



Article

Efficient Detection of Link-Flooding Attacks with Deep Learning

Chih-Hsiang Hsieh ¹, Wei-Kuan Wang ², Cheng-Xun Wang ³, Shi-Chun Tsai ^{3,*} and Yi-Bing Lin ³

¹ Institute of Data Science and Engineering, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan; starlit.cs09@nycu.edu.tw

² Institute of Network Engineering, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan; wkwang.cs09@nycu.edu.tw

³ Institute of Computer Science and Engineering, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan; mark870806.cs09@nycu.edu.tw (C.-X.W.); liny@nctu.edu.tw (Y.-B.L.)

* Correspondence: sctsai@nycu.edu.tw

Abstract: The DDoS attack is one of the most notorious attacks, and the severe impact of the DDoS attack on GitHub in 2018 raises the importance of designing effective defense methods for detecting this type of attack. Unlike the traditional network architecture that takes too long to cope with DDoS attacks, we focus on link-flooding attacks that do not directly attack the target. An effective defense mechanism is crucial since as long as a link-flooding attack is undetected, it will cause problems over the Internet. With the flexibility of software-defined networking, we design a novel framework and implement our ideas with a deep learning approach to improve the performance of the previous work. Through rerouting techniques and monitoring network traffic, our system can detect a malicious attack from the adversary. A CNN architecture is combined to assist in finding an appropriate rerouting path that can shorten the reaction time for detecting DDoS attacks. Therefore, the proposed method can efficiently distinguish the difference between benign traffic and malicious traffic and prevent attackers from carrying out link-flooding attacks through bots.

Keywords: distributed denial of service (DDoS) attack; link-flooding attack (LFA); deep learning (DL); software defined networking (SDN)



Citation: Hsieh, C.-H.; Wang, W.-K.; Wang, C.-X.; Tsai, S.-C.; Lin, Y.-B. Efficient Detection of Link-Flooding Attacks with Deep Learning. *Sustainability* **2021**, *13*, 12514. <https://doi.org/10.3390/su132212514>

Academic Editor: Abdollah Shafieezadeh

Received: 27 September 2021
Accepted: 9 November 2021
Published: 12 November 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Various network attacks had caused severe damage to the Internet community in the real-world over recent years [1–6]. The link-flooding attack is a kind of distributed denial of service (DDoS) attack that does not directly attack the target devices but congests the target link with a large number of low-speed flows to reduce the connectivity between end devices, so it is difficult to detect with the intrusion detection system or firewall. Thus, it draws the attention of designing an effective way to detect link-flooding attacks quickly to protect the Internet infrastructure from danger. Rasool et al. [2] gave a pretty nice survey on this issue, and they suggested a possible solution via the machine learning approach. We propose a deep learning method to help resolve the link-flooding problem based on software-defined networking.

Link flooding attack is one of the most troublesome attacks in modern Internet society [2]. It utilizes a large quantity of seemingly normal traffic to congest the target link in the network. It can be hard to identify the low-rate network traffics as malicious flows since they use normal flows to avoid being prone to distinguish. However, in practice, an attacker may carry out link-flooding attacks through many bots with low-rate flows to tremendously degrade the connection between end devices with low costs.

Due to the flexible architecture of the software-defined networking (SDN) environment, designing an SDN-based defense mechanism to deal with link-flooding attacks can be a good choice. SDN is a relatively new type of network paradigm that separates the

control plane from the data plane. SDN's programmable features can help quickly respond to threats that are usually difficult to counteract in the traditional network framework.

We use rerouting techniques to relieve the link-flooding attack by virtually increasing the bandwidth of the target link in the network [7,8], which is also easier to implement with SDN architecture. A fast defense workflow is crucial when addressing the link-flooding attacks problems to mitigate the harm to the Internet. A state-of-the-art link-flooding attacks defense mechanism, SPIFFY, develops a fast traffic-engineering algorithm to increase the bandwidth of the target link with a binary search procedure [7]. However, binary search is a deterministic procedure since it always searches the middle m value without leveraging different situations in the network. We combine the SDN-based defense mechanism with deep learning methods used to adjust the bandwidth to respond quickly to link-flooding attacks automatically. A CNN architecture aims to predict the probability of all candidate m values. That is, searching more adaptively and efficiently.

We leverage its flexibility to construct a network topology for simulation to better plan and manage network traffic and realize network security monitoring in the SDN environment. We can detect malicious attacks from attackers through rerouting technology and a deep learning model that assists in finding an appropriate rerouting path and shortens the reaction time. In addition, our method can efficiently distinguish between benign traffic and malicious traffic, and prevent link flooding attacks through bots.

2. Related Work

Recently, machine learning and deep learning approaches have been widely used in network security for optimization and classification due to the rapid growth in cloud computing and big data [2]. For instance, Lal et al. [9] introduced a framework that can guarantee the security and robustness of those artificial intelligence approaches by preserving the classification with correct labeling. In addition, in network security, an intrusion detection system (IDS) is an application that can effectively detect malicious behaviors. Mena [10] proposed that IDS can be categorized into two groups: misuse intrusion and anomaly intrusion, by its detection patterns. Precisely, the misuse intrusion approach will match the network traffic to all known intrusion types. In contrast, the anomaly intrusion will analyze network traffic behaviors and compare them to past behaviors to determine an attack.

The vast growth of the total number of Internet of Things (IoT) devices also makes DDoS attacks more lethal by IoT malware, such as Mirai [11], and there have been some profound research dedicated to dealing with these attacks. For example, Ahmed et al. [12] proposed a novel network traffic classification network based on application fingerprinting, classifying DDoS attacks and normal traffic with an accuracy higher than 97% in 5 different real-world datasets. In addition, Alharbi et al. [13] proposed a local-global best Bat Algorithm for Neural Networks (LGBA-NN) to classify ten different botnet attacks. These two studies are so-called misuse intrusion-based methods making use of network traffic classification techniques. However, the abnormal behaviors must be defined firstly for misuse intrusion-based methods. Otherwise, any unknown behavior will be considered normal.

In addition, SDN-based techniques are new approaches to network management that improve the performance in traditional attack mitigation and detection [14]. However, some attacks may also specifically have a devastating effect on SDN and increase risks different from traditional network architecture. Therefore, there are also some IDS designed in the SDN-based environment. For example, Wang et al. [15] proposed a Safe-Guard scheme to protect the control plane from DDoS attacks. The core concept is deploying multiple controllers and defending dynamically to reduce the impact of DDoS attacks on the control plane. However, the overhead of synchronization of multiple controllers is not yet moderately considered and raises the importance of efficient synchronization. Furthermore, Nam et al. [16] proposed two DDoS attack detection techniques to classify normal and DDoS network traffic based on self-organizing map (SOM), an unsupervised learning algorithm mapping from the high-dimensional space to two-dimensional space. However,

previous studies have shown that SOM-based IDS comes across low performance since SOM is insufficient for the topology representation with unbalanced distributions [17,18].

In recent years, many studies on mitigating DDoS attacks have explored the effectiveness of bandwidth reservation techniques. Zhang et al. [19] proposed a spatio-temporal heterogeneous bandwidth allocation (STBA) mechanism to mitigate DDoS attacks by enabling domain-level resource management. Wang et al. [20] proposed LFADefender, a system based on SDN to defend the link flooding attack. The method involves some critical technologies, especially the congestion link detecting and rerouting. Tran et al. [21] presented an in-depth analysis of the feasibility of the routing around congestion (RAC) defense and pointed out the challenges for the rerouting approach.

SPIFFY [7] is an anomaly-based intrusion detection approach to detect the link-flooding attack by temporary bandwidth expansion (TBE). The core idea is that after increasing adequate bandwidth with a factor m by rerouting, legitimate users will also increase their sending rate approximately m times. At the same time, the bots used by attackers will not work due to the cost consideration. The different behavior patterns can help the system correctly identify link-flooding attacks by performing the rate-change test. Moreover, SPIFFY used a greedy method to find an approximate solution for the effective factor m and the rerouting path through the target link with a binary search procedure. In this way, SPIFFY can find an acceptable solution of m with some searching cost and react quickly to the link-flooding attack. To further decrease the cost of detecting the link-flooding attack, we design a model combining a deep learning approach based on state-of-the-art studies. Thus, we replace the binary search with a deep learning approach.

3. Method

Our research aims to use deep learning methods to solve the excessive response time of traditional networks to malicious attacks. Furthermore, SDN's flexibility allows better planning and managing network behaviors and finding abnormal users, identified as the link-flooding attackers by rerouting. Eventually, we can reduce the time cost when recognizing the link-flooding attackers with the deep learning approach.

We first generated a link-flooding attack dataset related to our research goals—using such a dataset to train the deep learning models and learn the value of m required for rerouting, thereby shortening the reaction time for detecting DDoS attacks. Furthermore, our method can efficiently distinguish between benign and malicious traffic to prevent link flooding attacks through bots.

We implemented a defense mechanism for network security monitoring based on SDN technology. With a deep learning model trained in advance, we only need to analyze the current network traffic to quickly calculate the rerouting path and perform a rate-change test after rerouting to identify the attackers. Figure 1 shows the structure of our method.

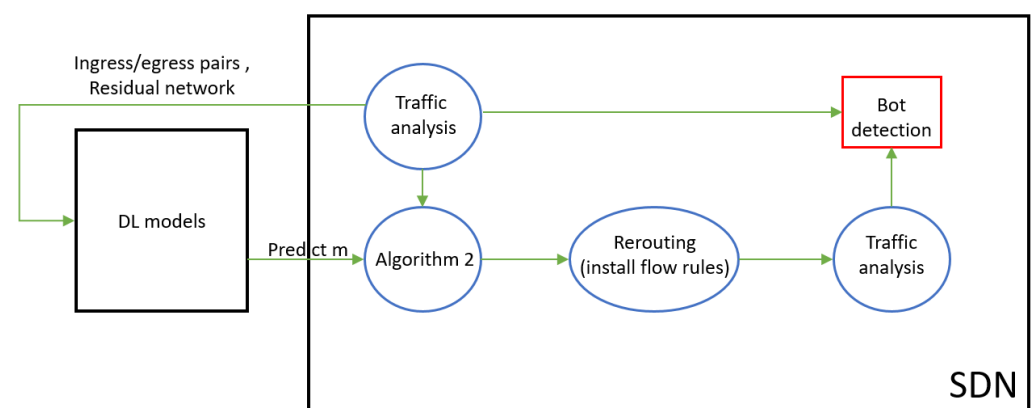


Figure 1. The structure of our method.

SPIFFY [7] has an algorithm to find a rerouting path when given the target link, ingress and egress pairs crossing the target link, residual network, and the magnification of bandwidth expansion if the path exists. Our pre-trained DL model will predict the magnitude of bandwidth expansion, which is a value of m . Then, when there is a link-flooding attack, do traffic analysis to get all sending rates of flows. By combining the DL model with SPIFFY's algorithm, the controller in our SDN environment will install flow rules into switches to perform rerouting. Then, do traffic analysis again, and perform a rate-change test to identify the bots used in link-flooding attacks.

As shown in Algorithm 1, at the start, the controller knows the overall topology and installs flow rules to the switches. Before the packet transmission starts, we use breadth-first search to pre-calculate the shortest path of all source-destination pairs and let all the switches on the path know how they should handle the packets. In a typical network, the switches usually forward packets according to their destinations, but in our method, we need to know the transmission rate of the source-destination pairs. Therefore, dividing into different flow rules helps us capture the usage rate of each flow rule. After installing all flow rules, if a packet is transmitted to the switch, it will check the flow rule. Then, the packet will be forwarded by following the matching flow rule, and the match bytes of which flow rule will be accumulated. Since there is no need to transmit the packet to the controller under normal circumstances, the speed will be faster than those sent to the controller.

Algorithm 1 An algorithm for initial flow rules

Require: H : hosts, S : switches, L : links

```

1: for all  $h_1 \in H$  do
2:   Applying breadth-first search (BFS) algorithm from  $h_1$  to all the others hosts
3:   for all  $h_2 \neq h_1 \in H$  do
4:     for all switch  $s$  on the paths from  $h_1$  to  $h_2$  do
5:       Install flow rule on switch  $s$ 
6:     end for
7:   end for
8: end for

```

As shown in Algorithm 2, when the system detects a flooded link, the controller starts to calculate the short-term transmission rate of all source-destination pairs and the residual network. For all the source-destination pairs flowing through the link, we need to recalculate whose paths. We expect the transmission rate of those pairs to be increased to m times on the new paths. In Algorithm 2, we select the pairs in random order. When recalculating the path for each selected pair, the topology consists of the links with capacity at least m times the transmission rate. We use breadth-first search to calculate the route again and then install new flow rules into switches. For normal TCP users, all the link capacities on their new routes are at least m times their original transmission rate, which should be increased to at least m times in a short time. It only takes tens of seconds to distinguish between benign users and malicious users. After that, we restore the original flow rules and continue to monitor the next attack.

Given the usage amount of network traffic and the pairs of nodes having the target link, we train a CNN model to output the probabilities of different magnifications of network traffic. Our model contains two parts: preprocessing and model learning. We explain as follows.

Algorithm 2 An algorithm for temporary bandwidth expansion (TBE)**Require:**

ℓ_{tg} : The target link,
 P : All ingress/egress pairs (s, t) crossing ℓ_{tg} ,
 L_R : All links with residual bandwidth in the topology
 M : A fixed-size array predicted from Deep Learning model, where M_i has a higher probability to be the answer when i is smaller. We divided $[1, 6)$ into 16 equal parts, and M is a permutation of which.

```

1: procedure REROUTING( $P, L_R, M, \ell_{tg}$ )
2:    $l_b \leftarrow 1, u_b \leftarrow 6$     ▷ We use lower bound and upper bound to prevent redundant
   computation. That is, it is not necessary to test  $M_i$  if  $M_i$  is out of  $[l_b, u_b]$ .
3:   for  $i = 1$  to 16 do                                ▷ We will test  $M_i$  in order.
4:     if  $M_i < l_b$  or  $u_b < M_i$  then                  ▷  $M_i$  is out of the range, so skip it.
5:       continue
6:     end if
7:      $L_T \leftarrow L_R$                                 ▷ A temporary copy of  $L_R$  for computing
8:      $ok \leftarrow True$     ▷ A flag variable to record whether there is a new path for every
    $(s, t) \in P$ 
9:     while  $(\exists (s, t) \in P$  hasn't been selected) do
10:       $L_M \leftarrow \{\ell \mid \text{bandwidth of } \ell \geq \text{bandwidth of } (s, t) \times M_i, \ell \in L_T \setminus \ell_{tg}\}$ 
11:      do breadth-first search to find a path from  $s$  to  $t$  with edges  $L_M$ 
12:      if (There is a path from  $s$  to  $t$ ) then
13:        for all link  $\ell \in L_T$  on the path do
14:          bandwidth of  $\ell :=$  bandwidth of  $\ell - \text{bandwidth of } (s, t) \times M_i$     ▷
   Recalculate the residual bandwidth of  $L_T$ .
15:        end for
16:      else                                            ▷  $M_i$  is too large to be the answer.
17:         $ok = False$ 
18:        break
19:      end if
20:    end while
21:    if  $ok = True$  then
22:       $l_b := M_i$ 
23:    else
24:       $u_b := M_i$ 
25:    end if
26:  end for
27:  ▷  $l_b$  is the best  $m$  value. That is,  $l_b$  is large enough, and for each pair  $(s, t) \in P$ , there
   is a new path whose bottleneck bandwidth is at least  $l_b$  times of the bandwidth of  $(s, t)$ .
28:  for all  $(s, t) \in P$  do
29:    Install new flow rules into all the switches on the new path.
30:  end for
31: end procedure
  
```

3.1. Preprocessing

The available network is translated to an adjacency matrix $A \in R^{n \times n}$, where n is the number of switches in a given network topology. Let C_{ij} be the link capacity between switch i and switch j . Then, $0 \leq A_{ij} \leq C_{ij}$ represents the residual network between them.

CNN only accepts the fixed-size input while the number of ingress/egress pairs crossing the target link often varies. Therefore, we translate the pairs into a fixed-size matrix $B \in R^{n \times n}$, where B_{ij} represents the transmitting rate from switch i to switch j , and n is the number of switches in the network topology. Note that the bandwidth of flows with the same source and destination switches will be summed up to the corresponding matrix entry. CNN is a supervised learning model. In order to formulate our labels, we initiate the possible values of m based on some observations. It turns out that most of the values between 1 and 6 are sufficient. Additionally, we divide this interval, i.e., $[1, 6)$, into

16 bins of the same size, $B_1, \dots, B_k, k = 16$, and then the label of a sample is equal to ℓ if its m value is in the interval $[1 + (\ell - 1) \times \frac{1}{k}, 1 + \ell \times \frac{1}{k}]$. Therefore, we use our proposed model to classify the best value of m by 16 possible candidates in the interval of $[1, 6)$.

3.2. Model Learning

This model takes the matrices A and B generated from the pre-processing part as the input of CNN. Let the training samples be $X = \{x_i | i = 1, \dots, N\}$ and the corresponding labels $Y = \{y_i | i = 1, \dots, N\}$, where x_i is a training sample with two channels A and B , $y_i \in \mathbb{R}^k$ is a one-hot vector represents the ground-truth label of x_i , and N is the number of samples. Our CNN architecture is shown in Figure 2, which contains three hidden layers and one fully connected layer. Each hidden layer contains a convolution layer, ReLU as the activation function, and a max-pooling layer. The three convolution layers contain 8, 16, and 32 feature maps, respectively. The size of the kernel is 3×3 . In the training phase, we minimize the cross-entropy loss function. The form of the objective function is shown as follows: $Loss(X, Y, \theta) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^k y_{ij} \log(\text{softmax}(\hat{y}_{ij}))$, where N is the sample size, k is the number of bins, $\hat{y}_i \in \mathbb{R}^k$ is the output of our non-linear model (i.e., the CNN architecture), and θ is the set of parameters of our proposed model.

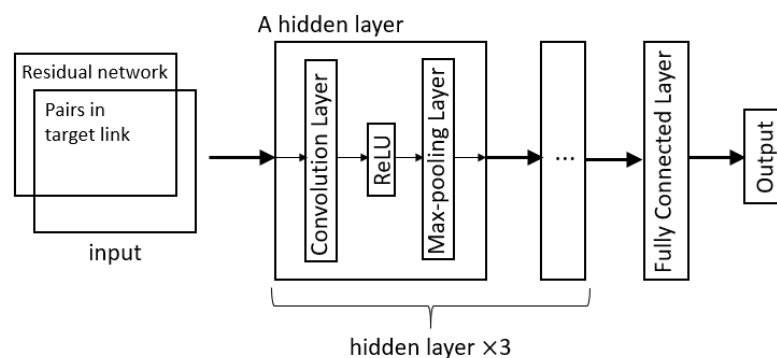


Figure 2. Our CNN architecture.

4. Experiment

Our proposed model aims to improve the approach proposed in SPIFFY by combining a CNN model. That is, reducing the number of searching times to find a proper factor m for rerouting. Thus, we create a synthetic dataset and compare our results with SPIFFY. In our experiment environment, we construct a topology with 10 switches and some links, as in Figure 3, to evaluate our method. The flow utilization and the target link are generated randomly to construct the dataset with approximately 45k different situations. In addition, the proportion of the train-test split is 2:1. Table 1 provides the dataset information for samples of each class.

Table 1. Number of samples for each class in the dataset.

| | | | | | | | | |
|-----|------|--------|-------|--------|------|--------|-------|--------|
| m | 1 | 1.3125 | 1.625 | 1.9375 | 2.25 | 2.5625 | 2.875 | 3.1875 |
| # | 2510 | 5528 | 476 | 3884 | 4336 | 1489 | 2547 | 5236 |
| m | 3.5 | 3.8125 | 4.125 | 4.4375 | 4.75 | 5.0625 | 5.375 | 5.6875 |
| # | 2485 | 1606 | 3247 | 1369 | 5854 | 1316 | 733 | 2638 |

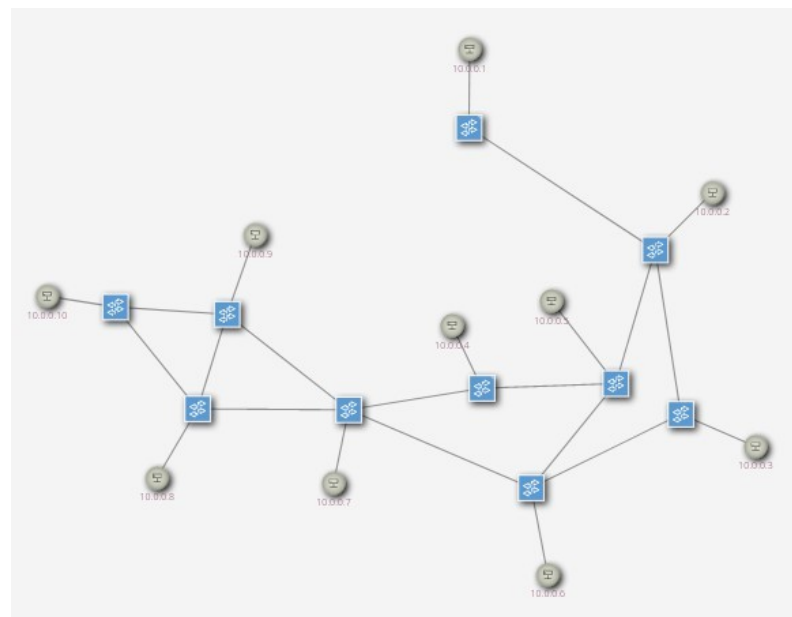


Figure 3. Topology. There are 10 switches and links (blue square points). Grey circles indicate the hosts.

The range of m value is from 1 to 6, and we partition the interval $[1, 6)$ into 16 equal bins. We continuously test m different values sorted in descending order with their probability given by our CNN architecture. In addition, the top- k accuracy is an evaluation metric that computes the predicted label's accuracy among the top k predicted labels. Therefore, we adopt top- k accuracy to evaluate the performance of our proposed model. Figure 4 shows the top- k accuracy of the CNN architecture. As the number of training epochs increases, the top- k accuracy can be improved; the higher the accuracy, the faster the execution speed of finding a rerouting path in the SDN environment. The accuracy can reach over 90% for the top 3 candidates of the best m value derived from CNN.

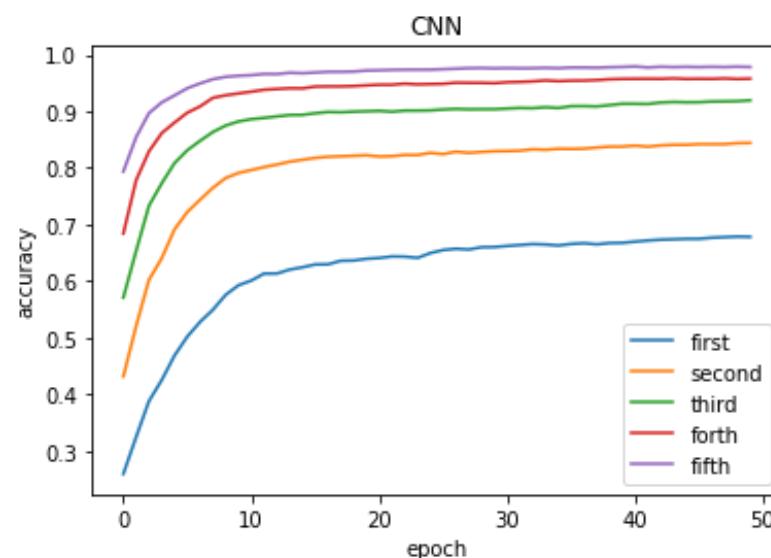


Figure 4. Accuracy for the first k candidates of the value of m .

We demonstrate our idea with a simple deep learning model to improve SPIFFY's performance since they are easier to design and understand. As a result, we have tried CNN and DNN architectures rather than other more complicated models such as RNN or LSTM. Table 2 provides the top- k accuracy for both CNN and DNN architectures, where

DNN contains three fully-connected layers with ReLU functions and there are 80, 100, and 16 neurons in three layers, respectively. According to our experiment results provided in Table 2, CNN has a better performance than DNN, so we take CNN architecture as the deep learning model in our proposed approach.

Table 2. The comparison of top- k accuracy for different deep learning models.

| Top- k Accuracy(%) | $k = 1$ | $k = 2$ | $k = 3$ | $k = 4$ | $k = 5$ | $k = 6$ |
|----------------------|---------|---------|---------|---------|---------|---------|
| CNN | 61.95 | 78.03 | 85.60 | 90.18 | 93.24 | 95.03 |
| DNN | 62.09 | 77.83 | 84.86 | 89.25 | 92.64 | 94.82 |

The binary search used in SPIFFY [7] aims to find the best m value for temporary bandwidth expansion by rerouting. Since the range of m is divided into 16 equal parts, binary search always takes four attempts to look for the best m value. Binary search will only search for the value greater or less than the current value of m . Similarly, we do not need to make unnecessary attempts in our proposed architecture, even if the candidate m value has a higher probability of becoming the best solution. For instance, if there is a rerouting path with $m = 3.5$, it implies that the path with $m = 2.25$ also exists without additional calculation. Figure 5 shows the distribution of times required for CNN to obtain the best m value. On average, CNN requires 3.17 times, which is better than the binary search approach. Thus, using the deep learning method to improve the original binary search method [7] can reduce the average number of searches. Except for very few cases, most of them only require a relatively small number of iterations.

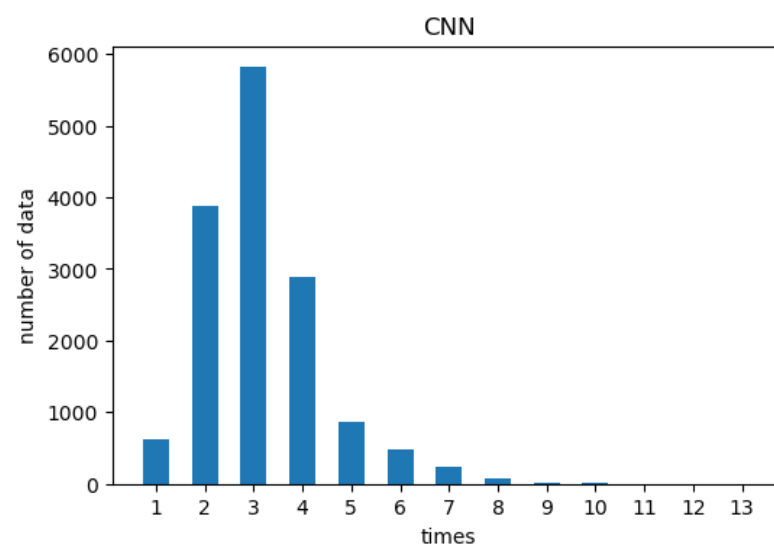


Figure 5. Numbers of searching for finding the best m value.

Moreover, Figure 6 shows the result of setting a threshold to further reduce the number of searches by considering the top- k accuracy of the CNN architecture. Since the top- k accuracy shows that there is a 95% chance that the best m value will fall in the first six candidate m values, most of the cases do not need to search all the possible m values. Experimental results show that the average number of searches required for CNN is only 2.56 times after adding the user-defined threshold. Note that we should consider whether this threshold does decrease the number of searching times significantly while keeping high accuracy.

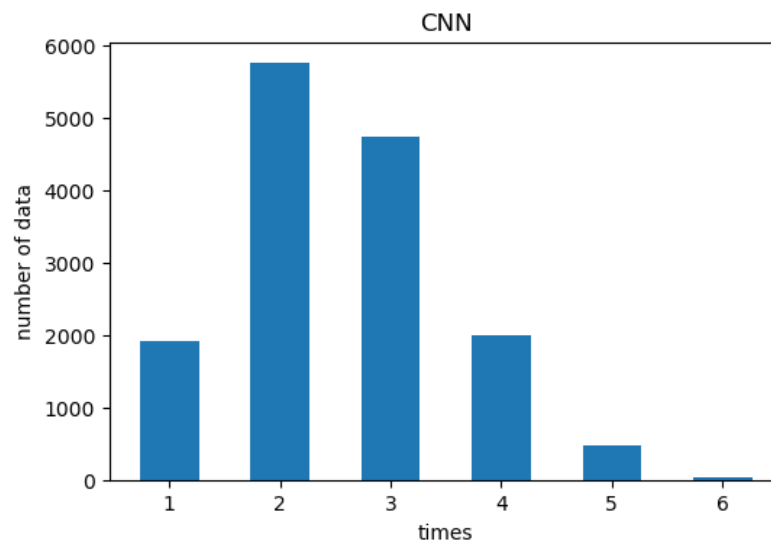


Figure 6. Numbers of searching for finding the best m value with a threshold.

Rasool et al. [2] proposed eight quality metrics for systematically judging the pros and cons of different link-flooding attack mitigation and detection approaches from an objective point of view. According to the basis of those performance metrics, we have improved SPIFFY [7] in terms of detection time and approach used, whereas maintaining the advantages of SPIFFY in other aspects, such as detection accuracy and scalability.

Specifically, SPIFFY utilizes binary search, which is a traditional technique. By contrast, we apply a DL-based technique for optimization and thus improve the performance. Rasool et al. [2] also illustrated the significance of using ML-based detection and mitigation techniques as an effective first-line of defense against link-flooding attacks. As we have already mentioned and discussed earlier, Figures 5 and 6 have shown that our proposed method can reduce the average number of searches for finding an effective m factor of TBE. Consequently, our approach takes less time for detecting bots used in launching link-flooding attacks and is more time-efficient than SPIFFY.

Additionally, Figure 6 shows that we can find an appropriate m value using a pre-defined threshold in most cases. Employing this threshold in our CNN model will maintain the detection accuracy of SPIFFY. As we develop our approach in the manner of deep learning, the scalability of our link-flooding attack detection solution can also be guaranteed, which means it can be deployed in a large-scale network environment.

5. Conclusions

SPIFFY [7] was proposed to defend against link-flooding attacks. However, it is time-consuming to recalculate the feasible paths for all the source-destination pairs that pass through the target link. Furthermore, it must be calculated online, unlike the initial transmission path that can be pre-calculated. Therefore, binary search is used in SPIFFY to search for the appropriate m value. However, the paths must be recalculated for every attempt of m value because the graph is associated with m . Therefore, we propose a deep learning method to reduce the number of search attempts, achieving over 95% top-6 accuracy. Our method generates the probability of the possible distribution of the m value through the model. We try from the m value with high probability first and then use the upper and lower bounds to avoid the impossible m values. On average, we can find the appropriate m value with fewer attempts of 2.56 times with a pre-defined threshold, which is better than 4 attempts with binary search in SPIFFY. As a consequence, in a real-time network environment, it can identify malicious attackers and reduce the impact of malicious attacks.

Author Contributions: Conceptualization, S.-C.T., C.-H.H., W.-K.W. and C.-X.W.; methodology, C.-H.H. and W.-K.W.; software, C.-H.H., W.-K.W. and C.-X.W.; writing—original draft preparation, C.-H.H., W.-K.W. and C.-X.W.; writing—review and editing, S.-C.T. and Y.-B.L.; supervision, S.-C.T.; project administration, S.-C.T.; funding acquisition, S.-C.T. and Y.-B.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in parts by the Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education in Taiwan, Ministry of Science and Technology under Grants 109-3111-8-009-002, 109-2221-E-009-087-MY2, 109-2218-E-009-010, 108-2221-E-009-051-MY3 and 105-2221-E-009-103-MY3; and in part by the Ministry of Economic Affairs under Grant 109-EC-17-A-02-S5-007.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

| | |
|------|-------------------------------|
| DDoS | Distributed denial-of-service |
| IoT | Internet of things |
| SDN | Software-defined networking |
| IDS | Intrusion detection system |
| DL | Deep learning |
| BFS | Breadth-first search |
| TBE | Temporary bandwidth expansion |
| CNN | Convolutional neural network |
| ReLU | Rectified Linear Unit |
| RNN | Recurrent neural network |
| LSTM | Long short-term memory |

References

- Gupta, S.; Grover, D.; Singla, J. Characterization of Flash events and DDoS Attacks: A Survey. In Proceedings of the 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 19–20 March 2021; Volume 1, pp. 1442–1447.
- Rasool, R.; Wang, H.; Ashraf, U.; Ahmed, K.; Anwar, Z.; Rafique, W. A survey of link flooding attacks in software defined network ecosystems. *J. Netw. Comput. Appl.* **2020**, *172*, 102803. [CrossRef]
- Studer, A.; Perrig, A. *Computer Security—ESORICS 2009*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 37–52, ISBN 978-3-642-04443-4.
- Kang, M.S.; Lee, S.B.; Gligor, V.D. The crossfire attack. In Proceedings of the 2013 IEEE Symposium on Security and Privacy, Berkeley, CA, USA, 19–22 May 2013; pp. 127–141.
- DAN GOODIN. How Extorted e-mail Provider Got Back Online after Crippling DDoS Attack. Available online: <https://arstechnica.com/information-technology/2015/11/how-extorted-e-mail-provider-got-back-online-after-crippling-ddos-attack/> (accessed on 21 August 2021).
- Can a DDoS break the Internet? Sure. . . Just Not All of It. Available online: <https://arstechnica.com/information-technology/2013/04/can-a-ddos-break-the-internet-sure-just-not-all-of-it/> (accessed on 21 August 2021).
- Kang, M.S.; Gligor, V.D.; Sekar, V. SPIFFY: Inducing Cost-Detectability Tradeoffs for Persistent Link-Flooding Attacks. In Proceedings of the NDSS, San Diego, CA, USA, 21–24 February 2016; Volume 1, pp. 53–55.
- Xie, L.; Ding, Y.; Yang, H.; Hu, Z. Mitigating LFA through segment rerouting in IoT environment with traceroute flow abnormality detection. *J. Netw. Comput. Appl.* **2020**, *164*, 102690. [CrossRef]
- Lal, S.; Rehman, S.U.; Shah, J.H.; Meraj, T.; Rauf, H.T.; Damaševičius, R.; Abdulkareem, K.H. Adversarial Attack and Defence through Adversarial Training and Feature Fusion for Diabetic Retinopathy Recognition. *Sensors* **2021**, *21*, 3922. [CrossRef] [PubMed]
- Mena, J. *Investigative Data Mining for Security and Criminal Detection*; Butterworth-Heinemann: Oxford, UK, 2003; ISBN 978-0750676137.
- Tahaei, H.; Afifi, F.; Asemi, A.; Zaki, F.; Anuar, N.B. The rise of traffic classification in IoT networks: A survey. *J. Netw. Comput. Appl.* **2020**, *154*, 102538. [CrossRef]

12. Ahmed, M.E.; Ullah, S.; Kim, H. Statistical application fingerprinting for DDoS attack mitigation. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 1471–1484. [[CrossRef](#)]
13. Alharbi, A.; Alosaimi, W.; Alyami, H.; Rauf, H.T.; Damaševičius, R. Botnet Attack Detection Using Local Global Best Bat Algorithm for Industrial Internet of Things. *Electronics* **2021**, *10*, 1341. [[CrossRef](#)]
14. Feng, B.; Zhang, H.; Zhou, H.; Yu, S. Locator/identifier split networking: A promising future Internet architecture. *J. IEEE Commun. Surv. Tutor.* **2017**, *19*, 2927–2948. [[CrossRef](#)]
15. Wang, Y.; Hu, T.; Tang, G.; Xie, J.; Lu, J. SGS: Safe-guard scheme for protecting control plane against DDoS attacks in software-defined networking. *IEEE Access* **2019**, *7*, 34699–34710. [[CrossRef](#)]
16. Nam, T.M.; Phong, P.H.; Khoa, T.D.; Huong, T.T.; Nam, P.N.; Thanh, N.H.; Thang, L.X.; Tuan, P.A.; Dung, L.Q.; Loi, V.D. Self-organizing map-based approaches in DDoS flooding detection using SDN. In Proceedings of the 2018 International Conference on Information Networking (ICOIN), Chiang Mai, Thailand, 10–12 January 2018; pp. 249–254.
17. Villmann, T.; Der, R.; Herrmann, M.; Martinetz, T.M. Topology preservation in self-organizing feature maps: Exact definition and measurement. *IEEE Trans. Neural Netw.* **1997**, *8*, 256–266. [[CrossRef](#)] [[PubMed](#)]
18. Ritter, H.; Martinetz, T.; Schulten, K. *Neural Computation and Self-Organizing Maps: An Introduction*; Addison-Wesley: New York, NY, USA, 1992; pp. 141–161, ISBN 978-0201554427.
19. Zhang, X.; Xie, L.; Yao, W. Spatio-temporal heterogeneous bandwidth allocation mechanism against DDoS attack. *J. Netw. Comput. Appl.* **2020**, *162*, 102658. [[CrossRef](#)]
20. Wang, J.; Wen, R.; Li, J.; Yan, F.; Zhao, B.; Yu, F. Detecting and mitigating target link-flooding attacks using SDN. *IEEE Trans. Dependable Secur. Comput.* **2018**, *16*, 944–956. [[CrossRef](#)]
21. Tran, M.; Kang, M.S.; Hsiao, H.C.; Chiang, W.H.; Tung, S.P.; Wang, Y.S. On the feasibility of rerouting-based DDoS defenses. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; Volume 5, pp. 1169–1184.