC)*, New York, NY, USA, 2012, pp. 13–16, doi: 10.1145/2342509.2342513.

[25] A. A. Mutlag, M. K. A. Ghani, N. Arunkumar, M. A. Mohammed, and O. Mohd, "Enabling technologies for fog computing in healthcare IoT systems," *Future Gener. Comput. Syst.*, vol. 90, pp. 62–78, Jan. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18314006

[26] M. Sookhak, F. R. Yu, Y. He, H. Talebian, N. S. Safa, N. Zhao, M. K. Khan, and N. Kumar, "Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing," *IEEE Veh. Technol. Mag.*, vol. 12, no. 3, pp. 55–64, Sep. 2017.

[27] "Fog computing and the Internet of Things: Extend the cloud to where the things are," Cisco Public, White Paper, Sep. 2015, pp. 1–6.

[28] A. Abeshu and N. Chilamkurti, "Deep learning: The frontier for distributed attack detection in fog-to-things computing," *IEEE Commun. Mag.*, vol. 56, no. 2, pp. 169–175, Feb. 2018.

[29] L. Goasduff. (2019). *Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020*. Accessed: Sep. 28, 2019. [Online]. Available: https://gtnr.it/35hq94q

[30] A. H. Lashkari, G. Draper-Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *Proc. ICISSP*, 2017, pp. 253–262.

[31] A. Kousaridas, S. Falangitis, P. Magdalinos, N. Alonistioti, and M. Dillinger, "SYSTAS: Density-based algorithm for clusters discovery in wireless networks," in *Proc. IEEE 26th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Aug. 2015, pp. 2126–2131.

[32] Y. Fu, F. Lou, F. Meng, Z. Tian, H. Zhang, and F. Jiang, "An intelligent network attack detection method based on RNN," in *Proc. IEEE 3rd Int. Conf. Data Sci. Cyberspace (DSC)*, Jun. 2018, pp. 483–489.

[33] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang, "Dilated recurrent neural networks," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2017, pp. 77–87. [Online]. Available: http://papers.nips.cc/paper/6613-dilated-recurrent-neural-networks.pdf

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[35] Z. Cui, R. Ke, and Y. Wang, "Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction," *CoRR*, vol. abs/1801.02143, 2018. [Online]. Available: http://arxiv.org/abs/1801.02143

[36] K. Cho, B. van Merrienboer, C. C. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[37] J. Chung, C. C. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[38] D. Lavrova, D. Zegzhda, and A. Yarmak, "Using GRU neural network for cyber-attack detection in automated process control systems," in *Proc. IEEE Int. Black Sea Conf. Commun. Netw. (BlackSeaCom)*, Jun. 2019, pp. 1–3.

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[40] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *CoRR*, vol. abs/1411.4389, 2014. [Online]. Available: http://arxiv.org/abs/1411.4389

[41] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS)*, Nov. 2015, pp. 1–6.

[42] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy (ICISSP)*, 2018, pp. 108–116.

[43] A. Verma and V. Ranga, "RPL-NIDDS17-A data set for intrusion detection in RPL based 6LoWPAN networks (Internet of Things)," *Int. J. Grid Distrib. Comput.*, vol. 11, no. 8, pp. 43–56, Aug. 2018, doi: 10.5281/zenodo.1406034.

[44] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici, "N-baiot: Network-based detection of IoT botnet attacks using deep autoencoders," *CoRR*, vol. abs/1805.03409, 2018. [Online]. Available: http://arxiv.org/abs/1805.03409

[45] *NSL-KDD Dataset*. Accessed: Feb. 15, 2019. [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html

[46] Y. Zhou, M. Han, L. Liu, J. S. He, and Y. Wang, "Deep learning approach for cyberattack detection," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2018, pp. 262–267.

[47] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proc. 31st IEEE Conf. Local Comput. Netw.*, Nov. 2006, pp. 641–648.

[48] (2019). *Contiki-NG*. Accessed: Jan. 26, 2020. [Online]. Available: https://github.com/contiki-ng/contiki-ng

AHMED SAMY received the M.Sc. degree from the Department of Information Technology, Faculty of Computers and Information, Menoufia University, Egypt, in 2015. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. He has been working as an Assistant Lecturer with the Faculty of Computers and Information, Menoufia University, since August 2011. His research interests include the Internet of Things security and privacy, cyber attacks detection, and deep learning.

HAINING YU received the B.S. and M.S. degrees in computer science, and the Ph.D. degree in information security from the Harbin Institute of Technology, Harbin, China, in 2006, 2008, and 2013, respectively. He is currently an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology. His research interests include security and privacy in the Internet of Things, and cloud computing security.

HONGLI ZHANG (Member, IEEE) received the B.Sc. degree in computer science from Sichuan University, Chengdu, China, in 1994, and the Ph.D. degree in computer science from the Harbin Institute of Technology, Harbin, China, in 1999. She is currently a Professor with the School of Computer Science and Technology, Harbin Institute of Technology. Her research interests include network and information security, network measurement and modeling, and parallel processing.