IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 June 2022, accepted 12 August 2022, date of publication 19 August 2022, date of current version 26 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3200477

RESEARCH ARTICLE

ShieldRNN: A Distributed Flow-Based DDoS Detection Solution for IoT Using Sequence Majority Voting

FARIS ALASMARY[1], SULAIMAN ALRADDADI[2], SAAD AL-AHMADI[1], (Senior Member, IEEE), AND JALAL AL-MUHTADI[1]

[1]Department of Computer Science, King Saud University, Riyadh 11451, Saudi Arabia
[2]Department of Computer Science and Engineering, Yanbu University College, Yanbu 46435, Saudi Arabia

Corresponding author: Faris Alasmary (faris.alasmary@gmail.com)

This work was funded and supported by the Deanship of Scientific Research at King Saud University through the initiative of the DSR Graduate Students Research Support (GSR).

ABSTRACT The Distributed Denial of Service (DDoS) attack is considered one of the most critical threats on the Internet, blocking legitimate users from accessing online services. Botnets have exploited insecure IoT devices and used them to launch DDoS attacks. Providing IoT devices with the ability to detect DDoS attacks will prevent them from becoming contributors to these attacks. This paper presents an efficient solution to defend IoT devices against such inevitable attacks. The proposed solution consists of two parts: an IoT node detector and a server detector. The IoT node detector is a lightweight classifier to monitor egress traffic. The server detector is a more accurate classifier that is used by the IoT node if it suspected itself to be a contributor to a DDoS attack. To develop an accurate server detector, this paper proposes *ShieldRNN*: a novel training and prediction approach for RNN/LSTM models. We compare *ShieldRNN* with other supervised and unsupervised models on the CIC-IDS2017 dataset and show that it outperforms them. Also, we set baseline results for DDoS detection on the CIC IoT 2022 dataset.

INDEX TERMS Distributed denial of service (DDoS), anomaly detection, intrusion detection system (IDS), machine learning, majority voting.

I. INTRODUCTION

The Denial of Service (DoS) attack is one of the most dangerous threats an organization may face. The attack is defined as an attempt to overload the capabilities of the victim's machine and makes it unavailable to other legitimate users and devices. In this type of attacks, the attacker uses a single machine to launch the attack. Another variant of this attack is called the Distributed Denial of Service (DDoS) attack which involves multiple machines, that are controlled by the attacker, to launch the attack at the same time on the victim's machine [1]. One of the most popular DDoS attacks was the Mirai Attack that happened in October 2016 when many popular websites were affected including: Twitter, Netflix, Amazon, and Github [2]. This attack was done using hundreds of thousands of Internet of Things (IoT) devices [3] that were infected by the Mirai botnet. A botnet is defined as a network of infected machines called *zombies* that are controlled by a *botmaster* [1]. Figure 1 shows the botnet architecture in general. Experts nowadays consider IoT botnets as the new norm of DDoS attacks [3]. There has been different techniques proposed to detect DDoS attacks that can be categorized as: *Payload-based* techniques and *Flow-based* techniques. A flow can be defined as the stream of packets that have common network characteristics such as the network protocol, source IP and destination port [4]. The flow-based methods inspect the packet header only while the payload-based methods analyze the information inside the packet. The flow-based analysis is not as accurate as the payload-based analysis since it inspects the packet header and it cannot detect the hidden attacks inside the packet payload [5], [6]. In DDoS attacks, the number of

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Da Lin.

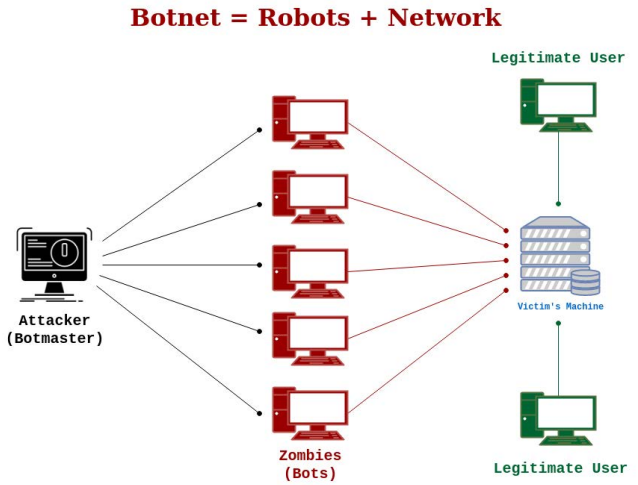


FIGURE 1. This diagram shows a DDoS attack using a botnet.

received packets at the victim's machine is huge and the packets have source IP addresses that are nearly random [4]. Moreover, the timing between received packets can play an important role in detecting incoming or outgoing DDoS attacks [2]. In this paper, we focus on DDoS attacks detection based on the traffic flow.

II. RELATED WORKS

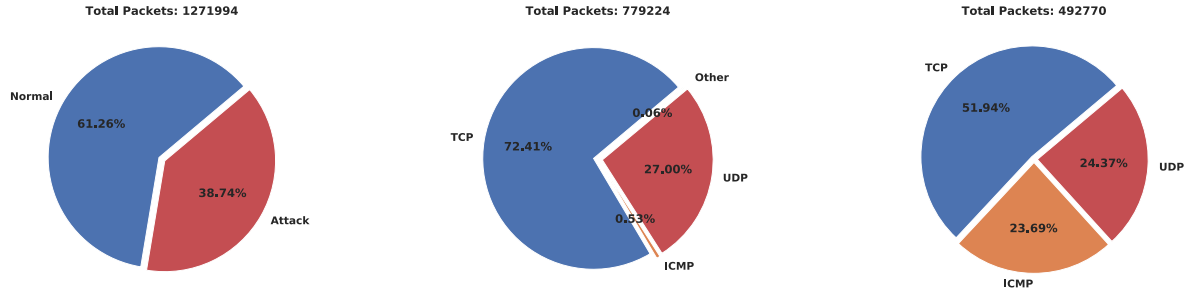
Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) can be classified mainly into *rule-based* and *machine-learning-based* systems. Most of the available detection systems rely on the signature of the attacks. These systems match incoming network traffic with a predefined set of rules to detect attack patterns [7]. One of the most popular rule-based IDSs is called Snort that was developed in 1998 by Martin Roesch [8]. Another well-known rule-based system is called Suricata [9]. Shah *et al.* [10] investigated the performance of Snort and Suricate in terms of utilization of computational resources as well as the processing efficiency. They showed that Snort is lighter, i.e., it utilizes less computational resources compared to Suricate. On the other hand, Suricate achieved a higher processing rate, with slightly higher memory usage, which was more than 82K packets/second compared to Snort's rate that was about 60K packets/second. The memory usage of both systems was more than 3 GBytes which indicates that they are not suitable to be implemented in low-resource IoT devices. Zitta *et al.* [11] implemented Suricate on Raspberry Pi 3 with some rules to detect port scanning activities but there was no data provided regarding the required processing time and the hardware utilization. Researchers in [12] proposed early DDoS detection in Software-Defined Networking (SDN) using Snort.

The machine-learning-based approaches include classical machine-learning algorithms such as Support Vector Machine (SVM), Artificial Neural Network (ANN), and Naïve Bayes as well as deep-learning algorithms such as Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long-Short Term Memory (LSTM). Machine-learning-based algorithms can guarantee the detection of known attacks and similar unknown attacks even if they were not seen before during the training phase [7]. Shafiq *et al.* [13] explored five well-known machine learning algorithms and they developed a framework for automatic effective algorithm selection. They found that the Naïve Bayes classifier achieved the best performance in anomaly and intrusion detection in IoT network. Doshi *et al.* [2] tested five different machine learning algorithms after they collected their own dataset and hand-engineered a set of useful features based on the traffic flow. Another group of researchers at the University of Chinese Academy of Sciences [14] proposed a solution based on the Random Forest algorithm using the NetFlow data to detect DDoS attacks. Hodo *et al.* [15] developed an ANN model for DDoS detection on a simulated IoT network. Khater *et al.* [16] implemented lightweight intrusion detection using the ANN algorithm on a Raspberry Pi 3 and reported the energy consumption and CPU utilization while running their system. Using deep learning algorithms, the papers [17], [18], [19] showed that the RNN algorithm can improve DDoS detection compared to the classical machine learning algorithms. The CNN algorithm has shown that it can be applied for anomaly detection tasks such as DDoS attacks [20], [21], [22].

Different from the above works, we propose a machine-learning-based system to be implemented in two stages: an IoT node detector and a server detector. The IoT node detector is a lightweight classifier to monitor egress traffic. The server detector is an accurate deep learning classifier that is utilized by the IoT node if it detected or suspected that it is attempting a DDoS attack. Our proposed system aims to provide a DDoS detection functionality on the IoT node and minimizes the workload of processing all traffic from all connected IoT nodes. It helps IoT nodes to recognize if they are zombies (bots) of a botnet DDoS attack by providing them with intelligent decision making capabilities. Figure 3 shows the proposed system architecture. This system architecture can be beneficial for organizations that have multiple IoT devices connected to the Internet to analyze the traffic of each IoT node inside the node itself instead of analyzing the whole traffic in a central server. For example, it can be used by an organization that has surveillance cameras which are connected to the Internet where each camera analyzes its own traffic to detect abnormal activities and sends the suspected packets to the accurate detector server to get the final decision.

III. EXPERIMENT

We setup our experiment to perform and detect four well-known DDoS attacks: TCP SYN flood, UDP flood, TCP PSH+ACK flood, and ICMP flood [23]. We developed a tool to launch these attacks randomly with a random number of packets per attack, random source and destination ports, and a randomly generated (spoofed) source IP address using



(a) Normal / Attack Packets (b) Normal Packets per Protocol (c) Attack Packets per Protocol

FIGURE 2. The distribution of packets in our dataset.

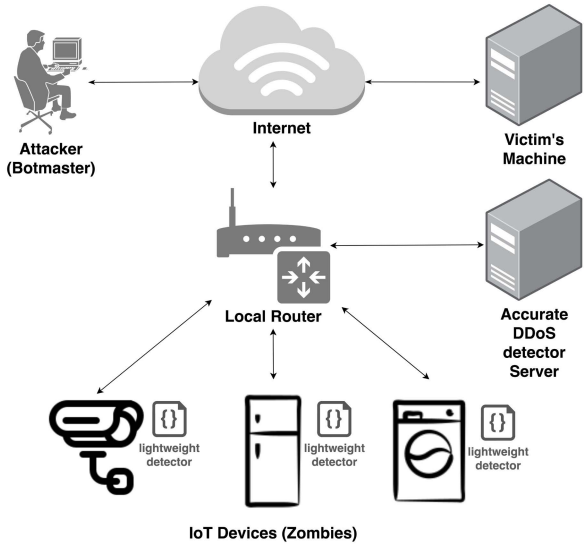


FIGURE 3. The proposed system architecture. At the beginning, the DDoS worm compromises the IoT device. Then the attacker sends a command to the worm to launch the DDoS attack. After that, the lightweight classifier in the IoT node detects an abnormal behaviour and the IoT device sends the suspected packets to the *ShieldRNN* server. Finally, the *ShieldRNN* server analyzes the suspected packets and sends the final decision to the IoT node.

TABLE 1. Features extracted from each packet.

Feature Name	Description
frame.len	Frame length on the wire
ip.hdr_len	IP header length
ip.len	IP total length
ip.flags.rb	IP reserved bit
ip.flags.df	IP don't fragment bit
ip.flags.mf	IP more fragments bit
ip.frag_offset	IP fragment offset
ip.ttl	IP time to live
tcp.srcport	TCP/UDP source port
tcp.dstport	TCP/UDP destination port
tcp.len	TCP/UDP segment length
tcp.ack	TCP acknowledgment number
tcp.flags.res	TCP reserved bit
tcp.flags.ns	TCP nonce bit
tcp.flags.cwr	TCP congestion window reduced bit
tcp.flags.ecn	TCP ECN-Echo bit
tcp.flags.urg	TCP urgent bit
tcp.flags.ack	TCP acknowledgment bit
tcp.flags.push	TCP push bit
tcp.flags.reset	TCP reset bit
tcp.flags.syn	TCP SYN bit
tcp.flags.fin	TCP FIN bit
tcp.window_size	TCP calculated window size
tcp.time_delta	Time since previous frame in this TCP stream
flow_speed	The time difference between two consecutive packets
protocol	Highest layer protocol

Scapy[1] library for packet manipulation. The source code of the implementation of this paper is available on Github.[2]

A. DATA COLLECTION

To collect data, we started packet sniffing using Wireshark[3] for nearly 24 minutes to collect normal traffic including web browsing, YouTube watching, and WhatsApp chatting in the web version. At the beginning, we collected 65,314 packets of normal traffic. After that, we launched random DoS attacks with random number of packets per attack on a victim server using our own Python script. We collected a mix of normal and attack packets for more than 102 minutes and we stopped the attack script. Finally, we collected about 17 minutes of normal traffic again and stopped Wireshark traffic sniffing. Figure 2 shows the distribution of packets in the collected dataset.

B. FEATURE EXTRACTION AND PACKET LABELLING

We used a combination of the features mentioned in [2] and [24]. The features were extracted from the frame header, IP packet header, TCP segment header, and UDP datagram header. Table 1 shows the extracted features from each packet. In the case of UDP packets, the features *tcp.srcport*, *tcp.dstport*, and *tcp.len* are replaced with those of the UDP header information. We selected the top 5 most frequent protocols from the feature *protocol* and one-hot encoded them as "is_TCP", "is_UDP", "is_SSL", "is_ICMP", and "is_DNS". For other protocols, we simply encoded them as "is_OTHER" [2].

[1]https://github.com/secdev/scapy
[2]https://github.com/farisalasmary/shieldrnn
[3]https://www.wireshark.org

For packets' labelling, a packet is considered as an *attack* packet if its destination IP address was the IP address of the victim's server, otherwise, it is considered as a *normal* packet.

C. DATA PREPROCESSING

As the first task, we kept the dataset in the same order in which it was captured. Then, the whole 2D dataset matrix was converted into a 3D tensor such that each consecutive *seq_len* packets are considered as a single example that would be fed into a sequence model like RNN and LSTM. For example, if the $seq\_len = 10$, it means that every 10 consecutive packets form a single training example that will be fed to the sequence model and each packet represents the input vector for a specific time step t. The final 3D tensor size is $num\_examples \times seq\_len \times num\_features$ instead of the original 2D matrix size $total\_num\_packets \times num\_features$ where *num_examples* is the number of examples, *seq_len* is the sequence length, i.e., the number of packets per example, *num_features* is the number of features which represents the input size of the model, and $total\_num\_packets = num\_examples \times seq\_len$ which is the total number of packets in the dataset. Before splitting the dataset into a training set and a testing set, we needed to ensure that the following two conditions are satisfied:

1) The packets must be in the same order as they were captured to allow the sequence model to learn the correct sequence.
2) Shuffling the data before splitting it into a training set and a testing set so that different normal scenarios and different attack scenarios can be seen in both sets.

These conditions contradict each other since we can't preserve the right order while we need to randomly shuffle the data before splitting it. To solve this problem, we developed an improved version of the train_test_split algorithm that takes different *seq_len* values, e.g. 5, 10, 20, 50, 100, 250, 500, 1000 as in our experiments, converts the 2D data matrix into a 3D tensor with the largest *seq_len* in the provided list, shuffle and split the data into training and testing data, and convert it back into a 2D matrix to use the training set in feature normalization and feature selection. The idea behind splitting data into training and testing datasets *after* we created the 3D tensor is that we wanted to preserve the right order in which the packets were captured and to shuffle by examples instead of shuffling by packets. Selecting the largest *seq_len* to convert the 2D matrix into a 3D tensor before data splitting ensures the minimal error in the number of examples that contains out of order packets. This will be useful later when using smaller values of *seq_len* during training. Algorithm 1 and Figure 4 explain these steps in details.

We split the dataset into a training set (90%) and a testing set (10%). We chose this data split ratio since the data we used has a large number of examples. Therefore, we want to use most of the data for training but not by affecting the testing phase which will still have a large number of examples to test the model on. After data splitting, we normalized the training set *feature-wise* across all packets in the training set to have a zero-mean and a unit-variance. One-hot encoded features and bit features, e.g. TCP flags bits, were not normalized. Feature normalization was crucial for the models during training to converge. We applied the same normalization process on the testing set using means and variances that were calculated using the training set. In production environment, we propose to use exponentially weighted moving averaging (EWMA) and variance (EWMV) [25] estimation to cope with any changes in DDoS attacks trends using equation 1 and equation 2:

$$A_t = \alpha x_t + (1-\alpha)A_{t-1} \tag{1}$$

$$V_t = \beta(x_t - A_t)^2 + (1-\beta)V_{t-1} \tag{2}$$

where x_t is the current packet, A_{t-1} and V_{t-1} are the previous EWMA and EWMV, respectively. α and β are controllable parameters that specify how much we depend on the previous EWMA, EWMV, and the current packet x_t to calculate the current EWMA (A_t) and EWMV (V_t), respectively. We set A_0 to be the average and V_0 to be the variance which were calculated on the training set.

Algorithm 1 improved_train_test_split

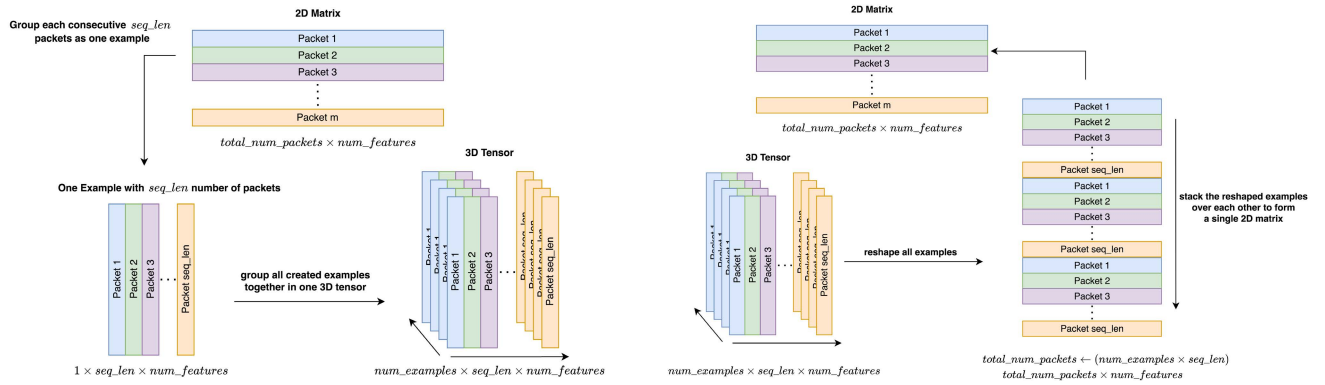
Input: S: a list of predefined sequence lengths.
X: a matrix of size $m \times n$ contains the packets.
y: a vector of size $m \times 1$ which contains the labels of each packet.
α: a scalar between 0 and 1 represents the percentage of the training dataset.
Where m is the number of packets and n is the number of features.

Output: X_{train}: a matrix of size $p \times n$.
y_{train}: a vector of size $p \times 1$.
X_{test}: a matrix of size $q \times n$.
y_{test}: a vector of size $q \times 1$.
Where $p \leftarrow floor(\alpha \times m)$ and $q \leftarrow (m - p)$

$T_{longest} \leftarrow MAX(S)$
$X_{3d},\ y_{3d} \leftarrow matrix\_to\_tensor(X, y, T_{longest})$
$X_{3d\_train},\ y_{3d\_train},\ X_{3d\_test},\ y_{3d\_test} \leftarrow train\_test\_split(X_{3d}, y_{3d}, \alpha)$
$X_{train},\ y_{train} \leftarrow tensor\_to\_matrix(X_{3d\_train},\ y_{3d\_train},\ T_{longest})$
$X_{test},\ y_{test} \leftarrow tensor\_to\_matrix(X_{3d\_test},\ y_{3d\_test},\ T_{longest})$
return $X_{train},\ y_{train},\ X_{test},\ y_{test}$

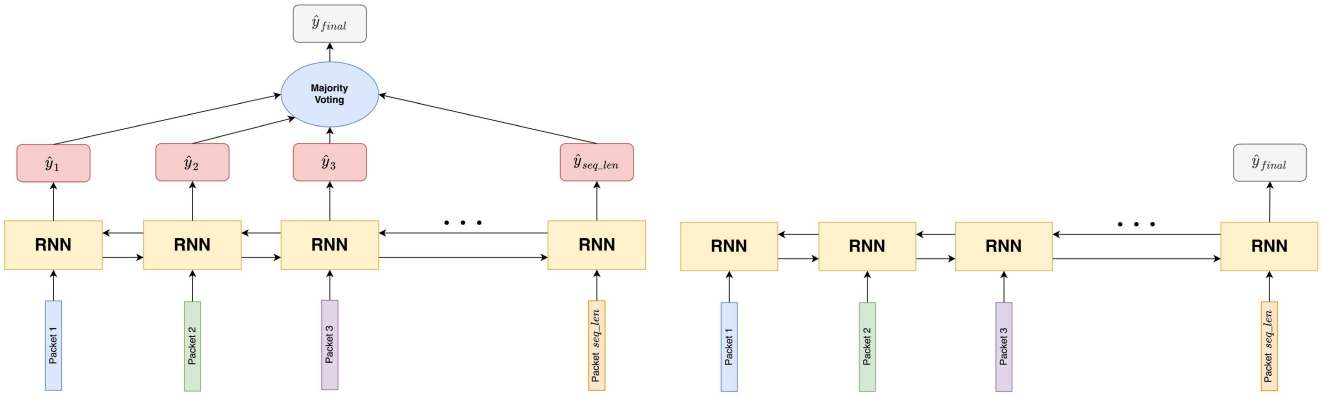
D. FEATURE SELECTION

We trained a logistic regression classifier with LASSO regularization and applied a grid search with 5-fold cross-validation to get the best estimate for the regularization coefficient. Then, we trained another classifier with the best estimated regularization coefficient and extracted the selected features by the algorithm. The best estimated regularization



(a) Matrix-to-Tensor (b) Tensor-to-Matrix

FIGURE 4. The process of converting the dataset from a 2D matrix into a 3D tensor of sequences of packets and vice versa.



(a) Majority Voting Prediction (b) Last Output Prediction

FIGURE 5. The prediction techniques that were explored in this paper.

coefficient was 10^3. Figures 8a and 8b in Appendix A show the selected features and the discarded features by LASSO, respectively.

E. TRAINING THE LIGHTWEIGHT DETECTOR FOR THE IoT NODE

We trained 12 classifiers: 4 ANNs with a single hidden layer with sizes: [3, 5, 10, 20] neurons, 1 Logistic Regression (LR) classifier, 3 Random Forest classifiers with different numbers of trees: [3, 5, 10], 3 SVM classifiers with kernels: [linear, RBF, Polynomial with the third degree], and 1 Naïve Bayes classifier. Random Forest classifiers achieved the best results compared to other classifiers when they were evaluated on the testing set. Models' results are shown in Figure 6.

F. TRAINING THE ACCURATE DETECTOR FOR THE SERVER

We trained multiple RNN and LSTM models using two techniques that we developed for training and prediction. The first technique is to train the sequence model to predict a label for each packet in the training example, i.e., a sequence-to-sequence training. The prediction for this technique is done as a majority voting, i.e., if most of the sequence packets are

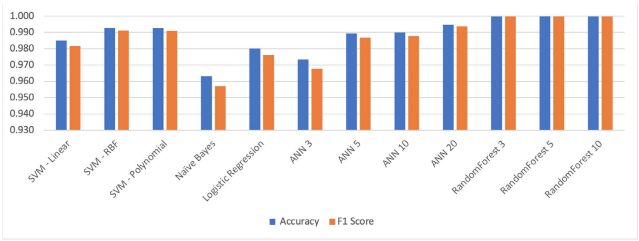


FIGURE 6. Comparison between different classifiers to choose the IoT node detector.

predicted as *attack* packets, then the final prediction of the whole sequence is *attack*, otherwise, the final prediction is *normal*. The training procedure is explained in Algorithm 5. The majority voting prediction is detailed in Algorithm 4 and shown in Figure 5a.

The second technique is to train the sequence model to predict the last output of a given training example as *attack* if the majority of its ground-truth labels are *attack* labels. The idea is to compute the majority of the labels in the ground-truth and force the model to learn to output the label of the last output of the given sequence as the label of the

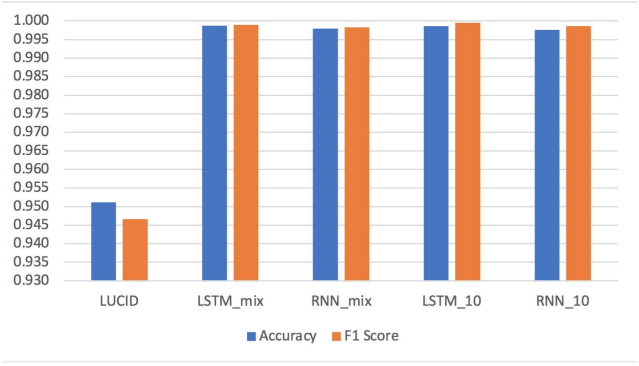


FIGURE 7. Our models vs. LUCID.

Algorithm 2 Train_RNN_Mix_Seq_lens

Input: M: a vanilla RNN model or its variants.
S: a list of predefined sequence lengths.
X: a matrix of size $m \times n$.
y: a vector of size $m \times 1$ which contains the labels of each packet.
epochs: the number of epochs to train.
training_mode: the training algorithm to use: "*train_seq_to_seq*" or "*train_last_output*".
where m is the number of packets and n is the number of features.
Output: M: a trained model

$S' \leftarrow S.copy()$
for *epoch* ***in*** *epochs* **do**
  $T \leftarrow random.choice(S')$
  $X' \leftarrow X.copy()$
  $y' \leftarrow y.copy()$
  $X_{3d},\ y_{3d} \leftarrow matrix\_to\_tensor(X',\ y',\ T)$
  if *training_mode* = "*train_seq_to_seq*" **then**
    Train_Seq_to_Seq($M, X_{3d},\ y_{3d}$)
  else
    Train_Last_Output($M, X_{3d},\ y_{3d}$)
  end
  $S'.remove(T)$
  if $S'.is\_empty()$ **then**
    $S' \leftarrow S.copy()$
  end
end
return M

Algorithm 3 Predict_Last_output

Input: M: a pretrained vanilla RNN model or its variants
X: a matrix of size $T \times n$ represents a single example.
T: a sequence length that is considered when converting matrix into tensor
where n is the number of features per packet
Output: $\hat{y}_{final}$: the prediction of the trained model: attack or normal

$X_{3d} \leftarrow matrix\_to\_tensor(X, T)$
$\hat{y}_{3d} \leftarrow M(X_{3d})$
$\hat{y}_{final} \leftarrow \hat{y}_{3d}[T]$ `// take the last label`
return $\hat{y}_{final}$

Algorithm 4 Predict_Majority_Voting

Input: M: a pretrained vanilla RNN model or its variants.
X: a matrix of size $T \times n$ represents a single example.
T: a sequence length that is considered when converting matrix into tensor.
where n is the number of features per packet.
Output: $\hat{y}_{final}$: the prediction of the trained model: attack or normal.

$X_{3d} \leftarrow matrix\_to\_tensor(X, T)$
$\hat{y} \leftarrow M(X_{3d})$
$count \leftarrow 0$
for $t = 1$ ***to*** T **do**
  if $\hat{y}_t =$ "*attack*" **then**
    $count \leftarrow count + 1$
  end
end
`// 50% or more is considered the majority`
if $(count\ /\ T) \geq 0.5$ **then**
  $\hat{y}_{final} \leftarrow$ "*attack*"
else
  $\hat{y}_{final} \leftarrow$ "*normal*"
end
return $\hat{y}_{final}$

majority. This technique is shown in Algorithms 3 and 6, and in Figure 5b. We trained both LSTM and RNN models for different values of *seq_len* to see how the sequence length affects the accuracy of prediction. In our experiments, we've set *seq_len* to values: 5, 10, 20, 50, 100, 250, 500, 1000 and trained RNN and LSTM models for each sequence length. Moreover, we developed Algorithm 2 to train RNN and LSTM models on a mix of sequence lengths. In the case of training on a single sequence length, we simply provide the algorithm with a set of sequence lengths S that contains only that single sequence length.

Figure 9 in Appendix B shows the F1-scores calculated for 18 RNN/LSTM models where the suffix "*_XXX*" represents the sequence length that the model was trained on it. If the suffix is "*_mix*", it means that the model was trained on a mix of different sequence lengths as described by Algorithm 2. We tested these models on a randomly generated list of sequence lengths and we found that the models trained using the proposed algorithm can generalize better for other

TABLE 2. The values of true negatives (TN), false positives (FP), false negatives (FN), and true positives (TP) when we evaluated our models on the testing set of CIC-IDS2017 - Friday [26]. The suffix "_mix_MV" in the model's name means that the model was trained using the algorithms 2, 4, and 5 where "MV" stands for Majority Voting and "mix" stands for the training using a mix of sequence lengths. The suffix "_T" indicates that the model evaluation is done on the list of training sequence lengths and the suffix "_R" indicates that the evaluation is done on a list of randomly generated sequence lengths.

Evaluation on The List of Training Sequence Lengths									Evaluation on The List of Random Sequence Lengths								
	LSTM_mix_MV_T				RNN_mix_MV_T					LSTM_mix_MV_R				RNN_mix_MV_R			
seq_len	TN	FP	FN	TP	TN	FP	FN	TP	seq_len	TN	FP	FN	TP	TN	FP	FN	TP
5	182280	0	0	17720	182241	39	51	17669	**26**	34532	0	1	3928	34516	16	5	3924
10	89642	0	0	10358	89609	33	18	10340	**57**	15887	0	0	1656	15880	7	1	1655
20	44788	0	0	5212	44770	18	5	5207	**77**	11779	0	0	1208	11776	3	2	1206
50	17905	0	0	2095	17894	11	1	2094	**212**	4284	0	1	431	4282	2	3	429
100	8950	0	0	1050	8946	4	0	1050	**329**	2794	0	0	245	2792	2	0	245
250	3586	0	0	414	3585	1	0	414	**597**	1571	0	0	104	1571	0	0	104
500	1791	0	0	209	1791	0	0	209	**643**	1463	0	0	92	1463	0	0	92
1000	895	0	0	105	895	0	0	105	**877**	1085	0	0	55	1085	0	0	55
Total	**349837**	**0**	**0**	**37163**	**349731**	**106**	**75**	**37088**	**Total**	**73395**	**0**	**2**	**7719**	**73365**	**30**	**11**	**7710**

Algorithm 5 Train_Seq_to_Seq

Input: M: a vanilla RNN model or its variants
X_{3d}: a tensor of size $m \times T \times n$
y_{3d}: the labels for each packet in X_{3d} with size $m \times T \times 1$
where m is the batch size, T is the number of packets per example (the sequence length), and n is the number of features
Output: M: the model trained for a single epoch

$loss_{total} \leftarrow 0$
for $i = 1$ ***to*** m **do**
  $\hat{y}_{3d} \leftarrow M(X_{3d}[i])$
  $loss_{sum} \leftarrow 0$
  for $t = 1$ ***to*** T **do**
    $y_t \leftarrow y_{3d}[i][t]$
    $\hat{y}_t \leftarrow \hat{y}_{3d}[t]$
    $loss \leftarrow MSE(y_t, \hat{y}_t)$
    $loss_{sum} \leftarrow loss_{sum} + loss$
  end
  $loss_{avg} \leftarrow loss_{sum} \ / \ T$
  $loss_{total} \leftarrow loss_{total} + loss_{sum}$
end
$loss_{total\_avg} \leftarrow loss_{total} \ / \ m$
$grads \leftarrow compute\_gradient(M, loss_{total\_avg})$
$M \leftarrow update\_weights(M, grads)$
return M

Algorithm 6 Train_Last_Output

Input: M: a vanilla RNN model or its variants
X_{3d}: a tensor of size $m \times T \times n$
y_{3d}: the labels for each packet in X_{3d} with size $m \times T \times 1$
where m is the batch size, T is the number of packets per example (the sequence length), and n is the number of features
Output: M: the model trained for a single epoch

$loss_{total} \leftarrow 0$
for $i = 1$ ***to*** m **do**
  $\hat{y}_{3d} \leftarrow M(X_{3d}[i])$
  $\hat{y}_{last} \leftarrow \hat{y}_{3d}[T]$ `// take the last label`
  $count \leftarrow 0$
  for $t = 1$ ***to*** T **do**
    $y_t \leftarrow y_{3d}[i][t]$
    if $y_t =$ "$attack$" **then**
      $count \leftarrow count + 1$
    end
  end
  `// 50% or more is the majority`
  if $(count \ / \ T) \geq 0.5$ **then**
    $y_{true} \leftarrow$ "$attack$"
  else
    $y_{true} \leftarrow$ "$normal$"
  end
  $loss \leftarrow MSE(y_{true}, \hat{y}_{last})$
  $loss_{total} \leftarrow loss_{total} + loss$
end
$loss_{total\_avg} \leftarrow loss_{total} \ / \ m$
$grads \leftarrow compute\_gradient(M, loss_{total\_avg})$
$M \leftarrow update\_weights(M, grads)$
return M

sequence lengths even if they were not trained on them. Appendix C shows the F1-scores of the models.

We found that our proposed training and prediction algorithms 2, 4, and 5 achieved the best F1-score where it achieved 100% correct predictions for all examples in the randomly generated list of sequence lengths: 77, 329, 597, 643, 877. Hyper parameters of all models were fixed. Each model had a single bidirectional hidden layer of size 20 that was trained for 3000 epochs with a batch size 64, a learning rate of 0.001, and a dropout of 50%. We compared the performance of our algorithms with LUCID [22]. We trained LUCID on our dataset with time window = 10 and packet flow length = 10. We found that our models achieved higher

TABLE 3. The values of true negatives (TN), false positives (FP), false negatives (FN), and true positives (TP) when we evaluated our models on the testing set of CIC-IoT2022 [27].

Evaluation on The List of Training Sequence Lengths									Evaluation on The List of Random Sequence Lengths								
	LSTM_mix_MV_T				RNN_mix_MV_T					LSTM_mix_MV_R				RNN_mix_MV_R			
seq_len	TN	FP	FN	TP	TN	FP	FN	TP	seq_len	TN	FP	FN	TP	TN	FP	FN	TP
5	550550	148	57	560245	550380	318	55	560247	**26**	104993	9	310	108341	104985	17	198	108453
10	273282	15	15	282188	273273	24	14	282189	**57**	47766	2	220	49468	47761	7	124	49564
20	136546	4	2	141198	136545	5	10	141190	**77**	35296	2	182	36662	35295	3	93	36751
50	54082	0	0	57018	54081	1	5	57013	**212**	12653	0	65	13484	12651	2	34	13515
100	26847	0	1	28702	26846	1	2	28701	**329**	8166	0	65	8653	8165	1	27	8691
250	10670	0	0	11550	10669	1	1	11549	**597**	4525	0	35	4744	4525	0	22	4757
500	5290	0	0	5820	5289	1	0	5820	**643**	4227	0	27	4385	4226	1	12	4400
1000	2632	1	0	2922	2632	1	1	2921	**877**	3106	0	26	3202	3105	1	14	3214
Total	**1059899**	**168**	**75**	**1089643**	**1059715**	**352**	**88**	**1089630**	**Total**	**220732**	**13**	**930**	**228939**	**220713**	**32**	**524**	**229345**

TABLE 4. The evaluation results of *ShieldRNN* and the state-of-the-art models on the CIC-IDS2017-Wednesday [26] dataset.

Model	Accuracy	F1-Score	Precision	Recall
LSTM_mix_MV_T	**99.998%**	**99.992%**	**99.983%**	**100.00%**
Voting [28]	99.89%	N/A	N/A	N/A
Autoencoder [29]	93.30%	N/A	86.40%	N/A
GRU-RNN [30]	N/A	99.00%	99.00%	99.00%
Tree-CNN (SRS) [31]	99.00%	N/A	N/A	N/A

TABLE 5. The evaluation results of different settings of *ShieldRNN* compared to the state-of-the-art models on the CIC-IDS2017-Friday [26] dataset.

Model	Accuracy	F1-Score	Precision	Recall
LSTM_mix_MV_T	**100.00%**	**100.00%**	**100.00%**	**100.00%**
RNN_mix_MV_T	99.953%	99.757%	99.715%	99.798%
LSTM_mix_MV_R	99.998%	99.987%	100.00%	99.974%
RNN_mix_MV_R	99.949%	99.735%	99.612%	99.858%
LUCID [22]	99.67%	99.66%	99.39%	99.94%
CNN+LSTM [32]	97.16%	98.25%	97.41%	99.10%
DeepGFL [33]	N/A	43.21%	75.67%	30.24%
LSTM+DBN [34]	99.815%	N/A	N/A	N/A
LSTM [35]	N/A	97.76%	96.43%	99.23%

scores compared to LUCID when we tested them on the test set. Figure 7 shows the results of our models compared with LUCID [22].

IV. STATE-OF-THE-ART COMPARISON

For a fair comparison between our solution and the state-of-the-art, we focus our comparison on solutions that used the CIC-IDS2017 [26] dataset. We prepared the data and extracted the features as we described in previous sections. Then, we trained different LSTM and RNN models using the algorithms: improved_train_test_split (Algorithm 1), Train_Seq_to_Seq (Algorithm 5), Predict_Majority_Voting (Algorithm 4), and Train_RNN_Mix_Seq_lens (Algorithm 2) with similar settings as described in the accurate classifier training section III-F. We used the traffic trace of

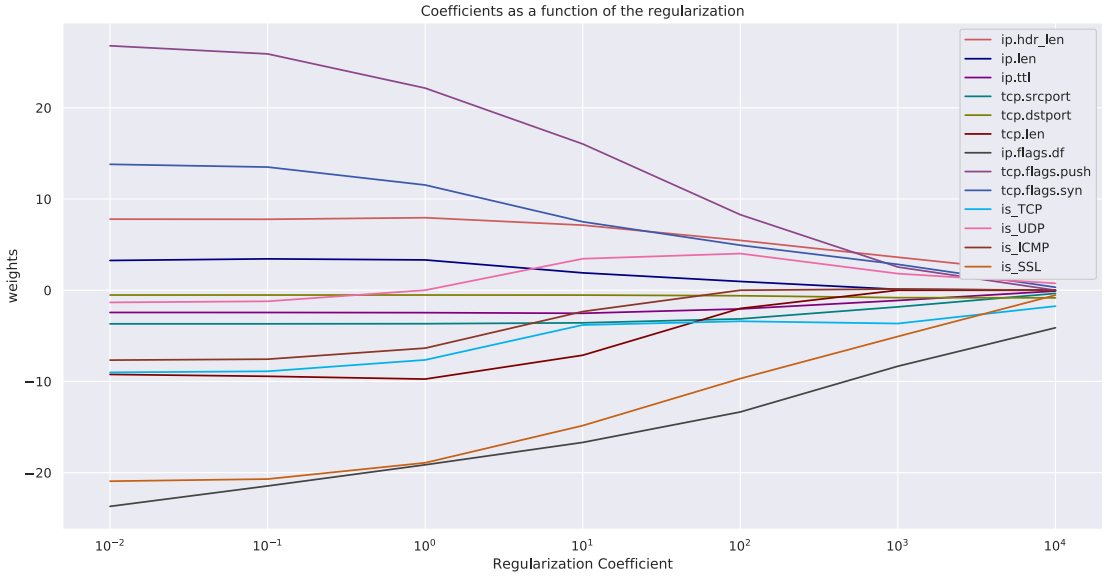
TABLE 6. A comparison between *ShieldRNN* and unsupervised techniques on CIC-IDS2017 Wednesday.

Model	Accuracy	F1-Score	Precision	Recall
ShieldRNN	**99.998%**	**99.992%**	**99.983%**	**100.00%**
SAE-1SVM [36]	99.35%	99.11%	99.97%	98.28%
OC-SVM [36]	98.00%	97.22%	96.26%	98.21%
MemAE [37]	N/A	95.61%	99.75%	92.02%

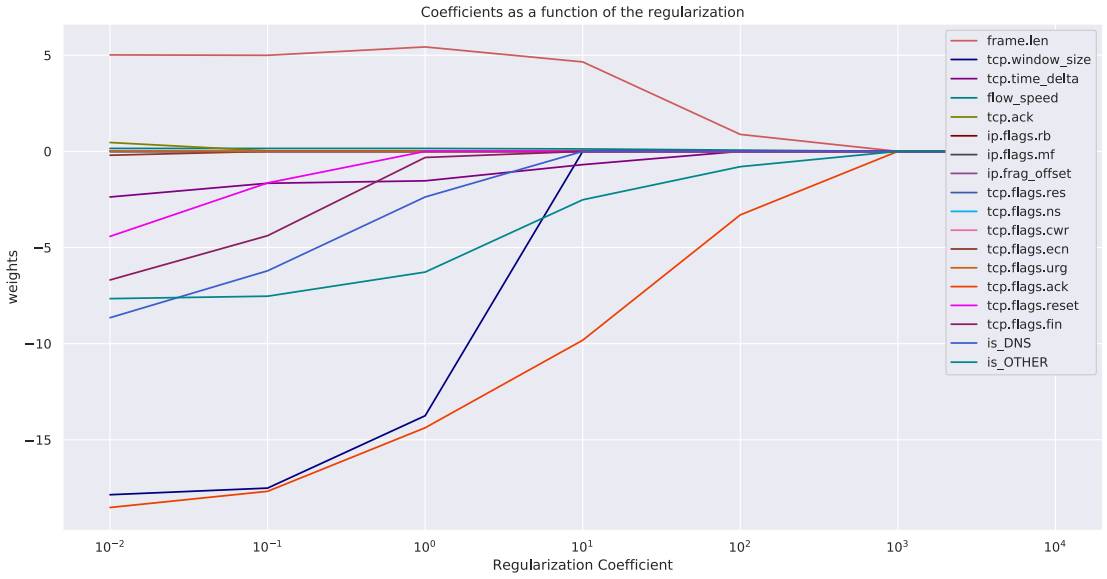
TABLE 7. Summary of *ShieldRNN* (LSTM_mix_MV_T) results on the CIC-IoT2022, CIC-IDS2017 Wednesday, and CIC-IDS2017 Friday datasets.

Dataset	Accuracy	F1-Score	Precision	Recall
CIC-IDS2017-Wed	99.998%	99.992%	99.983%	100.00%
CIC-IDS2017-Fri	100.00%	100.00%	100.00%	100.00%
CIC-IoT2022	99.989%	99.989%	99.985%	99.993%

Friday 7 (CIC-IDS2017-Friday) as it was used by [22], [32], and [33]. The total number of packets in the traffic trace was 9,997,874 where 926,978 packets were *attack* packets and 9,070,896 packets were *normal* packets. In the testing phase, we computed True Negatives (TN), False Positives (FP), False Negatives (FN), and True Positives (TP) for each sequence length in the training list of sequence lengths as well as on the list of randomly generated sequence lengths as shown in Table 2. After that, we calculated the metrics: F1-score, Accuracy, Precision, and Recall using the total TN, FP, FN and TP from Table 2. The reason to use the total TN, FP, FN and TP is that we wanted to evaluate the overall performance of our models against the state-of-the-art solutions. Table 5 shows the performance of our models against the state-of-the-art models. We repeated the same process to train and evaluate the model LSTM_mix_MV_T on the traffic trace of Wednesday 5 (CIC-IDS2017-Wednesday). The suffix "_mix_MV_T" in the model's name means that the model was trained using the algorithms 2, 4, and 5 and it was evaluated using the list of training sequence lengths. The total number of packets in the PCAP traffic trace of CIC-IDS2017-Wednesday was 13,788,878 where 1,383,651 packets were *attack* packets and 12,405,227 packets were *normal*



(a) Selected features

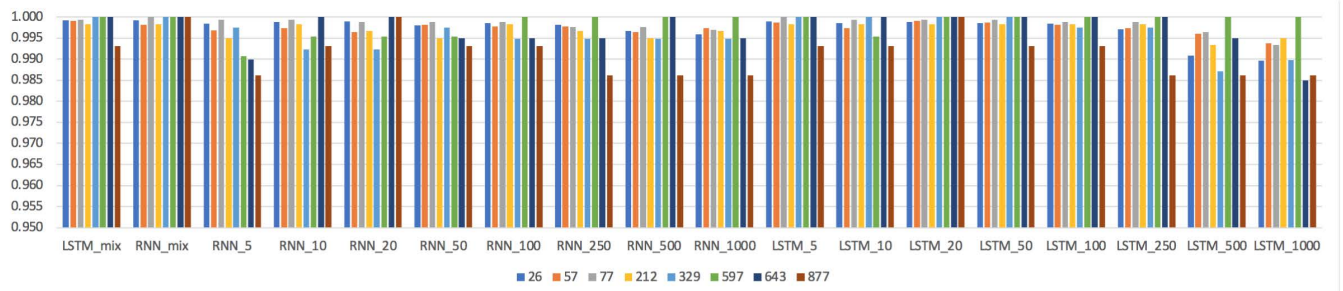


(b) Discarded features

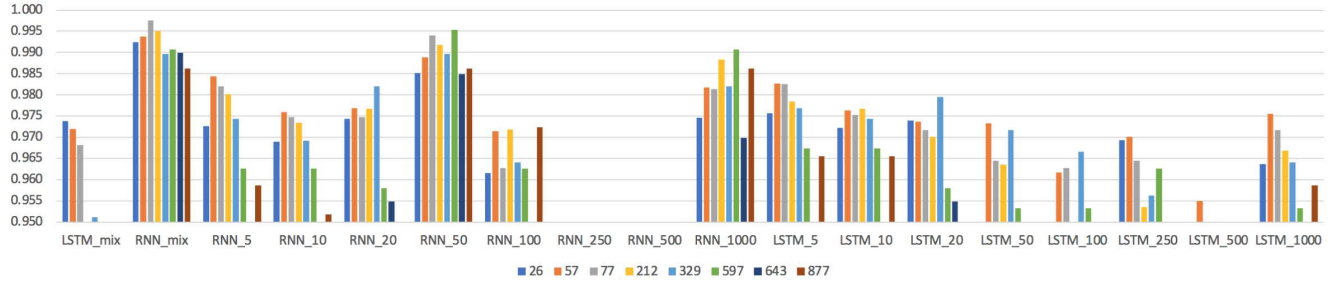
FIGURE 8. **LASSO feature selection when applied on our dataset.**

packets. The results of the LSTM_mix_MV_T model on CIC-IDS2017-Wednesday compared to the state-of-the-art models are shown in Table 4. Also, we compared *ShieldRNN* (LSTM_mix_MV_T) with unsupervised techniques on CIC-IDS2017-Wednesday dataset and we found that *ShieldRNN* achieved the best results among all models. Table 6 shows the results of unsupervised learning models compared to the results of *ShieldRNN*.

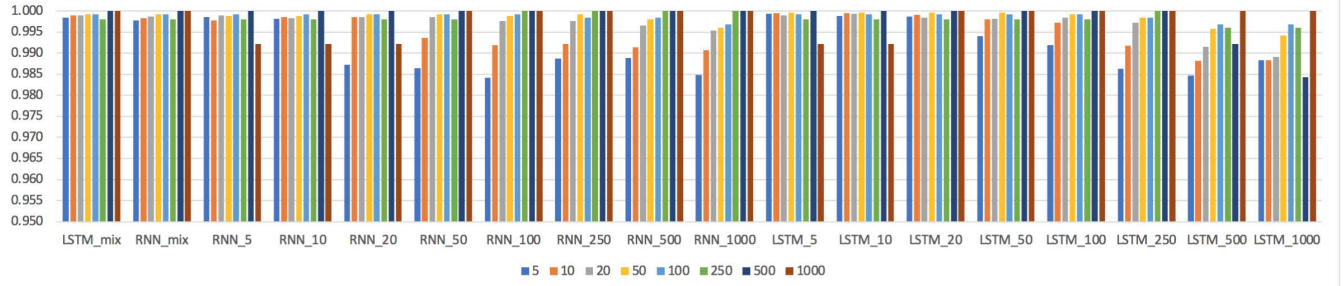
Moreover, we evaluated our techniques on the recently released CIC-IoT2022 [27] dataset which contains the normal traffic of multiple IoT devices in different scenarios that simulate the real life usage of IoT devices such as the traffic generated during the power on of the IoT device, while the IoT device is idle, and during human interaction with the IoT device. Also, the dataset contains the traffic of simulated DoS attacks performed on IoT devices. The total number of packets, used in building and evaluating the models, is 55,541,583 where the number of *normal* packets is 27,571,720 and the number of *attack* packets is 27,969,863. We prepared the data and trained the *ShieldRNN* models following the same steps as in the previous experiments. Our results set a baseline for DDoS detection on IoT devices using CIC-IoT2022 [27] dataset. Table 3 shows the details of the results of our models on CIC-IoT2022 [27] dataset. Table 7 summarizes the performance of our models on all of the CIC datasets that were used in this work.



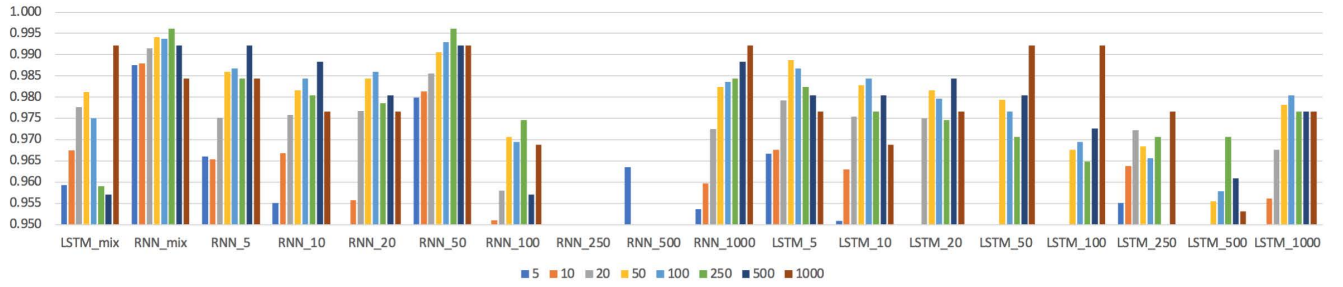
(a) TRAIN_SEQ_TO_SEQ + PREDICT_MAJORITY_VOTING Algorithms + Random Sequence lengths



(b) TRAIN_LAST_OUTPUT + PREDICT_LAST_OUTPUT Algorithms + Random Sequence lengths



(c) TRAIN_SEQ_TO_SEQ + PREDICT_MAJORITY_VOTING Algorithms + Training Sequence lengths



(d) TRAIN_LAST_OUTPUT + PREDICT_LAST_OUTPUT Algorithms + Training Sequence lengths

FIGURE 9. **The F1-score results of 18 models on the testing set after using our training techniques. The values in the X-axis represent the sequence lengths that each model was tested on. Figure 9a and 9b show the results of the models when they were tested on a randomly generated list of sequence lengths. Figure 9c and 9d show the results of the same models when they were tested on the same sequence lengths that they were trained on.**

V. CONCLUSION

This paper proposes a system architecture to better detect DDoS attacks originated from IoT devices that are infected and controlled by a botnet. The system is composed of two types of machine learning classifiers: a lightweight classifier that is deployed on the IoT node, and a more accurate classifier that is deployed on the server to do further analysis. For the IoT lightweight classifier, we trained 12 classifiers and we found that a simple Random Forest classifier with only 3 trees achieved 99.983% accuracy and 99.979% F1-score when we tested it on the test set which made it possible to implement such a simple classifier on an IoT device with limited resources. For the accurate classifier, we developed new algorithms of training and prediction for RNN that we called *ShieldRNN*. We found that using *ShieldRNN*, i.e., using a mix of sequence lengths combined with seq2seq training and majority voting prediction (Algorithms 1, 4, and 5) achieved the highest F1-score of values: 99.919%, 99.822%, 100%,

Sequence Length	26	57	77	212	329	597	643	877
LSTM_mix	0.999187	0.999109	0.999398	0.998342	1	1	1	0.993103
RNN_mix	0.999187	0.998218	1	0.998342	1	1	1	1
RNN_5	0.998375	0.996882	0.999398	0.995025	0.997429	0.990654	0.98995	0.986207
RNN_10	0.998781	0.997327	0.999398	0.998342	0.992288	0.995327	1	0.993103
RNN_20	0.998984	0.996437	0.998797	0.996683	0.992288	0.995327	1	1
RNN_50	0.997969	0.998218	0.998797	0.995025	0.997429	0.995327	0.994975	0.993103
RNN_100	0.998578	0.997773	0.998797	0.998342	0.994859	1	0.994975	0.993103
RNN_250	0.998172	0.997773	0.997593	0.996683	0.994859	1	0.994975	0.986207
RNN_500	0.99675	0.996437	0.997593	0.995025	0.994859	1	1	0.986207
RNN_1000	0.995937	0.997327	0.996992	0.996683	0.994859	1	0.994975	0.986207
LSTM_5	0.998984	0.998664	1	0.998342	1	1	1	0.993103
LSTM_10	0.998578	0.997327	0.999398	0.998342	1	0.995327	1	0.993103
LSTM_20	0.998781	0.999109	0.999398	0.998342	1	1	1	1
LSTM_50	0.998578	0.998664	0.999398	0.998342	1	1	1	0.993103
LSTM_100	0.998375	0.998218	0.998797	0.998342	0.997429	1	1	0.993103
LSTM_250	0.997156	0.997327	0.998797	0.998342	0.997429	1	1	0.986207
LSTM_500	0.990859	0.995991	0.99639	0.993367	0.987147	1	0.994975	0.986207
LSTM_1000	0.98964	0.993764	0.993381	0.995025	0.989717	1	0.984925	0.986207

(a) Results of Train Seq-to-Seq with Majority Voting Algorithms

Sequence Length	26	57	77	212	329	597	643	877
LSTM_mix	0.973796	0.971938	0.968111	0.946932	0.951157	0.920561	0.894472	0.855172
RNN_mix	0.992484	0.993764	0.997593	0.995025	0.989717	0.990654	0.98995	0.986207
RNN_5	0.972578	0.98441	0.981949	0.9801	0.974293	0.962617	0.949749	0.958621
RNN_10	0.968921	0.975947	0.974729	0.973466	0.969152	0.962617	0.944724	0.951724
RNN_20	0.974406	0.976837	0.974729	0.976783	0.982005	0.957944	0.954774	0.931034
RNN_50	0.985172	0.988864	0.993983	0.991708	0.989717	0.995327	0.984925	0.986207
RNN_100	0.961609	0.971492	0.962696	0.971808	0.96401	0.962617	0.914573	0.972414
RNN_250	0.910624	0.914922	0.887485	0.882255	0.902314	0.817757	0.839196	0.765517
RNN_500	0.929515	0.919376	0.89651	0.87728	0.879177	0.808411	0.834171	0.8
RNN_1000	0.974609	0.981737	0.981348	0.988391	0.982005	0.990654	0.969849	0.986207
LSTM_5	0.975625	0.982628	0.982551	0.978441	0.976864	0.96729	0.949749	0.965517
LSTM_10	0.972171	0.976392	0.975331	0.976783	0.974293	0.96729	0.944724	0.965517
LSTM_20	0.974	0.973719	0.971721	0.970149	0.979434	0.957944	0.954774	0.944828
LSTM_50	0.937843	0.973274	0.964501	0.963516	0.971722	0.953271	0.949749	0.931034
LSTM_100	0.948202	0.961693	0.962696	0.940299	0.966581	0.953271	0.934673	0.917241
LSTM_250	0.969328	0.970156	0.964501	0.953566	0.956298	0.962617	0.944724	0.917241
LSTM_500	0.942515	0.955011	0.948857	0.940299	0.943445	0.897196	0.889447	0.875862
LSTM_1000	0.96364	0.975501	0.971721	0.966833	0.96401	0.953271	0.949749	0.958621

(b) Results of Train Last Output with Last Output Prediction Algorithms

FIGURE 10. F1-scores of all RNN/LSTM models on the test set when they were tested on a randomly generated list of sequence lengths other than the list that the models were trained on. Each row shows the name of the model and its results when it was tested on the sequence length shown at the top of the column. The model name consist of two parts separated by an underscore: the first part shows the model name and the second part shows the sequence length that the model was trained on. For example, the model name: ***LSTM_10*** means that the model was LSTM and the sequence length was 10 which means that each 10 consecutive packets are fed to the model as a single example to train the model. In case the model name has the suffix ***_mix*** prefix instead of a number, it means that the model was trained on various sequence lengths each one was used to train the model for a single epoch.

99.834%, 100%, 100%, 100%, and 100% when it was tested on our dataset with the randomly generated list of sequence lengths: 26, 57, 77, 212, 329, 597, 643, and 877, respectively. Also, we found that using the same sequence length in both training and testing tends to achieve higher scores compared to testing on sequence length different than the sequence length in the training phase. Moreover, we evaluated *ShieldRNN* on CIC-IDS2017 [26] dataset to compare it with others algorithms and we found that it outperformed all of them. Finally, we set baseline results for DDoS detection using *ShieldRNN* on the CIC-IoT2022 [27] dataset.

APPENDIX A
FEATURE SELECTION BY LASSO

See Figure 8.

APPENDIX B
MODELS PERFORMANCE

See Figure 9.

Sequence Length	5	10	20	50	100	250	500	1000
LSTM_mix	0.998477	0.998906	0.998906	0.999219	0.999219	0.998047	1	1
RNN_mix	0.997812	0.998281	0.99875	0.999219	0.999219	0.998047	1	1
RNN_5	0.998594	0.997812	0.998906	0.998828	0.999219	0.998047	1	0.992188
RNN_10	0.998125	0.998594	0.998281	0.998828	0.999219	0.998047	1	0.992188
RNN_20	0.987187	0.998516	0.998594	0.999219	0.999219	0.998047	1	0.992188
RNN_50	0.986406	0.993594	0.998594	0.999219	0.999219	0.998047	1	1
RNN_100	0.984141	0.991875	0.997656	0.998828	0.999219	1	1	1
RNN_250	0.988672	0.992188	0.997656	0.999219	0.998437	1	1	1
RNN_500	0.988867	0.991406	0.996563	0.998047	0.998437	1	1	1
RNN_1000	0.984883	0.990703	0.995313	0.996094	0.996875	1	1	1
LSTM_5	0.999297	0.999453	0.998906	0.999609	0.999219	0.998047	1	0.992188
LSTM_10	0.998867	0.999531	0.999375	0.999609	0.999219	0.998047	1	0.992188
LSTM_20	0.998672	0.999062	0.998437	0.999609	0.999219	0.998047	1	1
LSTM_50	0.994023	0.997969	0.998125	0.999609	0.999219	0.998047	1	1
LSTM_100	0.991875	0.997266	0.998437	0.999219	0.999219	0.998047	1	1
LSTM_250	0.986289	0.991719	0.997188	0.998437	0.998437	1	1	1
LSTM_500	0.984727	0.988203	0.991563	0.995703	0.996875	0.996094	0.992188	1
LSTM_1000	0.988359	0.988359	0.989062	0.994141	0.996875	0.996094	0.984375	1

(a) Results of Train Seq-to-Seq with Majority Voting Algorithms

Sequence Length	5	10	20	50	100	250	500	1000
LSTM_mix	0.959258	0.9675	0.977656	0.98125	0.975	0.958984	0.957031	0.992188
RNN_mix	0.9875	0.987969	0.991563	0.994141	0.99375	0.996094	0.992188	0.984375
RNN_5	0.965977	0.965391	0.975156	0.985938	0.986719	0.984375	0.992188	0.984375
RNN_10	0.955078	0.966875	0.975781	0.981641	0.984375	0.980469	0.988281	0.976562
RNN_20	0.944102	0.955781	0.976719	0.984375	0.985938	0.978516	0.980469	0.976562
RNN_50	0.979883	0.981328	0.985625	0.990625	0.992969	0.996094	0.992188	0.992188
RNN_100	0.935	0.950937	0.957969	0.970703	0.969531	0.974609	0.957031	0.96875
RNN_250	0.938594	0.915234	0.915	0.909766	0.907031	0.90625	0.867188	0.898438
RNN_500	0.963594	0.944375	0.933906	0.921875	0.910937	0.902344	0.863281	0.890625
RNN_1000	0.953633	0.959688	0.9725	0.982422	0.983594	0.984375	0.988281	0.992188
LSTM_5	0.96668	0.967578	0.979219	0.988672	0.986719	0.982422	0.980469	0.976562
LSTM_10	0.95082	0.963047	0.975469	0.982812	0.984375	0.976562	0.980469	0.96875
LSTM_20	0.944961	0.949844	0.975	0.981641	0.979688	0.974609	0.984375	0.976562
LSTM_50	0.84625	0.861328	0.909219	0.979297	0.976562	0.970703	0.980469	0.992188
LSTM_100	0.853711	0.898047	0.940312	0.967578	0.969531	0.964844	0.972656	0.992188
LSTM_250	0.955078	0.963828	0.972187	0.968359	0.965625	0.970703	0.945312	0.976562
LSTM_500	0.859258	0.912266	0.943594	0.955469	0.957812	0.970703	0.960938	0.953125
LSTM_1000	0.942773	0.956094	0.967656	0.978125	0.980469	0.976562	0.976562	0.976562

(b) Results of Train Last Output with Last Output Prediction Algorithms

FIGURE 11. F1-scores of all RNN/LSTM models on the test set when they were tested on the same list of sequence lengths that they were trained on.

APPENDIX C
MODELS RESULTS TABLES

See Figures 10 and 11.

REFERENCES

[1] P. Kamboj, M. C. Trivedi, V. K. Yadav, and V. K. Singh, "Detection techniques of DDoS attacks: A survey," in *Proc. 4th IEEE Uttar Pradesh Sect. Int. Conf. Electr., Comput. Electron. (UPCON)*, Oct. 2017, pp. 675–679.

[2] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning DDoS detection for consumer Internet of Things devices," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2018, pp. 29–35.

[3] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, and D. Kumar, "Understanding the mirai botnet," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1093–1110.

[4] A. Saboor and B. Aslam, "Analyses of flow based techniques to detect distributed denial of service attacks," in *Proc. 12th Int. Bhurban Conf. Appl. Sci. Technol. (IBCAST)*, Jan. 2015, pp. 354–362.

[5] M. F. Umer, M. Sher, and Y. Bi, "Flow-based intrusion detection: Techniques and challenges," *Comput. Secur.*, vol. 70, pp. 238–254, Sep. 2017.

[6] I. Ullah and Q. H. Mahmoud, "A two-level flow-based anomalous activity detection system for IoT networks," *Electronics*, vol. 9, no. 3, p. 530, Mar. 2020.

[7] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, "Machine learning-based IoT-botnet attack detection with sequential architecture," *Sensors*, vol. 20, no. 16, p. 4372, Aug. 2020.

[8] M. Roesch, "Snort: Lightweight intrusion detection for networks," *Lisa*, vol. 99, pp. 229–238, Jun. 1999.

[9] OISF. (Aug. 2020). *Suricata User Guide*. [Online]. Available: https://suricata.readthedocs.io/en/latest/

[10] S. A. R. Shah and B. Issac, "Performance comparison of intrusion detection systems and application of machine learning to Snort system," *Future Generat. Comput. Syst.*, vol. 80, pp. 157–170, Mar. 2018.

[11] T. Zitta, M. Neruda, and L. Vojtech, "The security of RFID readers with IDS/IPS solution using raspberry Pi," in *Proc. 18th Int. Carpathian Control Conf. (ICCC)*, May 2017, pp. 316–320.

[12] S. Badotra and S. N. Panda, "SNORT based early DDoS detection system using opendaylight and open networking operating system in software defined networking," *Cluster Comput.*, vol. 24, no. 1, pp. 501–513, Mar. 2021.

[13] M. Shafiq, Z. Tian, Y. Sun, X. Du, and M. Guizani, "Selection of effective machine learning algorithm and BoT-IoT attacks traffic identification for Internet of Things in smart city," *Future Gener. Comput. Syst.*, vol. 107, pp. 433–442, Jun. 2020.

[14] J. Hou, P. Fu, Z. Cao, and A. Xu, "Machine learning based DDos detection through NetFlow analysis," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 1–6.

[15] E. Hodo, X. Bellekens, A. Hamilton, P.-L. Dubouilh, E. Iorkyase, C. Tachtatzis, and R. Atkinson, "Threat analysis of IoT networks using artificial neural network intrusion detection system," in *Proc. Int. Symp. Netw., Comput. Commun. (ISNCC)*, May 2016, pp. 1–6.

[16] B. S. Khater, A. A. Wahab, M. Idris, M. A. Hussain, and A. A. Ibrahim, "A lightweight perceptron-based intrusion detection system for fog computing," *Appl. Sci.*, vol. 9, no. 1, p. 178, 2019.

[17] 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.

[18] S. A. Althubiti, E. M. Jones, and K. Roy, "LSTM for anomaly-based network intrusion detection," in *Proc. 28th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2018, pp. 1–3.

[19] P. S. Muhuri, P. Chatterjee, X. Yuan, K. Roy, and A. Esterline, "Using a long short-term memory recurrent neural network (LSTM-RNN) to classify network attacks," *Information*, vol. 11, no. 5, p. 243, May 2020.

[20] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An empirical study on network anomaly detection using convolutional neural networks," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1595–1598.

[21] W. Jo, S. Kim, C. Lee, and T. Shon, "Packet preprocessing in CNN-based network intrusion detection system," *Electronics*, vol. 9, no. 7, p. 1151, Jul. 2020.

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

[23] M. M. Salim, S. Rathore, and J. H. Park, "Distributed denial of service attacks and its defenses in IoT: A survey," *J. Supercomput.*, vol. 76, no. 7, pp. 5320–5363, 2020.

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

[25] J. F. Macgregor and T. J. Harris, "The exponentially weighted moving variance," *J. Quality Technol.*, vol. 25, no. 2, pp. 106–118, Apr. 1993.

[26] 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*, vol. 1, Jan. 2018, pp. 108–116.

[27] *CIC IoT dataset 2022*. Accessed: May 16, 2022. [Online]. Available: https://www.unb.ca/cic/datasets/iotdataset-2022.html

[28] Y. Zhou, G. Cheng, S. Jiang, and M. Dai, "Building an efficient intrusion detection system based on feature selection and ensemble classifier," *Comput. Netw.*, vol. 174, Jun. 2020, Art. no. 107247.

[29] A. S. Iliyasu, U. A. Abdurrahman, and L. Zheng, "Few-shot network intrusion detection using discriminative representation learning with supervised autoencoder," *Appl. Sci.*, vol. 12, no. 5, p. 2351, Feb. 2022.

[30] T. A. Tang, D. McLernon, L. Mhamdi, S. A. R. Zaidi, and M. Ghogho, "Intrusion detection in SDN-based networks: Deep recurrent neural network approach," in *Deep Learning Applications for Cyber Security* (Advanced Sciences and Technologies for Security Applications). Cham, Switzerland: Springer, 2019, pp. 175–195.

[31] R. V. Mendonca, A. A. M. Teodoro, R. L. Rosa, M. Saadi, D. C. Melgarejo, P. H. J. Nardelli, and D. Z. Rodriguez, "Intrusion detection system based on fast hierarchical deep convolutional neural network," *IEEE Access*, vol. 9, pp. 61024–61034, 2021.

[32] M. Roopak, G. Y. Tian, and J. Chambers, "Deep learning models for cyber security in IoT networks," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2019, pp. 0452–0457.

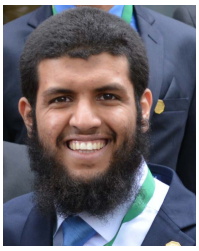
[33] Y. Yao, L. Su, and Z. Lu, "DeepGFL: Deep feature learning via graph for attack detection on flow-based network traffic," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 579–584.

[34] A. Chen, Y. Fu, X. Zheng, and G. Lu, "An efficient network behavior anomaly detection using a hybrid DBN-LSTM network," *Comput. Secur.*, vol. 114, Mar. 2022, Art. no. 102600.

[35] R. Ahmad, I. Alsmadi, W. Alhamdani, and L. Tawalbeh, "A comprehensive deep learning benchmark for IoT IDS," *Comput. Secur.*, vol. 114, Mar. 2022, Art. no. 102588.

[36] L. Mhamdi, D. McLernon, F. El-moussa, S. A. R. Zaidi, M. Ghogho, and T. Tang, "A deep learning approach combining autoencoder with one-class SVM for DDoS attack detection in SDNs," in *Proc. IEEE 8th Int. Conf. Commun. Netw. (ComNet)*, Oct. 2020, pp. 1–6.

[37] B. Min, J. Yoo, S. Kim, D. Shin, and D. Shin, "Network anomaly detection using memory-augmented deep autoencoder," *IEEE Access*, vol. 9, pp. 104695–104706, 2021.



FARIS ALASMARY received the B.S. degree in software engineering from the King Fahd University of Petroleum and Minerals, Dhahran, in 2018. He is currently pursuing the master's degree with the College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. He is also an employee working on developing various statistical and end-to-end speech recognition models for Arabic and English languages. His research interests include automatic speech recognition, text-to-speech, and speaker identification.

SULAIMAN ALRADDADI received the bachelor's degree in computer engineering from the Yanbu University College (YUC), in 2020. He is currently working in the cybersecurity field as a Penetration Tester interested in hardware and the IoT assessment.

SAAD AL-AHMADI (Senior Member, IEEE) is currently an Associate Professor with the Department of Computer Science, King Saud University, Saudi Arabia. He has published many articles in highly cited journals and worked as a part-time consultant in several government organizations as well as the private sector. His current research interests include the IoT security, machine learning for cybersecurity, and future generation networks.

JALAL AL-MUHTADI received the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign, USA. He is currently the Director and a Cybersecurity Consultant of the Center of Excellence in Information Assurance (CoEIA). He is also an Associate Professor with the Department of Computer Science, King Saud University. He has over 50 scientific publications in the areas of cybersecurity, information assurance, privacy, and the IoT security.