Challenge Collapsar (CC) Attack Traffic Detection Based on Packet Field Differentiated Preprocessing and Deep Neural Network

Xiaolin Liu[1,3], Shuhao Li[1,2](✉), Yongzheng Zhang[1,2,3], Xiaochun Yun[4], and Jia Li[4]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
liuxiaolin191@mails.ucas.ac.cn, {lishuhao,zhangyongzhen}@iie.ac.cn
[2] Key Laboratory of Network Assessment Technology, Chinese Academy of Sciences, Beijing, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[4] National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing, China
{yunxiaochun,lijia}@cert.org.cn

Abstract. Distributed Denial of Service (DDoS) attack is one of the top cyber threats. As a kind of application layer DDoS attack, Challenge Collapsar (CC) attack has become a real headache for defenders. However, there are many researches on DDoS attack, but few on CC attack. The related works on CC attack employ rule-based and machine learning-based models, and just validate their models on the outdated public datasets. These works appear to lag behind once the attack pattern changes. In this paper, we present a model based on packet Field Differentiated Preprocessing and Deep neural network (FDPD) to address this problem. Besides, we collected a fresh dataset which contains 7.92 million packets from real network traffic to train and validate FDPD model. The experimental results show that the *accuracy* of this model reaches 98.55%, the F_1 value reaches 98.59%, which is 3% higher than the previous models (SVM and Random Forest-based detection model), and the training speed is increased by 17 times in the same environment. It proved that the proposed model can help defenders improve the efficiency of detecting CC attack.

Keywords: Malicious traffic detection · CC attack · Packet Field Differentiated Preprocessing · Deep neural network

Supported by the National Key Research and Development Program of China (Grant No.2016YFB0801502), and the National Natural Science Foundation of China (Grant No.U1736218).

1 Introduction

With the development of the Internet and the advancement of technology, Distributed Denial of Service (DDoS) attack has become more and more serious. Challenge Collapsar (CC) attack is a type of DDoS attack that sends forged HTTP requests to some target web server frequently. These requests often require complicated time-consuming caculations or database operations, in order to exhaust the resource of the target web server. Because the HTTP request packets of CC attack are standard and sometimes their IPs are true, it's difficult to defend. According to reports, in February 2016, hackers launched a large-volume CC attack against XBOX , one of the world's largest online game-playing platforms, causing a 24-h impact on the business [5]. Similar incidents are happening endlessly, causing serious effects.

At present, the research on CC attack detection has the following limitations:

(1) **High feature extraction dependency.** Most of the previous detection models are based on specific rules of CC attack. For example, Moore et al. [13] found that the attack packet size and time interval had a certain regularity when the attack occurred. So they use size and time to detect. It requires a lot of statistical calculations and can't be automatically updated to cope with variant attacks.
(2) **Long model training time.** Current detection models are mostly machine learning model. We have implemented representative machine learning models to train the data, including Support Vector Machine (SVM) and Random Forest [7,17,23]. Experiments show that in the face of large-traffic dataset, these models need a long training period, which is severely delayed in practice. In order to show the contrast, we divide the massive data into several small datasets, and the results show that previous model is 17 times slower than our model.
(3) **Lack of real and fresh dataset.** Most previous works are based on public datasets [4,7,8,19], which is not new enough to cope with changing attack patterns. Public datasets are often obtained by experts through empirical and statistical analysis. Therefore, in the face of new attack variants, the samples of the public datasets appear to be backward, making it difficult to meet the ever-changing attack detection requirements in the real environment.

The contributions of this paper are as follows:

(1) **We propose a packet field differentiated preprocessing algorithm.** Considering the specific meanings of the fields in the packet, we divide all fields into four types (misleading fields, useless fields, discrete fields, non-discrete fields) and process them differently (drop, one-hot encoding, ASCII encoding). In traditional models, it is often necessary to extract the feature artificially, which depends on expert experience severely. Compared with this, our model greatly saves labor costs.

(2) **We present a CC attack detection model based on packet Field Differentiated Preprocessing and Deep neural network (FDPD).** Our model leverages its powerful self-learning capabilities to learn the implicit feature in traffic. The experimental results show that the accuracy reaches 98.55%, the recall is 98.41%, the precision is 98.76%, and the comprehensive evaluation index F_1 reaches 98.59%. Compared with the traditional SVM and Random Forest-based malicious traffic detection models, the comprehensive evaluation index value F_1 of our proposed model is increased by 3%, and the processing speed is increased by 17 times.

(3) **We collect a dataset of 7.92 million packets.** This dataset contains two parts. One is the CC attack packets from the CC attack script program based on HTTP protocol. The other is non-CC attack packtes from the backbone network traffic. And the data is labeled based on the system and manual sampling check. Compared with the public dataset, our data is more representative, which covers more CC attack ways.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 introduces traffic detection model of challenge collapsar attacks. The training and optimization process, experimental results and analysis will be showed in Sect. 4. Section 5 discusses the limitation of the proposed model. Finally, Sect. 6 concludes this paper.

2 Related Work

As an application layer attack based on HTTP protocol, CC attack is more subtle than traditional flood-based DDoS attack [16], posing an increasing challenge for Web service applications. Because of the low cost of attack and the disruptive impact, HTTP protocol packet is the primary target of GET flood attacks. As new attackers emerge, identifying these attacks is puzzling. Besides, due to the continuous upgrade of existing attack tools and the increasing network bandwidth [2], the cost of the attacker's continuous connection request to the website is getting lower and lower, and the Web server is more and more vulnerable. Sree et al. [18] suggested that the exponential growth of the usage of Internet led to cyberattacks. Among various attacks, the HTTP GET flood attack is one of the main threats to Internet services because it exhausts resources and services in the application layer. Because the attack request pattern is similar to a legitimate client, it is difficult to distinguish them.

In addition, HTTP DDoS attacks in cloud computing and SDN fields have intensified. Aborujilah et al. [1] mentioned that HTTP attack is one of the key attacks on cloud-based Web servers. Lin et al. [10] found that 43% of the three major network attacks are HTTP attack, and 27% of organizations face daily or weekly HTTP attack.

2.1 Rule-Based Detection

At present, many detection models rely on rules based on attack characteristics. Bin Xiao et al. [3] used the feature of time delay, obtained the feature distribution of the time internal by maximum likelihood estimation, and linked it to the self-organizing map neural network. And the abnormal detection is performed with the set of the detection threshold. Miao Tan et al. [20] extracted key information from different protocols packets, and proposed firefly group optimization algorithm by combining pattern search and boundary variation.

An Wang et al. [21] proposed a new dataset, and found three rules by analyzing this new dataset: 1. Location-based analysis shows most attackers come from active Botnet; 2. From the perspective of the target being attacked, multiple attacks on the same target also show a strong attack rule, so the start time of the next attack from some botnet families can be accurately predicted; 3. Different Botnets have a similar trend of initiating DDoS attacks against the same victim at the same time or in turn. Similarly, CC attack can be defended against by painting a picture of attacker, but it doesn't work when attackers change their attack way.

2.2 Machine Learning-Based Detection

In the field of CC attack detection research, machine learning models are widely used, including the original models and improvement models. Liu et al. [24] used improved neighbor propagation clustering algorithm to pre-classify the attackers' behavior with a small amount of prior knowledge, then merged and eliminated by Silhouette index, and timely made cluster perform re-cluster.

Xie et al. [22] established the Hidden Semi-Markov model and calculated the distance by the Euclidean distance formula. If the distance exceeds the limit value, they think that an attack has occurred. Kshira et al. [14] used the learning automaton (LA) model to implement DDoS defense based on the number of SYN requests and the actual number of TCP connections, and defended DDoS attacks on the hardware level to reduce the damage caused by the attack behavior. Sudip et al. [11] used learning automaton model to protect the network from DDoS attacks based on the existing optimized link state routing protocol.

2.3 Deep Learning-Based Detection

DDoS attacks often cause network delays. Xiao et al. [3] measures the delay from one network port to another, and use the maximum likelihood model to estimate the characteristic distribution of the traffic delay. They use a neural network of self-organizing maps to set thresholds based on learning results for anomaly detection. In order to solve the HTTP malicious traffic detection problem, Li et al. [9] proposed a model which combined convolutional neural network (CNN) and multilayer perceptron (MLP) based on the combination of raw data and empirical feature engineering. Their experiments showed good results. This proves the effectiveness of deep learning again. It's known that deep learning has excellent self-learning ability. Therefore, we employ deep learning to better cope with CC attack variants.

3 Modeling Methodology

Our FDPD model consists of two parts, one is a preprocessing algorithm and the other is a deep neural network, as is shown in Fig. 1.

3.1 Packet Field Differentiated Preprocessing Algorithm

CC attack is an interactive application layer attack based on the HTTP/HTTPs protocol. The potential features of this attack are often contained at the flow level, so the collected underlying TCP/IP data packets need to be restored and spliced, including CC attack flows and other flows. There are several factors that need to be considered. Firstly, the conditions that the packets can be regarded as a flow, here we introduce the flow period and the number of packets as the restoration and splice conditions. Secondly, the role of different fields in the packet should be considered while learning the CC attack behavior laws, here we classify the fields into four categories, a) misleading field, these fields can interfere with the model and have an adverse effect on model training; b) useless field, such as the fields that always stay same; c) discrete field; d) non-discrete field. Therefore, we propose a preprocessing algorithm for packet fields, as shown in Algorithm 1. Before explaining the algorithm, three functions that appear in the algorithm are illustrated.

$$f_1(protocolField) = \begin{cases} -1 & Misleadingfield \\ 0 & Constantfield \\ 1 & Discretefield \\ 2 & Non-discretefield \end{cases} \tag{1}$$

$$f_2(protocolField) = f_{one-hot}(protocolField) \tag{2}$$

$$f_3(protocolField) = f_{ASCII}(protocolField) \tag{3}$$

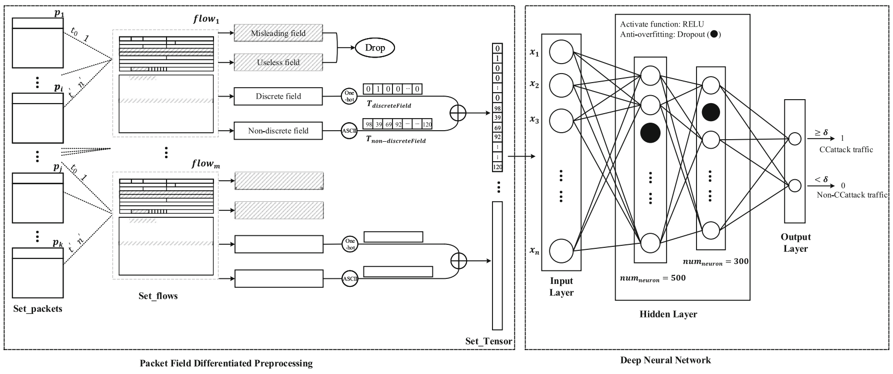


Fig. 1. The structure of FDPD model

In the algorithm, as for $p_j \in Set\_packets$, we firstly splice them into multiple flows by $quadruple = (sourceip, destinationip, sourceport, destinationport)$, and then we get $flow = \sum_{j=1}^{k} p_j$, each flow has the same quadruple. As a result, we get many flows, the set of flows is written as $Set\_flows = \sum_{i=1}^{m} flow_i$.

Considering the flow period and the number of packets, we define the time interval threshold α and the number of packets threshold β, where α is the time period constraint of a single flow, corresponding to the input variable $t\_flowPeriod$ in the algorithm, and β is the constraint of the number of packets in a single flow, corresponding to the input variable $n\_packetNumber$ in the algorithm. The p_j in each $flow_i$ is trimmed, and the original $flow_i$ is reorganized and updated. The updated $flow_i$ includes new header and payload. The structure is shown in Fig. 2. The header contains the TCP/IP packet header fields (such as "ip", "port", "reserved", "flags", etc.), and the payload contains the HTTP packet header fields and all other fields. Combining the specific meanings of the fields in the packet, we divide all fields into four types and process different types of fields differently.

(1) **Misleading field.** Misleading field mainly refers to "IP", "url", "host". For this type of fields, we adopt the strategy of dropping. Because the training sample is limited, and the value of this type of fields are fixed, but in fact, when the CC attack is launched, the attacker will constantly change the attack node and the attack target. The values of these fields are uncertain,

Algorithm 1. Packet Field Differentiated Preprocessing algorithm of FDPD

```
Input: Set_packets, t_flowPeriod, n_packetNumber
Output: Set_Tensor, Set_lengthOfTensor
1: $Initialization\ t' = t_0 + t\_flowPeriod, n' = n\_packetNumber$
2: Set_flows = Set_packets.groupby(quadruple) #classify packets to flows
3: for each $flow_i \in Set\_flows$ do
4:     for each $p_j \in flow_i$ do
5:         if $p_j.time < t'\ and\ j < n'$ then
6:             $flow_i.header = merge1(flow_i.header, p_j.header)$ #merge the different part
7:             $flow_i.payload = merge2(flow_i.payload, p_j.payload)$ #merge all the part
8: for each $flow_i \in Set\_flows$ do
9:     $l_{payload}, T_{payload} = f_3(flow_i.payload)$
10:    if $f_1(flow_i.header.protocolField) == 1$ then
11:        $l_{discreteField}, T_{discreteField} = f_2(flow_i.header.protocolField)$
12:    if $f_1(flow_i.header.protocolField) == 2$ then
13:        $l_{nondiscreteField}, T_{nondiscreteField} = f_3(flow_i.header.protocolField)$
14:    $Tensor = concat(T_{payload}, T_{discreteField}, T_{nondiscreteField})$
15:    $lengthOfTensor = l_{payload} + l_{discreteField} + l_{nondiscreteField}$
16:    Set_Tensor.add(Tensor)
17:    Set_lengthOfTensor.add(lengthOfTensor)
return Set_Tensor,Set_lengthOfTensor
```

so judging whether the traffic has CC aggressiveness through such fields has a very large error. Dropping these fields can prevent the overfitting problem of deep learning; By the way, this fields are sensitive, and removing them can protect privacy.

(2) **Useless field.** Useless field mainly refers to "reserved". We also adopt the strategy of dropping for such fields. Because such fields stay unchanged in the packet, they are not distinguishable. Although it may be helpful for other detections, such as hidden channel discovery, it does not make any sense to determine whether the traffic has CC aggressiveness. So retaining such fields will increase the cost of model training, and it may even reduce the accuracy of the model because it learns the features of other kinds of attack behaviors.

(3) **Discrete field.** Discrete field mainly refers to "flags" and "HTTP version". The values of these fields are discrete. For such fields, we use one-hot encoding. Because deep learning often uses distance when classifying, the one-hot encoding method will make the distance calculation between features more reasonable and protect the meaning of the original fields. For example, there are six values in "flags" field: 'URG' 'ACK' 'PSH' 'RST' 'SYN' 'FIN'. After one-hot encoding, the distance between two values is 1. Generally, the distance is used to measure the similarity between them, and the classification is based on the distance. By this way, they are obviously divided into the same category, it's in line with the facts; if ASCII encoding is used, the distance between values will be unreasonable.

(4) **Non-discrete field.** For Non-discrete field, such as "Payload" , use ASCII encoding to get the value from 0 to 127.

Finally, the processed fields are spliced to obtain a $Tensor$, which is used as the input of the deep neural network, as described in Sect. 3.2.

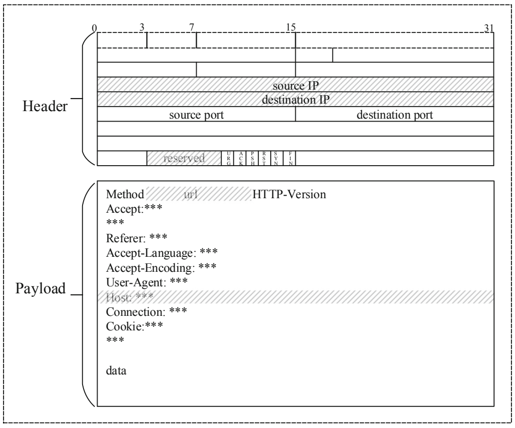


Fig. 2. Updated $flow_i$, containing the new header and payload

3.2 Deep Neural Network Structure

In addition to the necessary data preprocessing, rule-based or machine learning-based malicious traffic detection methods also require manual extraction of attack features. Most rule-based models need to summarize the representative rules of CC attack behavior through many statistical calculations. Then they use the rules directly or combine machine learning to classify the traffic, but once the attacker changes the CC attack mode slightly, these models will not work.

In the data preprocessing as described in Sect. 3.1, we get input data $Tensor$ that fully covers the packet information. After that, we didn't spend a lot of manual labor to perform feature extraction on the remaining useful fields. Instead, we automated this work. We set up a deep neural network after data preprocessing, and use its powerful self-learning capabilities to mine the hidden features in the traffic, let it automatically learn the difference between CC attack traffic and non-CC attack traffic at the packet level, which helps further improve the efficiency of CC Attack traffic detection.

Our deep neural network is a fully connected network with a four-layer structure, containing the input layer, two hidden layers, and the output layer, uses ReLU as the activation function in the hidden layers, uses Logistic Regression as the classifier in the output layer, and uses the cross entropy as the loss function to update the weight and bias values.

The number of input layer neurons depends on the dimention of $Tensor$, its value is related to three factors: $t\_flowPeriod$, $n\_packetNumber$ and *the length of packet*. The first two factors are the limiting conditions of data preprocessing, and the latter factor is needed because our deep neural network model can only process fixed-length inputs, so the length of the input data must be uniformly standardized. The determination of these three values will be discussed in the experimental section. As far hidden layers, the most important thing is the number of neurons in the hidden layers. We finally determined the numbers to be 500 and 300 respectively, and the detail is also discussed in the experimental section. The number of output layer neurons is the number of classifications, and our issue is a two-category problem. So there are two kinds of results: CC attack traffic or Non-CC attack traffic. There is a threshold δ, if the output value is not less than δ, we identify it as CC attack traffic and Non-CC attack traffic conversely.

The deep neural network learns the numerical input data, calculates the loss function, and updates the parameters through the back propagation. In order to prevent overfitting problem, we adopt Dropout strategy in all the hidden layers.

4 Experiments and Analysis

The basic experimental environment: CPU frequency is 2061 MHZ; CPU core is 64; RAM is 64 GB; GPU is GeForce GTX TITAN and the number is 8.

4.1 Dataset

The data source of this paper: a) CC attack traffic comes from the attack scripts based on HTTP protocol; b) Non-CC attack traffic comes from backbone network traffic. We annotate the dataset through system detection and manual verification. The dataset contains more than 1.8 million CC attack traffic data and more than 6.12 million non-CC attack traffic data. We share the dataset on Github[1] .

Table 1. Attacked domain name

Name	Types	Domain name
Baidu	Search	https://www.baidu.com/
Eastern Military Network	Military	http://mil.eastday.com/
Jingdong Mall	Shopping	https://www.jd.com/
Shanghai Pudong Development Bank	Bank	https://www.spdb.com.cn/
Starting point novel	Read	https://www.qidian.com
Blood war song	Online games	https://mir2.youxi.com/
Straight flush	Stock	https://www.10jqka.com.cn/
Wangyi cloud music	Music	https://music.163.com/
Ctrip	Travel	https://www.ctrip.com/
YouKu	Video	https://www.youku.com/
58City	Life	https://bj.58.com/
Tencent sports	Sports	https://sports.qq.com/
Today's headlines	News	https://www.toutiao.com/

The CC attack traffic data is generated by running CC attack scripts. These scripts covers a lot of CC attack ways, and the scripts are placed on Github[2]. In order to ensure that CC attack traffic will not cause substantial damage to the attack target, we build a network filter, by which the CC attack packets will be intercepted locally and stored as samples of the dataset. On the Ubuntu system, run the compiled CC attack scripts to launch attacks against 13 different kinds of domain names and open our network filter to capture the generated traffic. The domain name to be attacked are as shown in Table 1. And then more than 1.8 million CC attack packets are obtained through our filter.

Non-CC attack traffic data comes from communication among multiple servers and hosts. It is obtained in the backbone network, and captured through the whitelist. After capturing, more than 6.12 million packets are obtained, which basically covers the network communication among various web services.

[1] https://github.com/xiaolini/Sample_Dataset_afterprocess.
[2] https://github.com/xiaolini/Script.

4.2 Training and Validation

The evaluation indexes are as follows:

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{4}$$

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

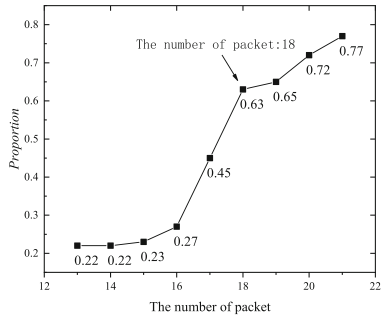
$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$F_\beta = \frac{(1+\beta^2) * Precision * Recall}{(\beta^2) * Precision + Recall}, \beta = 1 \tag{7}$$

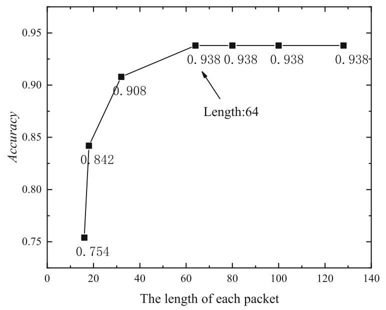
where TP (True Positive): CC traffic is correctly identified as CC traffic; FP (False Positive): Non-CC attack traffic is incorrectly identified as CC traffic; FN (False Negative): CC traffic is incorrectly identified as Non-CC attack traffic; TN (True Negative): Non-CC attack traffic is correctly identified as Non-CC attack traffic.

We divided the dataset into two parts: Training set (60%) and Test set (40%). In order to determine specific model parameters, we further divided the training set into another two parts: Estimation set and Validation set, of which the estimation set accounts for 2/3, the validation set accounts for 1/3.

In the preprocessing phase, we first need to determine the time threshold $t\_flowPeriod$ and the number threshold of packets in a flow $n\_packetNumber$. In addition, the model can only adapt to fixed-length inputs, so we need to determine a fixed value. CC attackers often try to exhaust the victim's resources, the attack traffic is often short and frequent, so we set $t\_flowPeriod$ to 0.5s. $n\_packetNumber$ and *the length of packet* is determined by experiments that shown in Fig. 3. Figure 3(a) shows the percentage of the flow with less packets than $n(n = 13, 14, ..., 21)$. We can see 63% flows contains less than 18 packets, and the statistical average value is 17.95. Therefore, we determined



(a) $n\_packetNumber$

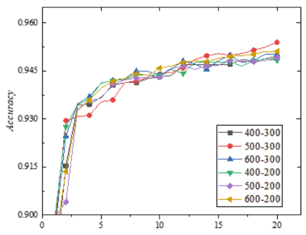


(b) *the length of packet*

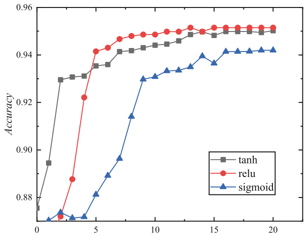
Fig. 3. The parameter determination in the data preprocessing

n_packetNumber = 18, so that it doesn't cause excessive abandonment and filling during data processing. Figure 3(b) is about the accuracy, we determined *the length of packet* is 64(bytes).

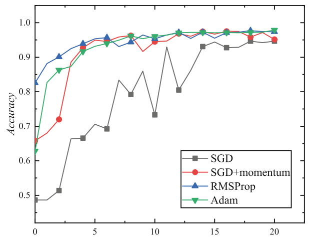
According to *t_flowPeriod*, *n_packetNumber* and *the length of packet*, we drop the redundant packets, fill the insufficient packets with −1, and remove the excessive part. Then, we get updated flow with new header and payload. We drop the misleading fields and useless fields of the new flow, use one-hot encoding for discrete fields and use ASCII encoding for non-discrete fields. After that, we convert all the useful data into numeric data, and finally obtain 1152-dimension tensors as the input of DNN.



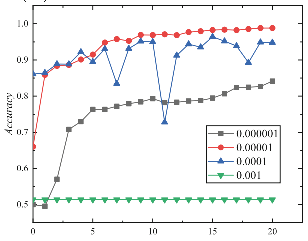
(a) The nodes of hidden layers



(b) Activation function



(c) Optimizer



(d) Learning rate

Fig. 4. The parameter determination in the model optimization ('x' axis represents 'iterations')

Our DNN is a fully connected network with a four-layer structure, containing the input layer, two hidden layers, and the output layer. Optimize DNN through four steps in Fig. 4.

1) We use two hidden layers and change the number of nodes in each hidden layer. Considering the final accuracy, we determine 500 and 300 (Fig. 4(a)). 2) We try to use *sigmoid*, *tanh* and *ReLU* as the activation function of hidden layer. It's obvious that *ReLU* has a great acceleration effect on the convergence compared with the other two (Fig. 4(b)). 3) We carried out experiments with *SGD*, *Momentum*, *RMSprop* and *Adam* respectively. As the gradient becomes sparser, *Adam* converges faster than the other three and the accuracy remains at a high level. So we choose the *Adam* optimizer to update the model parameters

(Fig. 4(c)). 4) We control the learning rate at 0.00001. Because the training process is more stable and the final accuracy rate is higher (Fig. 4(d)).

Through above steps, we obtain an optimal model. Finally, the model is trained in batches, the minimum batch is 20 flows every time, with a total of 20 iterations. The testing results are shown in Fig. 5, the accuracy increases from 75.45% to 98.55%, the recall reaches 98.41%, the precision reaches 98.76%, and the comprehensive evaluation index F_1 reaches 98.59%. Obviously, our model can accurately and quickly detect CC attack.

4.3 Robustness Analysis

In order to evaluate the robustness of the model, we adjust the ratio of attack traffic to non-CC attack traffic in training set to 1:1, 1:2, 1:3, 1:4, 1:8 and 1:11. For this, the number of CC-attack packets is controlled at 360,000 and the number of non-CC attack packets is changed several times. Besides, the test set contains 180,000 CC-attack packets and 720,000 non-CC attack packets.

The test results in different experimental environments are shown in Fig. 6. Through the experimental results, it can be found that the test results of the model under different environments are good. The accuracy is around 98%, indicating that the model has a good classification effect; the recall approaches 99%, indicating that the FN value is small, the model underreporting rate is low; the accuracy is about 95%, which means that the FC value is small and the model false positive rate is low; the F_1 value is about 97%, and the model comprehensive evaluation index is high, which can meet the needs of different practical application environments. In summary, the model has strong capability to detect attack traffic and can cope with more complex actual network environments.

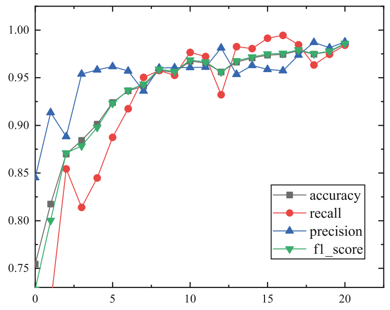


Fig. 5. The experimental results of FDPF model ('x' axis represents 'iterations')

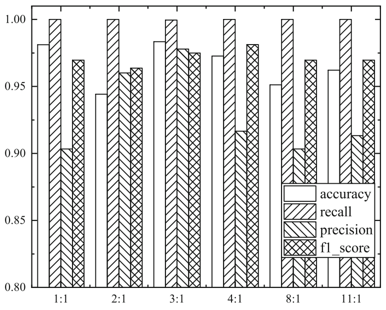


Fig. 6. The experimental results in different training set ratios

4.4 Comparison with Related Methods

In order to objectively evaluate the detection efficiency of our model, we choose classic and widely used related detection methods for comparative experiments.

Many good research, such as [7,17,23], used SVM classifier or random forest-based model for DDoS attack detection. Their experimental results show that these model has satisfying effect for their research problem. Therefore, we adopt SVM and random forest as comparison, and carry out the training and test under the same dataset used in this paper.

From the perspective of algorithm time complexity, the time complexity of the model used in this paper is $O(e * H * (K + d) * N_{sample})$ [6], where e is the number of network model training cycles; H is the number of hidden layers; K is the number of output layer nodes; d is the number of input layer nodes. The time complexity of Random Forest is $O(N_{sample} * N_{feature} * logN_{sample})$ [12], where N_{sample} is the number of samples, $N_{feature}$ is the dimension of features. The time complexity of SVM is $O(N^2_{Hessian}) + O(N_{sample}) + O(N_{sample} * N_{feature} * N_{Hessian})$ [15], where $N_{Hessian}$ is the number of matrix rows after feature matrixing, equals to N_{sample}, $O(N_{sample} * N_{feature} * N_{Hessian})$ is the time complexity of an iterative process, $O(N_{sample})$ is the time complexity of storing intermediate results, $O(N^2_{Hessian})$ is the time complexity of storing feature matrix. Through this comparison, it can be found that when the dataset is large, the time complexity of the Random Forest and SVM algorithms will increase exponentially, and the training cost of the model will increase greatly; in the case of the same amount of data, SVM needs to spend more time to store data matrices and intermediate results.

Table 2. Evaluation index and speed comparison

	SVM-RBF	Random forest	FDPD model
Accuracy	95.00%	94.89%	**98.55%**
Recall	88.89%	93.00%	**98.42%**
Presicion	96.43%	95.00%	**98.76%**
F_1	94.12%	94.00%	**98.59%**
Speed	7.96 Mb/s	5.78 Mb/s	**137.37 Mb/s**

The SVM model, the Random Forest model, and our proposed model are trained on our dataset. Experimental comparison results are shown in Table 2, and it can be found that the accuracy of the proposed model is 3% higher than SVM and Random Forest-based malicious traffic detection model. Apart from this, the test speed of our model is nearly 17 times faster than the related machine learning model.

5 Discussion

Due to the changing attack patterns and limited traffic collection methods, the traffic contained in the dataset for experiments can't cover all the CC attack variants, so the trained model may have a unsatisfactory effect when detecting unknown CC attack traffic. However, if the training data is sufficient enough, the model proposed in this paper can detect more attack traffic efficiently, because it has good data-preprocessing ability and self-learning ability. The FDPD model can adequately learn the features implied in the traffic and use them for classification. Furthermore, the reason that we don't use other neural networks, such as CNN, RNN, is to improve the efficiency of training and detection while ensuring the effect. Since the dimension of the input after preprocessing is not very high and our model shows good result, it is not necessary to spend extra designing and training costs to build overly complex models.

6 Conclusion and Future Work

In order to cope with CC attack, we proposed a packet Field Differentiated Preprocessing and Deep neural network (FDPD) model to detect CC attack traffic. And we collected a fresh dataset that contains 7.92 million packets to validate our model. In the design of FDPD model, efficient strategies are adopted to prevent overfitting problem of deep learning. In the preprocessing phase, we combine the specific meaning of each field of the packet to drop misleading field such as 'IP' and useless fields such as 'reserved'. In the model training phase, a dropout strategy is adopted for each hidden layer. The experimental results show that the accuracy of FDPD model reaches 98%. In the same training environment, our model was compared with the SVM-based and Random Forest-based traffic detection models. The accuracy is increased by 3%, and the speed is increased by 17 times.

In the future, we will improve in the following aspects: 1) Train a more effective attack traffic detection model; 2) Enrich our dataset.

References

1. Abdulaziz, A., Shahrulniza, M.: Cloud-based DDoS http attack detection using covariance matrix approach. J. Comput. Netw. Commun. **2017**(38), 1–8 (2017)
2. Adi, E., Baig, Z., Hingston, P.: Stealthy denial of service (DoS) attack modelling and detection for http/2 services. J. Netw. Comput. Appl. **91**, S1084804517301637 (2017)
3. Xiao, B., Chen, W., He, Y., Sha, E.M.: An active detecting method against SYN flooding attack. In: International Conference on Parallel & Distributed Systems (2005)
4. Cheng, R., Xu, R., Tang, X., Sheng, V.S., Cai, C.: An abnormal network flow feature sequence prediction approach for DDoS attacks detection in big data environment. Comput. Mater. Continua **55**(1), 095–095 (2018)
5. Douglas, D., Santanna, J.J., Schmidt, R.D.O., Granville, L.Z., Pras, A.: Booters: can anything justify distributed denial-of-service (DDoS) attacks for hire? J. Inf. Commun. Ethics Soc. **15**(1), 90–104 (2017)
6. Alpaydin, E.: Introduction to Machine Learning. MIT press, Cambridge (2009)
7. Idhammad, M., Afdel, K., Belouch, M.: Detection system of HTTP DDoS attacks in a cloud environment based on information theoretic entropy and random forest. In: Security and Communication Networks (2018)

8. Kumar, D., Rao, C.G.: Leveraging big data analytics for real-time DDoS attacks detection in SDN. Int. J. Res. Eng. Appl. Manag. IJREAM **04**(02), 677–684 (2018)
9. Li, J., Yun, X., Li, S., Zhang, Y., Xie, J., Fang, F.: A HTTP malicious traffic detection method based on hybrid structure deep neural network. J. Commun. **40**(01), 28–37 (2019)
10. Lin, Y.H., Kuo, J.J., Yang, D.N., Chen, W.T.: A cost-effective shuffling-based defense against HTTP DDoS attacks with sdn/nfv. In: IEEE International Conference on Communications, pp. 1–7 (2017)
11. Misra, S., Krishna, P.V., Abraham, K.I., Sasikumar, N., Fredun, S.: An adaptive learning routing protocol for the prevention of distributed denial of service attacks in wireless mesh networks. Comput. Math. Appl. **60**(2), 294–306 (2010)
12. Idhammad, M., Afdel, K., Belouch, M.: Detection system of HTTP DDoS attacks in a cloud environment based on information theoretic entropy and random forest. Secur. Commun. Netw. **2018**(1263123), 1–13 (2018)
13. Moore, D., Voelker, G.M., Savage, S.: Inferring internet denial-of-service attack. In: Conference on Usenix Security Symposium (2001)
14. Sahoo, K.S., Tiwary, M., Sahoo, S., Nambiar, R., Sahoo, B., Dash, R.: A learning automata-based DDoS attack defense mechanism in software defined networks. In: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, pp. 795–797 (2018)
15. Sahu, S.K., Jena, S.K.: A multiclass SVM classification approach for intrusion detection. In: International Conference on Distributed Computing & Internet Technology (2016)
16. Singh, K., Singh, P., Kumar, K.: Application layer HTTP-get flood DDoS attacks: research landscape and challenges. Comput. Secur. **65**, 344–372 (2017)
17. Singh, K., Singh, P., Kumar, K.: User behavior analytics-based classification of application layer HTTP-GET flood attacks. J. Netw. Comput. Appl. **112**, 97–114 (2018)
18. Sree, T.R., Bhanu, S.M.S.: Hadm: detection of HTTP get flooding attacks by using analytical hierarchical process and dempster-shafer theory with mapreduce. Secur. Commun. Netw. **9**(17), 4341–4357 (2016)
19. Su, L., Yao, Y., Li, N., Liu, J., Lu, Z., Liu, B.: Hierarchical clustering based network traffic data reduction for improving suspicious flow detection. In: IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 744–753 (2018)
20. Tan, M.: Research and implementation of DDoS attack detection based on machine learning in distributed environment. Ph.D. thesis (2018)
21. Wang, A., Mohaisen, A., Chang, W., Chen, S.: Delving into internet DDoS attacks by botnets: Characterization and analysis. In: IEEE/IFIP International Conference on Dependable Systems & Networks, vol. 26, pp. 2843–2855 (2015)
22. Xie, Y., Yu, S.Z.: A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors. IEEE/ACM Trans. Netw. **17**(1), 54–65 (2009)
23. Ye, J., Cheng, X., Zhu, J., Feng, L., Song, L.: A DDoS attack detection method based on SVM in software defined network. In: Security and Communication Networks (2018)
24. Liu, Z., Zhang,B.: Self-learning application layer DDoS detection method based on improved AP clustering algorithm (2018)