Low Rate DoS Attack Detection in IoT – SDN using Deep Learning

Harun Surej Ilango
School of Electrical and Electronic Engineering
Nanyang Technological University
Singapore
harunsur001@e.ntu.edu.sg

Maode Ma
College of Engineering
Qatar University
Qatar
mamaode@qu.edu.qa

Rong Su
School of Electrical and Electronic Engineering
Nanyang Technological University
Singapore
rsu@ntu.edu.sg

***Abstract*— The lack of standardization and the heterogeneous nature of IoT, exacerbated the issue of security and privacy. In recent literature, to improve security at the network level, the possibility of using SDN for IoT networks was explored. An LR DoS attack is an insidious DoS attack that hinders the availability of the network to its legitimate users. LR DoS attacks are difficult to detect and can be deadly to a network due to their hidden nature. Recently, the possibility of using ML or DL algorithms to detect LR DoS attacks have gained traction due to advancements in computing technology. The ML and DL algorithms that are currently available in the literature have a detection rate of 95 percent at best. In this work, a novel deep learning scheme called FFCNN is proposed to detect LR DoS attacks in a SDN environment. The CIC DoS 2017 and CIC IDS 2017 datasets provided by the Canadian Institute of Cybersecurity were used for the experimental analysis. The empirical analysis of the proposed algorithm shows that it outperforms the existing machine learning based algorithms. FFCNN promises a lower false alarm rate and better detection rate in the detection of LR DoS.**

Keywords— Internet of Things, Software Defined Networking, Deep Learning, Low-Rate DoS Attacks, Network Security, CIC DoS 2017, CIC IDS 2017

I. INTRODUCTION

Kevin Ashton coined the term 'The Internet of Things' [1] back in 1999. IoT gained traction recently as the internet got accessible to a larger population of the world. As more and more people connect their appliances and devices to the internet, there is a rising concern regarding the security of these smart devices in this connected ecosystem. The nascent applications, lack of standardisation and heterogeneous nature of IoT has only exacerbated this issue. Major limitations that hinder the development of state-of-the-art security solutions are the limited availability of computing capability, storage capacity and power across the layers of the IoT system.

The IoT system consists of three layers: The perception layer, the communication networks layer, and the application layer. The perception layer consists of devices that aggregate environmental data using sensors and actuators that make physical changes to the environment. With their embedded intelligence, the devices are also capable of making intelligent decisions based on locally available data (Edge Computing). The communication layer provides networking services between the perception layer and the application layer; within perception layer devices, and within the application layer services. The application layer takes intelligent decisions based on the data from the lower layers. The decisions are either based on real-time data or aggregated data.

This work focuses on the security of the network layer. To improve security at the network layer of the IoT architecture, Software Defined Networking (SDN) can be used to offload some of the network computation to a central controller. The data plane and the control plane are separated in SDN. The control plane performs the logical operations that are needed to forward the data packets from the source to the destination. The data plane transfers the data packets using the control signals from the control plane. SDN architecture consists of infrastructure layer, control layer and application layer. Networking devices like switches and routers make up the infrastructure layer. Infrastructure layer devices forward the packets based on control signals from the control layer. In contrast, in a conventional packet-switching network, each intermediate node between the source node and the destination node maintains a routing table to forward the arriving data packets. The application layer hosts applications that ensure QoS, perform load balancing and ensure the security of the network. Communication between the application layer and control layer is handled by the Northbound APIs; communication between the control layer and the infrastructure layer is handled by Southbound APIs. SDN enables easier detection of malicious devices in the network due to its centralised nature; improved resistance to network attacks and ensures better overall security to the IoT network.

Any device in a network is subject to attacks that compromise the CIA (Confidentiality, Integrity and Availability) triad. Denial of Service (DoS) attack specifically targets the availability of network resources to its legitimate stakeholders. A variant of DoS, the Low-Rate DoS (LR DoS) attack is an insidious type of attack that has been recently devised to evade unsuspecting normal DoS attack detectors in a network. The data rate of an LR DoS attack is similar to benign network traffic generated by legitimate devices in the network which makes them hard to detect. LR DoS attacks are of particular interest to IoT, as IoT devices usually transmit data at a very low data rate and LR DoS attacks may go undetected in such low network traffic scenarios.

In literature, ElSayed et al [2] proposed a hybrid model that used Convolutional Neural Network (CNN) and Random Forest to classify the arriving flow in a SDN environment as anomalous or benign. The experimental results showed that the hybrid model achieved good performance on a wide range of publicly available datasets. Though the work focuses on DoS attacks in general, it does not focus LR-DoS attacks. Also, the dataset is balanced before performing performance analysis of the proposed system. Balancing the dataset improves the performance during the analysis of the proposed system. But in a real-world scenario, the number of attacks flows is the network is much lower than benign flows and the model needs to perform well in such imbalanced scenarios. Similarly, Abubakar et al [3] proposed an anomaly-based intrusion detection system that was integrated with the

existing signature-based intrusion detection system. Analysis of the integrated system further enhanced the security of the SDN. The authors use the NSL-KDD which is comparatively an older dataset to develop intrusion detection systems. The dataset does not contain any traces of LR-DoS. This makes the IDS susceptible to LR-DoS attacks. A hybrid deep learning model that used a combination of an autoencoder and random forest algorithm was proposed by Isa et al [4]. The experimental study on the NSL-KDD dataset showed a classification accuracy rate of almost 98 percent. On a similar note, a Gated Recurrent Unit Recurrent Neural Network (GRU-RNN) was proposed by Tang et al [5] for anomaly detection. The proposed algorithm showed a detection accuracy of 89 percent on the same dataset. The uniqueness of this work was that it used a minimal number of features, six to be precise, yet achieves very good detection accuracy when compared to other machine learning algorithms using the same number of features. However, GRU-RNN had a significant impact on the throughput performance of the controller.

The performance of Stacked Denoising AutoEncoder (SDAE) integrated with Extreme Learning Machine (ELM) (SDAE-ELM) and Deep Belief Networks (DBN) with a Softmax classifier (DBN-Softmax) were compared by Wang et al [6]. While both the proposed models showed good detection, the DBN-Softmax consumed a significantly longer training time than SDAE-ELM. Though this work does not detect attacks on a SDN environment, it showed how deep learning models enable better and precise detection of network attacks.

Santos et al [7] performed empirical analysis on the performance of SVM, MLP, Decision Tree, and Random Forest algorithms to detect attacks on the layers of the SDN architecture. The analysis of the empirical results showed that the decision tree algorithm took the least time to process whereas random forest promised the highest detection accuracy. Similarly, Perez-Diaz et al [8] specifically tested the performance of machine learning algorithms in detecting LR DoS attacks on a SDN environment. The authors proposed a flexible architecture where the detection and mitigation processes are decoupled from the network. The performance of J48, Random Tree, REP Tree, Random Forest, Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP) were compared. A F1 score of 0.58, 0.63, 0.56, 0.8, 0.93 and 0.95 were obtained respectively.

Considering the best-case scenario for the detection of an LR DoS attack, there is a 5% chance of an attack not being detected. For time-critical applications like IoT, this probability could be the difference between life and death. Identifying the gap in the literature, this work proposes a Feedforward – Convolutional Neural Network (FFCNN) scheme that outperforms the existing LR DoS detection schemes in IoT-SDN environments.

The remainder of the paper is organised in the following way. Section II disseminates the proposed FFCNN scheme. The methodology and the results of the experiments are discussed in Section III and the paper is concluded in Section IV.

II. THE FFCNN SCHEME:

The authors of this work propose a deep learning model, Feedforward – Convolutional Neural Network (FFCNN) for the detection of LR-DoS attacks. FFCNN uses a combination of a feedforward neural network (FFNN) and Convolutional Neural Network (CNN) as its primary units. The proposed FFCNN scheme will be hosted in the application layer of the SDN architecture and will communicate with the control layer through the Northbound API.

Three neural networks are chained to develop FFCNN. In the first stage, a FFNN is used to upscale the input features. The number of neurons in the input layer of FFNN matches the number of input features of the dataset. The subsequent dense layers in FFNN scale the inputs to 144 outputs. Relu activation function was used in each dense layer of FFNN. Relu activation function outputs zero if the input is less than 0 and outputs the input if the value is greater than zero. The output of the Relu layer is then reshaped to a 12 by 12 grid using a reshape filter.

In the next stage, CNN is used to identify the underlying patterns (mine) in the scaled inputs. A filter (kernel) smaller than the input grid is chosen. In FFCNN, 60 3*3 kernels were chosen for the first convolutional layer. The inputs are multiplied over the input grid as the filter is moved across the entire input grid. Eventually, this computation yields feature maps of the input. Relu activation function is used in this step. The use of Relu activation function in convolutional layers increases the non-linearity [2]. As a result of this convolution operation, 60 feature maps with a size of 10*10 were obtained.

A max-pooling layer is applied on the output feature maps of the convolution layer. A max-pooling layer returns the maximum value in the feature map that is overlapped by the kernel grid. In FFCNN, the first max-pooling layer has a 2*2 kernel. As multiple convolutions and max-pooling operations are performed on the inputs, the size of the resulting output converges to 1 eventually. Padding was used to add an additional layer of values on the border of the grid to slow down the convergence to 1. It is to be noted that padding is applied only in the max-pooling layer in FFCNN.

To mine the features further, the next convolutional layer used 120 filters with a kernel size of 3*3. The Max-pooling layer was subsequently applied to the output feature maps of the convolution layers. The resulting 120 feature maps were 2*2. The output feature maps of CNN were flattened to make a 1*480 vector for decision making. Subsequently, the 1*480 vector was given as inputs to another set of dense layers of FFNN to scale down the input vector to just 1 value for decision making. The hidden dense layers of FFNN used the Relu activation function whereas a sigmoid activation function was used in the output layer for decision making.

III. NUMERICAL EXPERIMENTS, RESULTS AND ANALYSIS

To enable the integration of the proposed deep learning model in a SDN environment, Python was chosen as the programming language of choice as it is usually well supported across all operating systems. All computations were carried out on a machine running Windows 10 with Intel Core i7-7700HQ CPU, 16 GB RAM and NVIDIA GTX 1050 GPU. MATLAB R2021a and Python 3.9 were used for data processing and to develop FFCNN, respectively. CUDA 11.4 and Tensorflow 2.5.0 was used to utilise the GPU for training and testing FFCNN.

A. The Datasets:

The CIC DoS 2017 Dataset [9] and CIC IDS 2017 Dataset [10], provided by the Canadian Institute of Cybersecurity (CIC) were used for empirical analysis of the FFCNN. The

CIC DoS dataset was developed by combining the attack free traces from the ICSX 2012 [11] with the generated application layer attacks. In CIC IDS 2017, the dataset that was collected on Wednesday is of specific interest as they contain traces of LR-DoS attacks. The benign traces from the dataset collected on Monday were also used to increase the number of samples to train the FFCNN which results in better performance and generalisation of the model.

Though these datasets explicitly do not contain traces of IoT traffic, the benign traces in these datasets closely resemble IoT network traffic. IoT uses the same TCP/IP protocol as a normal network device. It is hence agreeable to train and test FFCNN using these datasets and compare its performance to other machine learning models using the same set of datasets. The details of the record traces available in each dataset are given in Table I. The selected datasets contain samples of both high volume and low volume DoS attacks which makes the resulting LR-DoS detection FFCNN more robust to DoS attacks in general. GoldenEye, DDoSSim, HTTP Unbearable Load King (HULK) are high volume DoS attacks whereas Slowhttptest, RUDY, Slowris, Slowbody2, Slowheaders and Slowread are low rate DoS attacks.

TABLE I. DESCRIPTION OF DATASETS

Dataset		Labels
CIC DoS 2017		Benign Slowloris HULK GoldenEye DDoSSim RUDY Slowbody2 Slowheaders Slowread
CIC IDS 2017	Monday	Benign
	Wednesday	Benign Slowloris Slowhttptest HULK GoldenEye

B. Dataset Pre-processing:

The CIC Flow Meter was used as the feature extractor, to extract significant features from the network packets. This enables the FFCNN algorithm to understand the characteristics of traffic flow in more detail due to the presence of more significant features. To avoid biasing the FFCNN, personally identifiable features such as source IP, destination IP, ports and protocol were removed from all datasets. Had these features not been removed, FFCNN will associate certain source or destination IPs/Ports to attacks and will generate false alarms. Further, all flag features were removed from the dataset as they are near-zero variance features. Zero variance features were removed from the dataset using the variance threshold feature in scikit learn [12] during runtime. Zero variance and pseudo-zero variance features were removed from the dataset as they are largely time-invariant and do not contribute to decision making.

Any instances of NaN and infinite values were checked and removed from the dataset. To bring the features of the dataset to a common scale, the dataset was normalised (the value of individual features is brought between a value of 0 and 1). The dataset was then encoded (the benign and attack flows were labelled as 0 and 1 respectively). It is to be noted that the resulting dataset is highly imbalanced – the number of benign records (1,219,308) is very much larger than the number of attack records (294,215).

C. Performance Metrics

The entire dataset was split into training and testing subsamples. The testing subset was 20 percent of the entire dataset. The class imbalance was maintained during the split to prevent the model from overfitting the benign class. It was also done to mimic the real-world data stream scenario where the attack packets constitute a very small percentage of the total arriving packets. The model was trained on the training data. Binary cross-entropy and Adam optimiser were used as the loss function and optimiser respectively. The model is trained for 10 epochs with a batch size of 64. After each epoch, validation was done using the test dataset to ensure that a generalised model, that does not overfit the training dataset, is obtained.

The performance of the FFCNN was measured using the metrics - accuracy, precision, recall and F1 score. The metrics precision and recall are suited for highly imbalanced datasets like the one used in this work. Precision is the ratio of records that are true out of all records that were predicted true. Recall is the ratio of records that were predicted true out of all records that were labelled true. However, to arrive at a single metric of performance comparison, F1 score is used. The F1 score is the harmonic mean of recall and precision. The formulas to compute these metrics are given in Eq. 1 – 4.

$$Accuracy = (TP+TN)/(TP+FP+FN+TN) \quad (1)$$

$$Precision = TP/(TP+FP) \quad (2)$$

$$Recall = TP/(TP+FN) \quad (3)$$

$$F1\ score=2\times(Precision\times Recall)/(Precision+Recall) \quad (4)$$

FFCNN was trained using the training subset. During training, to analyse the training performance, the training accuracy, validation accuracy, training loss and validation loss were monitored. Fig. 1 visualizes the above training metrics for further analysis. The testing subset was then given as the inputs to the deep learning model and the model predicted if the arriving flow was benign or attack. The predictions were then used to construct the confusion matrix and to calculate the performance metrics.

From the plots in Fig. 1, it is evident that the training accuracy and validation accuracy increases as the training loss and validation loss drops. It can be concluded that the model is not overfitting, and a good, generalised model is achieved. A model starts to overfit when it begins updating its weights based on the noise in the training data set and extracts features based on it. The model will show excellent performance on the training dataset but will perform poorly on the validation dataset. Had FFCNN overfit the training data, it will be evident from the validation accuracy plot, which is not the case. Subsequently, the test dataset was given as the input to the trained model and the classification performance is shown in Table II as a confusion matrix.

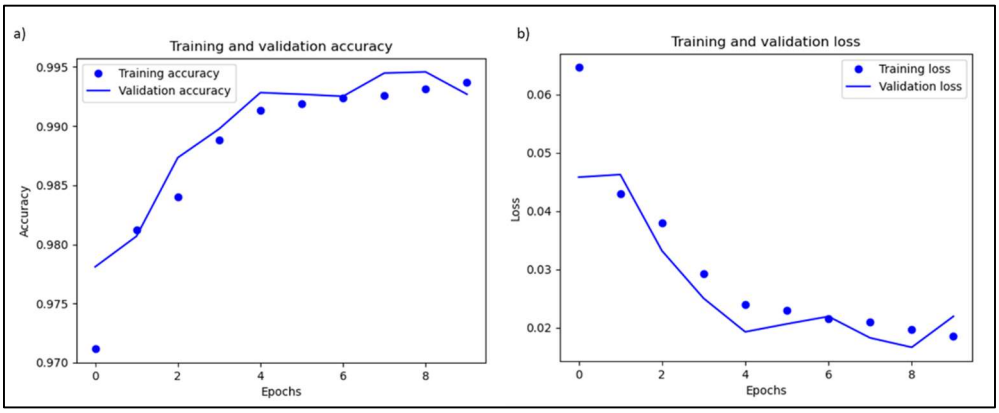

Fig. 1. Training Performance – a) Plot of training accuracy and validation accuracy b) Plot of training and validation loss

TABLE II. CONFUSION MATRIX OF THE MODEL FOR THE TEST DATASET

		Actual	
		Attack	***Benign***
Prediction	***Attack***	58391	1343
	Benign	280	242364

True Positives (TP) are the number of instances in the dataset which were true, and the classifier correctly predicted it as true. Similarly, True Negatives (TN) are records that were correctly predicted as false for records that were labelled false. False Positives (FP) are records in the dataset that were false, but the classifier predicted it as true. False Negatives (FN) are records where the actual label was true, but the classifier predicted it as false. From the confusion matrix in Table II, it is evident that FFCNN has very few FP and FN compared to the entire size of the dataset. From a network administrator point of view, an Intrusion Detection System (IDS) having fewer FPs is highly preferred as the administrators do not have to deal with many false alarms. From a network point of view, fewer FN means fewer packets dropped assuming an attack flow. This behavioral property is of particular importance for a highly time-sensitive network like IoT, as it ensures timely delivery of time-sensitive data packets in the network. The FFCNN model performance metrics were calculated by substituting the values from Table 2 in Eq. 1 – 4. The performance metrics are compared to the existing machine learning models in the literature and are given in Table III.

From Table III, just by comparing the F1score, it is evident that FFCNN outperforms the existing models in the literature. Though accuracy is not a good performance metric for comparison for our case, it is evident that FFCNN outperforms other machine learning models in this metric as well. Analyzing the performance in much more detail, FFCNN outperforms the machine learning models in precision and recall performance metrics – 98 percent and 99.5 percent respectively. This implies that if a flow is classified as an attack by FFCNN, it is 98 percent likely that it is an attack flow. This in turn adds to the fact that FFCNN generates fewer false alarms, which is a desirable behavior for IDS. Comparing the precision of FFCNN to the machine learning algorithm that performs the best, the MLP, given that a flow is classified as an attack, it is only 95 percent likely that it is an attack flow. On a similar note, if FFCNN was given the task of classifying 100 attack flows, the scheme will be able to classify almost all 100 attack flows as attack. Comparing the recall of FFCNN to MLP, MLP will only be able to classify 94 attack flows as attack. The 6 attack packets will still pass through the IDS. Scaling things up to a real network, there is always a six percent chance that an attack packet is undetected by machine learning based IDS.

TABLE III. PERFORMANCE COMPARISON WITH EXISTING MACHINE LEARNING/ DEEP LEARNING MODELS IN LITERATURE

Machine Learning/ Deep Learning Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
J48	0.9068	0.6522	0.528	0.5836
Random Tree	0.9176	0.7283	0.5565	0.6309
REP Tree	0.9037	0.6417	0.5044	0.5648
Random Forest	0.9441	0.7833	0.8186	0.8005
SVM	0.931	0.92	0.93	0.93
MLP	0.9501	0.9546	0.9451	0.9498
FFCNN	***0.9941***	***0.9748***	***0.9954***	***0.9850***

To analyze the performance of FFCNN further, the Area Under the Curve of Receiver Operating Characteristic (ROC – AUC) is analyzed. ROC is a probability curve and is the measure of the ability of a model to differentiate different classes of the dataset. A ROC curve plots False Positive Rate vs True Positive Rate at different classification thresholds. Together, the ROC – AUC curve indicates the performance of a classification model at different classification thresholds. False Positive Rate is indicated in the x-axes and True Positive Rate is indicated in the y-axes. Lowering the classification threshold, more items are classified as positive increasing the number of FP and TN instances. The red diagonal in the ROC-AUC curve indicates the line where the False Positive Rate equals the True Positive Rate.

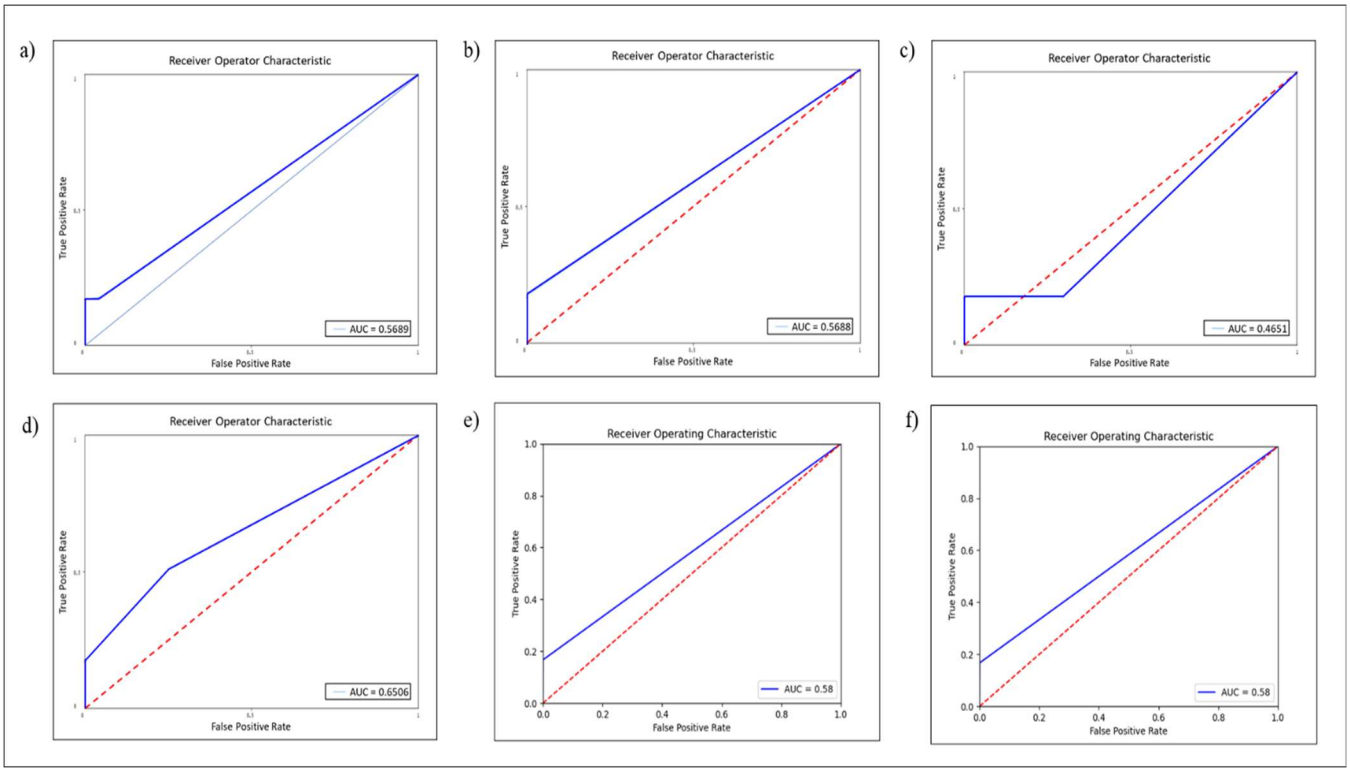

Fig. 2. ROC – AUC of the machine learning based classifiers: a) J48 b) Random Tree c) REP Tree d) Random Forest e) SVM f) MLP

Any point on this red diagonal indicates that the number of instances of FP equals the number of instances of TN. A classifier with a high True Positive Rate for a low False Positive Rate is preferred. In practice, the threshold is set based on the acceptable number of false positives. With regards to AUC, the classifier perfectly distinguishes both the classes in the dataset for an AUC of 1. If the AUC is between 0.5 and 1 the probability that the classifier classifies both the classes are high, but it is not perfect. An AUC is 0.5 indicates that the classifier is unable to differentiate between classes and is behaving randomly.

Fig. 2 shows the ROC-AUC curve for the machine learning models in the literature. The ROC-AUC curve of the FFCNN scheme is shown in Fig. 3. The AUC of the FFCNN scheme is 0.99 indicating good classification performance i.e., the classifier can finely distinguish between the attack and benign flows. The ROC plot also shows a very high True Positive Rate for a very low False Positive Rate which is highly desirable. In comparison to the ROC-AUC of machine learning models in Fig. 3, it is evident that the ROC-AUC of the machine learning models ranges between 0.5 to 0.6 indicating poor classification performance for lower thresholds of False Positive Rate. Comparing the performance of FFCNN to the best performing machine learning algorithm in terms of F1 score, MLP performs close to FFCNN, but has a significantly low AUC-ROC. This implies that the MLP is performing well. But to achieve the best True Positive Rate, a high False Positive Rate threshold must be chosen which leads to many false alarms by the IDS which is undesirable.

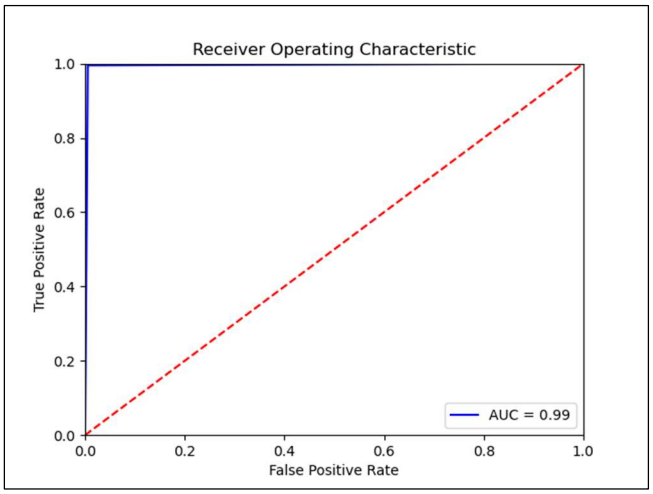

Fig. 3. ROC – AUC for the FFCNN scheme

IV. Conclusion

A novel deep learning model, FFCNN scheme, to detect LR DoS attacks in an IoT – SDN environment is proposed in this research work. The FFCNN scheme was developed using FFNN and CNN as its building blocks. From the analysis of the training performance, it was evident that a good, generalised model that did not overfit the training dataset was achieved. The performance analysis using the test dataset showed that the FFCNN scheme outperforms the existing machine learning based LR DoS detection models on all four performance metrics. The analysis of the ROC-AUC implied that the FFCNN scheme can finely distinguish between attack and benign flows. Due to the presence of both high rate and low rate DoS samples in the training dataset, it can be concluded that the FFCNN scheme performs the best with regards to detecting DoS attacks in an IoT-SDN environment in general.

ACKNOWLEDGMENT

This research is supported by A*STAR under its RIE2020 Advanced Manufacturing and Engineering (AME) Industry Alignment Fund – Pre Positioning (IAF-PP) (Award A19D6a0053). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of A*STAR.

REFERENCES

[1] K. Ashton, "That@ Internet of Thing“ Thing," 1999.

[2] M. S. ElSayed, N.-A. Le-Khac, M. A. Albahar, and A. Jurcut, "A novel hybrid model for intrusion detection systems in SDNs based on CNN and a new regularization technique," Journal of Network and Computer Applications, vol. 191, p. 103160, 2021/10/01/ 2021, doi: 10.1016/j.jnca.2021.103160.

[3] A. Abubakar and B. Pranggono, "Machine learning based intrusion detection system for software defined networks," in 2017 Seventh International Conference on Emerging Security Technologies (EST), 6-8 Sept. 2017 2017, pp. 138-143, doi: 10.1109/EST.2017.8090413.

[4] M. M. Isa and L. Mhamdi, "Native SDN Intrusion Detection using Machine Learning," in 2020 IEEE Eighth International Conference on Communications and Networking (ComNet), 27-30 Oct. 2020 2020, pp. 1-7, doi: 10.1109/ComNet47917.2020.9306093.

[5] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep Recurrent Neural Network for Intrusion Detection in SDN-based Networks," in 2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), 25-29 June 2018 2018, pp. 202-206, doi: 10.1109/NETSOFT.2018.8460090.

[6] Z. Wang, Y. Liu, D. He, and S. Chan, "Intrusion detection methods based on integrated deep learning model," Computers & Security, vol. 103, p. 102177, 2021/04/01/ 2021, doi: 10.1016/j.cose.2021.102177.

[7] Santos, D. Souza, W. Santo, A. Ribeiro, and E. Moreno, "Machine learning algorithms to detect DDoS attacks in SDN," Concurrency and Computation: Practice and Experience, vol. 32, no. 16, p. e5402, 2020.

[8] J. A. Pérez-Díaz, I. A. Valdovinos, K. K. R. Choo, and D. Zhu, "A Flexible SDN-Based Architecture for Identifying and Mitigating Low-Rate DDoS Attacks Using Machine Learning," IEEE Access, vol. 8, pp. 155859-155872, 2020, doi: 10.1109/ACCESS.2020.3019330.

[9] H. H. Jazi, H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling," Computer Networks, vol. 121, pp. 25-36, 2017/07/05/ 2017, doi: 10.1016/j.comnet.2017.03.018.

[10] I. Sharafaldin, A. H. Lashkari, and A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization," in ICISSP, 2018, doi: 10.5220/0006639801080116.

[11] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," Computers & Security, vol. 31, no. 3, pp. 357-374, 2012/05/01/ 2012, doi: 10.1016/j.cose.2011.12.012.

[12] G. V. Fabian Pedregosa, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay, "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, 85, pp. 2825-2830, 2011.