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

FARIS ALASMARY1, SULAIMAN ALRADDADI2, SAAD AL-AHMADI1, (Senior Member, IEEE), AND JALAL AL-MUHTADI1

1Department of Computer Science, King Saud University, Riyadh 11451, Saudi Arabia
2Department 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
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 Suricata 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 Suricata. On the other hand, Suricata 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 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 Suricata 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

FIGURE 1. This diagram shows a DDoS attack using a botnet.
F. Alasmary et al.: ShieldRNN: A Distributed Flow-Based DDoS Detection Solution

FIGURE 2. The distribution of packets in our dataset.

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

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.

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].

| Feature Name       | Description                      |
|--------------------|----------------------------------|
| iframe.len         | Frame length on the wire         |
| ip.chk_len         | IP header length                 |
| ip.ttl             | 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.flags.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 acknowledgement 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 acknowledgement 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           |

\(^1\)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 \( \text{num}_\text{examples} \times \text{seq}_\text{len} \times \text{num}_\text{features} \) instead of the original 2D matrix size \( \text{total}_\text{num}_\text{packets} \times \text{num}_\text{features} \) where \( \text{num}_\text{examples} \) is the number of examples, \( \text{seq}_\text{len} \) is the sequence length, i.e., the number of packets per example, \( \text{num}_\text{features} \) is the number of features which represents the input size of the model, and \( \text{total}_\text{num}_\text{packets} = \text{num}_\text{examples} \times \text{seq}_\text{len} \) 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:

\[
A_t = \alpha x_t + (1 - \alpha) A_{t-1} \\
V_t = \beta (x_t - A_t)^2 + (1 - \beta) V_{t-1}
\]

where \( x_t \) is the current packet, \( A_{t-1} \) and \( V_{t-1} \) are the previous EWMA and EWMV, respectively. \( \alpha \) and \( \beta \) 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.

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 coefficient was then used to estimate the remaining features using the training set.
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 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 4.** The process of converting the dataset from a 2D matrix into a 3D tensor of sequences of packets and vice versa.

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

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

Algorithm 3 Predict_Last_output

**Input:** $M$: a pretrained vanilla RNN model or its variants.

**Output:** $\hat{y}_{\text{final}}$: the prediction of the trained model:

1. $X_{3d} \leftarrow \text{matrix_to_tensor}(X, T)$
2. $\hat{y}_{3d} \leftarrow M(X_{3d})$
3. $\hat{y}_{\text{final}} \leftarrow \hat{y}_{3d}[T] // \text{take the last label}$
4. return $\hat{y}_{\text{final}}$

Algorithm 4 Predict_Majority_Voting

**Input:** $M$: a pretrained vanilla RNN model or its variants.

**Output:** $\hat{y}_{\text{final}}$: the prediction of the trained model:

1. $X_{3d} \leftarrow \text{matrix_to_tensor}(X, T)$
2. $\hat{y} \leftarrow M(X_{3d})$
3. $\text{count} \leftarrow 0$
4. for $t = 1 \text{ to } T$ do
5.   if $\hat{y}_t = \text{"attack"}$ then
6.     $\text{count} \leftarrow \text{count} + 1$
7.   end
8. end
9. if (count / $T$) $\geq$ 0.5 then
10.   $\hat{y}_{\text{final}} \leftarrow \text{"attack"}$
11. else
12.   $\hat{y}_{\text{final}} \leftarrow \text{"normal"}$
13. end
14. return $\hat{y}_{\text{final}}$
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 performance.

### 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

```plaintext
1. \( \text{loss}_\text{total} \leftarrow 0 \)

2. for \( i = 1 \) to \( m \) do

3. \( \hat{y}_{3d} \leftarrow M(X_{3d}[i]) \)

4. \( \text{loss}_\text{sum} \leftarrow 0 \)

5. for \( t = 1 \) to \( T \) do

6. \( y_t \leftarrow y_{3d}[i][t] \)

7. \( \hat{y}_t \leftarrow \hat{y}_{3d}[t] \)

8. \( \text{loss} \leftarrow \text{MSE}(y_t, \hat{y}_t) \)

9. \( \text{loss}_\text{sum} \leftarrow \text{loss}_\text{sum} + \text{loss} \)

10. end

11. \( \text{loss}_\text{avg} \leftarrow \text{loss}_\text{sum} / T \)

12. \( \text{loss}_\text{total} \leftarrow \text{loss}_\text{total} + \text{loss}_\text{sum} \)

13. end

14. \( \text{loss}_\text{totalavg} \leftarrow \text{loss}_\text{total} / m \)

15. \( \text{grads} \leftarrow \text{compute_gradient}(M, \text{loss}_\text{totalavg}) \)

16. \( M \leftarrow \text{update_weights}(M, \text{grads}) \)

17. 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

```plaintext
1. \( \text{loss}_\text{total} \leftarrow 0 \)

2. for \( i = 1 \) to \( m \) do

3. \( \hat{y}_{3d} \leftarrow M(X_{3d}[i]) \)

4. \( \hat{y}_{\text{last}} \leftarrow \hat{y}_{3d}[T] \) // take the last label

5. \( \text{count} \leftarrow 0 \)

6. for \( t = 1 \) to \( T \) do

7. \( y_t \leftarrow y_{3d}[i][t] \)

8. if \( y_t = \text{“attack”} \) then

9. \( \text{count} \leftarrow \text{count} + 1 \)

10. end

11. end

12. if (\( \text{count} / T \) \( \geq 0.5 \)) then

13. \( y_{\text{true}} \leftarrow \text{“attack”} \)

14. else

15. \( y_{\text{true}} \leftarrow \text{“normal”} \)

16. end

17. \( \text{loss} \leftarrow \text{MSE}(y_{\text{true}}, \hat{y}_{\text{last}}) \)

18. \( \text{loss}_\text{total} \leftarrow \text{loss}_\text{total} + \text{loss} \)

19. end

20. \( \text{loss}_\text{totalavg} \leftarrow \text{loss}_\text{total} / m \)

21. \( \text{grads} \leftarrow \text{compute_gradient}(M, \text{loss}_\text{totalavg}) \)

22. \( M \leftarrow \text{update_weights}(M, \text{grads}) \)

23. return \( M \)
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].

| seq_len | LSTM_mix_MV_T | RNN_mix_MV_T | LSTM_mix_MV_R | RNN_mix_MV_R |
|---------|---------------|--------------|---------------|--------------|
| TN      | FP            | FN           | TP            | FN           |
| 5       | 550550        | 148          | 57            | 560245       |
| 10      | 273242        | 15           | 15            | 282188       |
| 20      | 136546        | 4            | 2             | 141198       |
| 50      | 54082         | 0            | 0             | 57013        |
| 100     | 26847         | 0            | 1             | 28701        |
| 250     | 10670         | 0            | 11550         | 2             |
| 500     | 5290          | 0            | 5820          | 1             |
| 1000    | 2632          | 1            | 2922          | 1             |

Total: 1059899 | 168 | 75 | 1089643 | 88 | 1089630 |

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  | 99.99%   | 99.99%   | 99.93%    | 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]   | 99.00%   | 99.00%   | 99.00%    | N/A    |
| 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   | 99.95%   | 99.75%   | 99.71%    | 99.79%
| LSTM_mix_MV_R  | 99.98%   | 99.87%   | 100.00%   | 99.97%
| RNN_mix_MV_R   | 99.94%   | 99.73%   | 99.61%    | 99.83%
| 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]  | 99.81%   | N/A      | N/A       | N/A    |
| LSTM [35]      | N/A      | 97.76%   | 96.43%    | 99.23% |

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

| Model         | Accuracy | F1-Score | Precision | Recall |
|---------------|----------|----------|-----------|--------|
| ShieldRNN     | 99.99%   | 99.99%   | 99.93%    | 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]    | 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| 99.99%   | 99.99%   | 99.93%    | 100.00%
| CIC-IDS2017-Fri| 100.00%  | 100.00%  | 100.00%   | 100.00%
| CIC-IoT2022    | 99.98%   | 99.98%   | 99.98%    | 99.99% |

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

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.
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%,
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. 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.
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/
I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a CIC IoT dataset 2022,” in Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC), Jan. 2019, pp. 0452–0457.

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.

A. Chen, Y. Fu, X. Zheng, and G. Lu, “An efficient network behavior anomaly detection using a hybrid DBN-LSTM network,” IEEE Access, vol. 11, Mar. 2022, Art. no. 102600.

R. Ahmad, I. Alsmadi, W. Alhamdani, and L. Tawalbeh, “A comprehensive deep learning benchmark for IoT IDS,” IEEE Access, vol. 11, Mar. 2022, Art. no. 102588.

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 ALASMERY received the B.S. degree in software engineering from the King Fahd University of Petroleum and Minerals, Dharan, 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.