Detecting and Mitigating Target Link-Flooding Attacks Using SDN

Juan Wang, *Member, IEEE,* Ru Wen, Jiangqi Li

Abstract—DDoS attacks have caused very serious damage to enterprise networks. Recently, a new kind of DDoS attack called link-flooding attack (LFA), has surfaced and is already being used by attackers to flood and congest network critical links. LFA is very difficult to detect since adversaries often utilize large-scale legitimate low-speed flows and rolls target links to isolate target areas for launching attacks. To address such a critical security problem, we design and implement a novel LFA defense system called LFADefender that leverages some key features, such as programmability, network-wide view, and flow traceability, of an emerging network technology, Software-Defined Networking (SDN), to effectively detect and migrate LFA. In LFADefender, we propose a LFA target link selection approach and design a LFA congestion monitoring mechanism to effectively detect LFA. In addition, we present a multiple optional paths rerouting method to temporarily mitigate links congestion caused by LFA. We further propose a malicious traffic blocking approach to radically mitigate LFA. Our evaluation results show that LFADefender can accurately detect and rapidly mitigate LFA, but only imposes minimal overhead in the communication channels between network controllers and data planes.

Index Terms—DDoS, Link-Flooding Attack, SDN, Flow tracing, Bot identification.

1 Introduction

DDoS attacks are one of the most serious threats to network security. According to a survey [1] by Black Lotus, a famous DDoS protection vendor, 64% platform providers, 66% managed solution providers, and 66% VoIP service providers have been affected by DDoS attacks. Recently, a new kind of DDoS attack, target link-flooding attack (LFA), has been introduced [2], [3]. In particular, it has already been used by attackers to congest critical links of four major Internet exchange points in Europe and Asia [4]. In emerging cloud environments, LFA is also effective because many service providers share the same cloud infrastructure. Therefore, attackers can leverage the shared servers as decoy servers to flood and congest share links to attack other service providers.

LFA cuts off critical infrastructure through flooding and congesting critical network links of a target area. It presents two unique characteristics that make it very hard to detect and mitigate [3]. First, LFA uses large-scale legitimate low-speed flows to launch attacks. Therefore, traditional anomaly-based IDSes and signature-based IDSes cannot effectively identify those attacks. Secondly, LFA is adaptive and often changes the target links in real time, easily bypassing the traditional defence mechanisms which are often deployed in fixed locations of networks. Attackers usually select the links with high flow density firstly, then coordinate bots to generate legitimate traffic to flood the critical links between bots and the target area. If a defender reroutes the link traffic, attackers will select other critical links to attack the target area. Although several approaches [4], [5], [6] have been recently proposed to defend against LFA, those approaches are not flexible and cost-efficient, since they are built on traditional network architectures and do not fully consider and deal with the above unique characteristics of LFA.

An emerging networking paradigm, Software-Defined Networking (SDN) [7], which features granularity, flexibility and elasticity, provides promising ways to defend against network attacks [8]. SDN has several benefits, such as network-wide view, logically centralized control, software-based traffic analysis, and dynamic updating of forwarding rules, which can be leveraged for more effective attack defence.

In this paper, we propose LFADefender, a novel LFA defence system, that leverages key features provided by SDN to detect and mitigate LFA. A critical step for a LFA is to find target links. Using SDN technology, LFADefender can easily identify target links that are potentially used by LFA attackers, since the SDN controller has a globe view of network topology and can dynamically trace the flowpaths in the network. Hence, we propose a target link selection algorithm based on SDN to find high flow density links that can be exploited as target links by LFA. Also, since LFA floods and congests the target links to cut off the communications from/to a target area, the current approach [5] detects link congestion through sending a lot of real-time probing packets from the probes deployed inside or outside of a target area. Such an approach would bring significant detection delay due to the time delay of probing packets to target links. By using sFlow [9], a high-performance sampling technology, we design a link congestion monitor in LFADefender. The link congestion monitor can be dynamically deployed at each end of any target link to efficiently capture traffic data and then send them to the SDN controller where they are analyzed to produce a rich, real-time, network-wide view of traffic flows. Once

• *Juan Wang, Ru Wen, Jiangqi Li are with the Department of Computer Science, Wuhan University, Wuhan, China, 430072. Key Laboratory of Aerospace Information Security and Trust Computing, Ministry of Education, Wuhan, China, 430072.*
E-mail: jwang@whu.edu.cn, ruyulh@whu.edu.cn, lijqi@whu.edu.cn.

LFADefender detects target links congestion, a multiple optional links rerouting approach is used to mitigate the target links congestion. However, the rerouting approach can only temporarily mitigate LFA because the LFA bots can still send malicious traffic and keep consuming the bandwidth resource. Hence, we further propose a malicious traffic blocking approach to identify LFA bots and then radically prevent the malicious traffic from them.

To demonstrate the feasibility of LFADefender, we implement four modules including a target link selection module, a link congestion monitoring module, a traffic rerouting module, and a malicious traffic blocking module on top of Floodlight [10], an open source SDN controller. We also implement a link congestion monitor agent based on sFlow, which can be deployed at the endpoints of a target link to measure the congestion of the target link. Furthermore, we evaluate LFADefender using CloudLab [11], an open cloud platform, with real-world network topology. Our evaluation results show that LFADefender can agilely and effectively detect and migrate LFA.

The remainder of this paper is organized as follows. Section 2 introduces the background. In Section 3, we give an overview of LFADefender. Sections 4 and 5 illustrate the detailed approaches of detecting and mitigating LFA. Section 6 presents the implementation and evaluation of LFADefender. Section 7 discusses several important problems. Sections 8 and 9 provide related work, and conclusion and future work, respectively.

2 Background

2.1 Link-Flooding Attack

Link-flooding attack, which is considered as a new class of DDoS attacks, was discovered recently. There are two types of LFA, Coremelt attack [2] and Crossfire attack [3]. Coremelt attack is the origin of LFA, which defines target links and involves three steps in launching an attack: 1) select a core link in the network as the target link; 2) identify the pairs of subverted machines that can generate traffic to traverse the target link; and 3) send traffic between the machines identified in step 2 to overload the target link. Furthermore, the Crossfire attack is an upgrade of the Coremelt, which introduces some new concepts: decoy server, flow density, and rolling attack. One major difference between Coremelt and Crossfire is that the latter doesn't need to deploy collaborative bots in the target areas for sending and receiving traffic to overload target links. Instead, bots in Crossfire can send legitimate traffic to a decoy server, which is located near the target area. A Crossfire attack usually has four stages: 1) link map construction; 2) flow density computation and target-link selection; 3) bots coordination; and 4) rolling attack.

Compared with traditional DDoS attacks, there are several remarkable features in LFA. First, the attacked target is different. In traditional DDoS attacks, attackers send huge unexpected traffic to the target server directly. Instead, LFA attackers organize a large number of bots to send low-speed traffic to decoy servers and attempt to flood crucial links that connect a target area to the Internet. Thus, the victim doesn't even know it is being attacked. Second, traditional DDoS attacks only render one server unavailable while LFA prevents all servers in the target area from accessing Internet. Third, current DDoS defense systems attempt to eliminate unwanted traffic, while the bots in LFA send wanted or legitimate traffic. Hence, the traditional DDoS defense systems can't effectively distinguish attack traffic from normal traffic.

2.2 Software-Defined Networking

Software-defined networking (SDN) [7] is a dynamic, manageable, cost-effective, and adaptable network architecture, seeking to be suitable for the high-bandwidth and dynamic nature of today's applications. As a new network paradigm, the biggest difference between SDN and conventional network structures is that it can flexibly define the forwarding capabilities of network devices by operating flow tables. In addition, SDN architecture decouples the control plane and the data plane of a network, and the control plane can manage the network devices on the forwarding plane by using unified interface protocols, such as OpenFlow [12]. The SDN controller possesses a global view of a network that makes the SDN network more intelligent. Meanwhile, SDN provides flexible programmability in which users can customize upper business and applications to satisfy specific requirements. This new network-enabled architecture requires only a centralized software upgrade at the control node that enables network managers to configure, manage, secure, and optimize network resources very quickly via dynamic and automated SDN programs.

In this paper, we leverage SDN's remarkable features to detect and mitigate LFA. First, we get the flowpaths by flow analysis and monitor target links using link congestion monitoring. Since the SDN controller can get the whole network topology and all the flow tables in OpenFlow switches, we can depict all the flowpaths of the network to find target links. Furthermore, we can dynamically deploy link congestion monitor agents at the endpoints of the target links to monitor them. Secondly, during the course of being attacked by LFA, the state of the entire network is constantly changed. Thus, it's essential to obtain flexible control of the network. Furthermore, we identify the LFA bots through dynamic flow traceability provided with the SDN network.

3 System Overview

Using key features of SDN, such as network-wide view, flow traceability, and dynamic reconfiguration, we design the LFA defence system called LFADefender to detect and migrate LFA. Figure 1 depicts an overview of LFADefender. It consists of four main modules: target link selection, link congestion monitoring, traffic rerouting and malicious traffic blocking. We design the target link selection module and link congestion monitoring module to detect LFA. Then, the traffic rerouting and malicious traffic blocking modules are used to mitigate LFA. The target link selection module can communicate with the SDN controller, and obtain flow entries in OpenFlow switches from the controller. This module marks the links with high flow density as target links [3] and returns them. The link congestion monitoring module can retrieve the target links from the target link selection module, and then deploys link congestion monitor agents

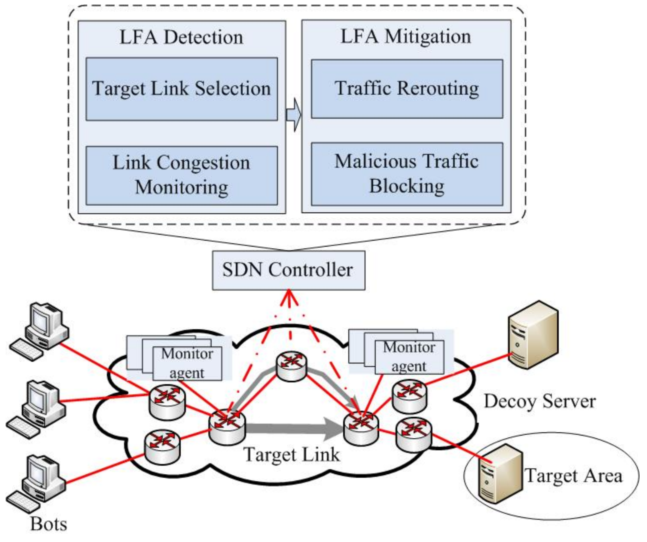


Fig. 1: Overview of LFADefender

on these links and reports link states to the SDN controller. The traffic rerouting module is used to mitigate link congestion via traffic rerouting. And the malicious traffic blocking module can identify attack traffic through traffic tracing, and tells switches to drop malicious traffic.

In LFA, an attacker first finds target links, which have high flow density [3] and then floods them. Therefore, LFADefender also needs to find those possible target links in advance, detect whether they are congested, and finally mitigate LFA by rerouting traffic and blocking malicious traffic. We give these four logical stages as follows:

1) Target Link Selection: In this stage, our goal is to find target links. We present a target link selection algorithm. We get flow tables from each switch, and then analyze these tables to compute flowpaths [13], which are the routing paths of each flow. We count the number of flows through each link and the links with high flow density are selected as target links.
2) Link Congestion Monitoring: We design a link congestion monitoring module. For all possible target links, link congestion monitoring installs monitor agents at the end of the target links to capture traffic information. Then, the SDN controller gains the network traffic status analyzed and collected by the link congestion monitor agents and determines whether the links are congested.
3) Traffic Rerouting: Once the links are congested, we need to reroute the traffic from the congested target links to other links. For each flow of the traffic, we will find a different routing path to reroute this flow to the new routing path until the target link is not clogged. Since a LFA may simultaneously congest several target links so as to isolate the target area, we design a multiple paths rerouting approach to temporarily mitigate LFA.
4) Malicious Traffic Blocking: In order to radically mitigate LFA, we design a malicious traffic blocking module, which can identify LFA bots and prevent the malicious traffic through multiple rerouting and monitoring traceroute packets. The attackers of LFA need to construct link-map and obtain stable target links, hence they must send a large number of traceroute packets to find the stable routes. Once the routes of the target links are changed via rerouting, the attackers will use the rolling attack to congest the new target links. However, since the attackers don't have an unlimited number of bots, some bots must occur on the new target links. According to the above features, the malicious traffic blocking module can identify LFA bots through malicious flow tracing based on multiple rerouting and the analysis of traceroute packets. Once LFA bots are identified, the malicious traffic blocking module will send block rules to SDN switches so as to drop the traffic from the LFA bots.

4 Detecting Link-Flooding Attacks

4.1 Target Link Selection

For a LFA adversary, it is essential to build an accurate link-map of network and select effective target links. As for a defender, it is hard to know which link is selected as an attack target. However, in an SDN network, the controller knows the whole network topology, and all the forwarding packets should follow the flow tables stored in OpenFlow switches. Hence, in this section, we propose a target link selection approach based on the analysis of flows in the SDN network, which finds target links by computing the flow density of a link. In Crossfire [3], the flow density is defined as the number of flows between bots and target-area servers that can be created through that link.

First, we introduce the concept of flowpath [13]. Flowpath represents the forwarding path of each flow in SDN. In our approach, we use Header Space Analysis (HSA) [14] to describe flowpath. By parsing the routing and configuration tables automatically, HSA provides a method to compute forwarding path of packets. In HSA, a packet header is defined as a point in space $\{0,1\}^L$ which is called Header Space (L is the length of the packet (in bit)). Based on Header Space, all network forwarding devices can be modeled as a Switch Transfer Function T, which can simulate the forwarding process. For example, when a header h arrives, it is forwarded to port p:

$$T(h,p) \rightarrow \{(h_1,p_1),(h_2,p_2,\cdots)\}$$

In our system, we model the flowpath between a pair of nodes as following $\{switch, rule\}$ sequence, where the rule is the packet processing rule in a switch.

$$(s_1,r_1) \rightarrow \cdots \rightarrow (s_{n-1},r_{n-1}) \rightarrow (s_n,r_n)$$

Here, we describe the detailed steps of obtaining the flowpaths in an SDN network. First, we obtain the header of the head node as the start point of a flowpath. As we known, SDN controller has the global network view, which makes it easy to gain network routing information. So, it is easy to know the next hop (e.g. switch $s1$) of this head node. Second, we read the header information of flow tables in $s1$, convert these information into a binary vector and calculate the intersection of head node's destination IP address and $s1'$ source IP address. If the result is not an empty set, $s1$ is added to the flowpath. Similarly, we need continue to obtain the next hop of s1, and compare the destination IP of $s1$ with the source IP of its next hop (e.g. switch $s2$). Next, what we need to do is to repeat the above steps until we find the destination node, and at the same time, we find a complete flowpath.

Based on the flowpaths, we need to compute the flow density of links in the later phase. Note that the link between current hop and next hop has a weight, which depends on the match field of the matched flow entry and the actual hosts in the network. Some terms are defined as follows:

W : the weight of a link;

P_{src} : the source IP space of a matched flow;

P_{dst} : the destination IP space of a matched flow;

Q : the set of all hosts' IPs in SDN;

and $W = (P_{src} \cap Q) * (P_{dst} \cap Q)$

For example, a flow with match fields is defined as follows:

$flow\_1$={$''src\_ip''$:"10.0.1.x/24",$''dst\_ip''$:"10.0.2.x/24"}, Q={"10.0.1.100","10.0.1.101","10.0.1.102","10.0.2.110","10.0.2.111"}

There are three hosts in the source IP space and two hosts in the destination space so that we can figure out the weight contributed by the $flow\_1$ is 6, which means that there are six flows that can be created through the link. Hence, the flow density of a link can be defined as follows: supposing that there are N flowpaths throughout the link L and each flowpath has a source IP space P_{isrc} and a destination IP space P_{idst} on L, the flow density D of L can be expressed as:

$$D = \sum_{i=1}^{N} W_i = \sum_{i=1}^{N} \{(P_{isrc} \cap Q) * (P_{idst} \cap Q)\} \tag{1}$$

In SDN, there are two kinds of flows: *proactive flow* and *reactive flow*. Proactive flows are installed on switches in advance and can be added/deleted by the network administrator according to the situation of the network, while reactive flows are inserted into switches in real time by the SDN controller and can only exist $5s$. In addition, the proactive flow has a higher priority than the reactive flow. Therefore, taking into account the priority, survival time and removal of the two flow tables above, when selecting target links by computing flowpaths and the flow density, we perform the target link selection algorithm in a certain time interval, such as $5s$.

For a LFA, when the flooding packets from bots reach an OpenFlow switch, there are two situations: 1) the switch could match these packets with proactive flows; and 2) the switch could not match these packets and sends them to the controller. Then, the controller pushes reactive flows to switches. In situation 1), our target link selection algorithm has already covered the proactive flows. That is to say, we have known possible target links that an adversary wants to get by the traceroute operation. In situation 2), our algorithm is continuously running so that we can get the updated target links by parsing reactive flows as soon as bots try sending packets to decoy servers. Note that the packets from bots to decoy servers are always forwarded according to either proactive or reactive flows. Therefore, the target links found by our approach are same as the attackers, and our algorithm can get the target links before LFA starts. When we find the target links, we can start the link congestion monitoring module to deploy the monitor agents and begin monitoring the target links. Our target link selection algorithm is presented in Algorithm 1.

Algorithm 1 Target link selection

```
Input: Boundary Open vSwitch set $sw[1 \ldots n]$
Output: The set of target links
1:  define link $< sw_i, sw_{i+1} >$
2:  define link_map $< link, Integer >$, $L$
3:  flows = get_flows($sw_1$)
4:  for flow in flows do
5:      if flow.match.dst == target_area then
6:          $sw_i$ = flow.action.output.ovs
7:          w = flow.match.weight
8:          L.put(new link($sw_1$, $sw_i$), w)
9:          break
10:     end if
11: end for
12: while $sw_i$ != NULL do
13:     flows = get_flows($sw_i$)
14:     for flow in flows do
15:         if flow.match.dst == target_area then
16:             $sw_{i+1}$ = flow.action.output.ovs
17:             w = flow.match.weight
18:             if L.has(new link($sw_i$, $sw_{i+1}$)) then
19:                 L.get(new link($sw_i$, $sw_{i+1}$)).value += w
20:             else
21:                 L.put(new link($sw_i$, $sw_{i+1}$), w)
22:                 break
23:             end if
24:         end if
25:     end for
26:     $sw_i$ = $sw_{i+1}$
27: end while
28: return L[top_k]
```

L stores the flow density of each link. The critical steps of Algorithm 1 are described as follows. First, we choose a boundary switch and traverse the flow entries in it. Second, we can get a path hop by hop according to the forward action of each flow. Then, the above steps are repeated to find all paths in the network. Finally, we compute the density of each link and retrieve top-k links with maximum flow density as the target links.

In the target link selection algorithm, the time complexity depends on the number of OpenFlow switches n, and the average number of flow entries m in the flow tables. Thus, the time complexity of this algorithm is $O(nm)$.

4.2 Link Congestion Monitoring

When the target links have been found, we need to detect whether they are congested by the LFA. In LFA, adversaries often dynamically change the set of target links and extend the duration of attack almost indefinitely. Continuous link flooding of the same set of target links would lead to the changes of bot-server routes since it would inevitably activate the routers failure detection mechanism [3]. Hence, changing the set of target links can assure attack persistence and enable the attack to remain a pure data-plane attack.

Because of this, we need to dynamically and rapidly deploy monitor agents on the possible target links to detect the LFA. Therefore, we design a congestion monitoring module to detect link congestion.

This link congestion monitoring module runs in the SDN controller, which is responsible for the deployment of agents and the analysis of link congestion. When receiving the target link information from the target link selection module, the monitoring module installs the link congestion monitor agents at the switches of target links, and when a target link is changed to a non-target link, this module removes the agents immediately. The SDN controller can read and analyze the network data of the link congestion monitor in real time.

Next, we discuss the problems of measurement deviation and deployment of link congestion monitoring.

- *Measurement Deviation*: Packet pair is commonly used in the network measurement. However, cross traffic can randomly enter or leave a probing path, which is called one-hop persistent cross traffic, and results in the deviation [15]. Therefore, in our solution, we use the link congestion monitor agent to sample and measure traffic data, and directly connect the monitor agents with the target links to eliminate the deviation.
- *Flexibility and Scalability*: In a LFA, there may exist many target links in a target link set, and an adversary periodically selects several flooding target links from the set to confine the attack. According to the features of LFA, our design should be flexible and scalable. On the one hand, we should deploy agents on both ends of a target link, and remove them when they aren't required for saving resources. On the other hand, we should have the ability to provide enough agents for monitoring links. Our link congestion monitor agent can meet the two requirements to monitor target link, which is flexible and scalable, and easy to be deployed in OpenFlow switches.

The link congestion monitor agent can gather network traffic information. We use the following link performance metrics to determine whether a link is clogged.

- Packet loss rate, which is increased when a link is clogged.
- RTT (Round-Trip Time), which is also increased because of the queue overflow in a router under link flooding attack.
- Available bandwidth, which is clearly decreased because of a lot of traffic traverses the target links.

In a router, the packets are temporarily stored in cache when the arriving rate is greater than the outgoing rate. The cache is implemented as a queue with FIFO. Once this situation lasts for a while, the queue will be full, and the subsequent packets will be dropped, which causes a substantial increase in the packet loss rate and RTT. We also test the result by experiments.

In our experiments, we found that if the bandwidth of the attack traffic has reached or even exceeded the maximum bandwidth, the available bandwidth we detect will be fixed at a certain range. Regardless of the increase in the attack traffic, the available bandwidth will float around a certain value. However, the packet loss rate and RTT will have tremendous changes. Hence, we can confirm that the link is congested, if these changes have occurred. We use ABW (Available BandWidth), Loss and RTT to represent the detected available bandwidth, packet loss rate and round-trip time of target link and use MABW, MLoss and MRTT to represent their thresholds respectively. In normal circumstances, we start the link congestion monitoring module and obtain the traffic data. If Loss, RTT is higher than MLoss and MRTT and ABW is lower than MABW, we confirm that the target link is congested. We use an adaptive threshold algorithm [16] to compute MABW, MLoss and MRTT. Through the link congestion monitor module, we can monitor target links to detect LFA in real time.

5 Mitigating Link-Flooding Attacks

Our LFA mitigation mechanism consists of traffic rerouting module and malicious traffic blocking module. Once link congestion is detected, the link congestion monitoring module will inform the SDN controller to start the traffic rerouting. Meanwhile, the malicious traffic blocking module will identify LFA bots through flow tracing and traceroute packets analysis. When the bots are identified, the SDN controller will send block rules to drop the malicious flows.

5.1 Traffic Rerouting

When the target links are detected to be congested, we need to find optional links to reroute the partial traffic of the congested links. To mitigate LFA, we propose a multiple optional paths rerouting method. By this method, we can obtain multiple optional links that have sufficient remaining capacity so as to temporally mitigate the target link congestion.

We formulate this traffic rerouting problem for minimizing the maximum link utilization in a network. Before giving the formulation of our approach, we introduce the following definitions:

Let us assume that we have a network graph $G = (V; E)$. V represents the set of routers and E represents the set of links between two routers.

Definition 1. Let c represent the current router or the host, and d represent the destination router or host. We use

$$N_{c,d} = v \quad where \quad c, d, v \in V \tag{2}$$

to represent that the next hop of node c for destination d is node v.

Definition 2. Given a network graph $G = (V; E)$, we use $L_{s,d}$ to represent all potential paths from source s to destination d and $P_{s,d}$ represent a specific forwarding path from s to d in $L_{s,d}$.

$$P_{s,d} = (v_1, v_2, \ldots, v_n) \quad where \quad N_{v_i,d} = v_{i+1} \tag{3}$$

Definition 3. The bandwidth utilization between a pair of neighbor nodes is defined as the ratio between the used bandwidth and the capacity of this link. For a multi-node link, the utilization is determined by the biggest link utilization of neighbor nodes. We use $U(P_{s,d})$ to represent the bandwidth utilization of a specific path from node s to node d, which can be shown as:

$$U(P_{s,d}) = Max(U(P_{s,v_i}), U(P_{v_i,d})), i \in [1, n] \quad (4)$$

Definition 4. Let $BP_{s,d}$ represent the best path from node s to node d in network G.

$$BP_{s,d} = \{P_{s,d} | U(P_{s,d}) = Min(U(P_{s,d})), \\ where\ P_{s,d} \in L_{s,d}\} \quad (5)$$

Compared with the general situation, two other factors should be taken into account in LFA. First, the adversary selects several target links to flood. Thus, there may exist more than one link needed to be rerouted at the same time. Second, rerouting time should be as short as possible to avoid attackers changing their targets in advance. In addition, rerouting might have influence on flow density, which will be updated later. Thus, this should be taken into consideration that links with a higher flow density need to be removed to prevent following congestion. Therefore, we need to choose several optional paths and determine how much traffic should be routed to the optional links so as to avoid new congestion. We use α to describe the percentage of the traffic that will be rerouted, and β to make a restricted parameters of flow density. Both of them can be adjusted according to the links condition.

As mentioned in Section 4.1, the flow density of L is noted as D. To account each endpoint of this link in detail, we take a further representation $D(v_i, v_j)$ to denote the flow density between nodes v_i and v_j.

We use $R(v_i, v_j)$ to denote the bandwidth that has been consumed of the link (v_i, v_j) and $C(v_i, v_j)$ to denote the capacity of the link (v_i, v_j). Our rerouting condition of selecting optional links is defined by the following Eq:

$$\begin{cases} U(P_i) \leq U(p) for P_i \in L_{s,d}, \\ \qquad \forall p \in L_{s,d} \backslash \{\sum_{i=1}^{m-1} P_i\}, i \in [1, m] & (6) \\ C(v_i, v_j) - R(v_i, v_j) \geq \alpha U(T) & (7) \\ D(v_i, v_j) \leq \beta D(T) & (8) \end{cases}$$

where T represents the congested link to be rerouted and m is the number of optional links.

- The formulation (6) represents that the m links with minimum utilization are selected as optional links.
- The formulation (7) ensures that link(v_i, v_j) is not congested after rerouting. If the link cannot satisfy the condition, it will be removed.
- The formulation (8) means to avoid the link being selected for its high flow density in the following attack.

According to the formulation above, we propose an algorithm based on *Dijkstra* algorithm [17]. This algorithm takes the network graph $G = (V; E)$, the target link T, the capacity of each link $e \in E$, and the ratio α as inputs. It outputs the new routing path for the flows from s to d. The pseudocode of this algorithm is given in Algorithm 2. In the initialization phase, the links that don't have enough capacity for routing and those with high flow density will be deleted. We start from the source nodes and try to optimize paths step by step. For the current node, we consider all of its neighbor nodes and calculate the utilization between them. Then, by comparing the newly calculated utilization with the current utilization, we assign the bigger value to those extended path. That means we take the utilization as the measurement of this modified Dijkstra. When the search process extends to the target node, the optimal route will be found. In this algorithm, once the link which can not meet condition above, it will be removed from the set E. Hence, the link selected by our solution is an optimal local link.

Algorithm 2 Traffic rerouting algorithm

```
Input:
The network graph $G = (V; E)$ ;
The congested link set $T_{s,d}$;
The capacity of each link $e \in E$;
The bandwidth ratio $\alpha$.
Output:
The new routing path
 1: find $L'_{s,d} = \{BP_{s,d}, P^1_{s,d}, ..., P^{m-1}_{s,d}\}$, where $U(BP_{s,d}) \leq U(P^1_{s,d}) \leq ... \leq U(P^{m-1}_{s,d}) \leq U(p), \forall p \in L_{s,d} \backslash L'_{s,d}$
    Select:
 2: for $(v_i, v_j) \in P_{s,d}\ in\ L'_{s,d}$ do
 3:     if
            $(C(v_i, v_j) - R(v_i, v_j)) < \alpha \max(U(t))$
            or   $D(v_i, v_j) > \beta D(t), \forall t \in T_{s,d}$
        then
 4:         remove($P_{s,d}$)
 5:     end if
 6: end for
 7: if $L'_{s,d} == \emptyset$ then
 8:     update the value of $\alpha$
 9:     go to Select
10: end if
11: while $\exists t_{s,d} \in T_{s,d}$ do
12:     reroute($t_{s,d}$) to $BP_{s,d}, BP_{s,d} \in L'_{s,d}$
13:     update($R(L_{s,d}), C(L_{s,d}), T_{s,d}$)
14: end while
```

The time complexity of rerouting algorithm depends on the number of switch nodes n and the number of links m deployed among the congested area which contains all possible paths from the congestion sources to their destinations. In the step of initial filter, links without enough bandwidth or proper flow density are removed. Thus, the time complexity of the algorithm is $O(m)$. After that, the time complexity of Dijkstra algorithm is $O(n^2)$. Therefore, the time complexity of our rerouting algorithm is $O(m+n^2)$.

5.2 Malicious Traffic Blocking

Traffic rerouting can just temporarily mitigate LFA attacks because the LFA bots can still send malicious traffic and consume the bandwidth resource. Therefore, we further design a malicious traffic blocking module to identify the malicious bots so as to radically mitigate LFA attacks.

In order to construct link-map and obtain stable target links in LFA, adversaries manipulate a large number of bots to perform traceroute. After that, adversaries coordinate bots to flood the target links and monitor the target links until any changes. Once the target links are rerouted, adversaries must reconstruct a new link map, select new target links and reassign attack flows. Some bots must occur on the new target link because the bots owned by the adversaries are limited. Hence, we leverage malicious flow tracing based on multiple rerouting and monitoring traceroute packets to identify LFA bots. Furthermore, once the LFA bots have been identified, SDN controller can install block rules to drop the malicious packets from LFA bots.

5.2.1 Malicious Flow Identification Based on Rerouting

Once a target link is selected, we add source IPs of the flows traversing through the target link into the set A_1. When the target link is congested, traffic rerouting will be started so that the overall state of the network will be changed. Attackers will reconstruct link-map and reselect target links according to the new network routing situation immediately. Hence, we record the source IPs of the flows traversing through the new target link and get a new set A_2. After n times, we can get sets $A_1, A_2 ... A_n$. The number n can be determined by the administrator according to actual circumstances. We take the intersection of them as set B_1:

$$B_1 = A_1 \cap A_2 \cap ... \cap A_n \tag{9}$$

The IPs in set B_1 will be identified as suspicious bots. With the increase of n, the accuracy of the selection will be improved gradually.

After rerouting, target links will be updated, and adversaries need sending attack flows to flood the new target links. Through several rounds of detection, B_1 can be almost determined as bot IP set. Once a new IP address is added to the set B_1, a block flow message can be sent from the SDN controller to switches to drop packets from the IP address. Hence, the malicious flow could be blocked.

5.2.2 Malicious Flow Identification Based on Traceroute Packets Analysis

In LFA, adversaries coordinate individual bots to flood target links, and must keep each per-flow rate low enough (e.g. 4Kbps) to avoid being detected by the network protection mechanism or IDS. At the same time, for enhancing the undetectability of the target area, the aggregated attack traffic to flood all the target links is assigned to multiple bots evenly (e.g. 1Mbps). That is, each bot sends the attack traffic to many decoy servers (e.g., 250, 1Mbps/4Kpbs) at least. And in order to gain the stable and multiple routes, a bot runs at least 6 traceroute operations to the same decoy server. That is to say, each bot may need sending 250*6=1500 traceroute packets. In the real situation, the attackers may instruct individual bots to run more traceroute. Therefore, compared with the normal circumstances, the number of traceroute packets must increase drastically, which can be exploited to detect bots. In addition, traceroute packet is an ICMP packet containing incremental TTL value and invalid destination port. With these two unique features, we can distinguish traceroute packet from others.

As we mentioned before, adversaries send a lot of traceroute packets to construct link-map and select target links, which inevitably leads to an increase of the number of traceroute packets in the network. Our proposal aims towards detecting the point where the traceroute packets increase suddenly.

In the following, we provide more detailed description about our approach. First, we monitor target links and record the number of traceroute packets in a sliding window. The size of the sliding window is T, and it consists of t intervals and moves one interval each time. As mentioned in the Target Link Selection module, top-k links with maximum flow density are filtered as target links. And we denote n_i to represent the sum of traceroute packets through all the target links during an interval of the sliding window. That is, all the data of each sliding window is just as the following equation:

$$N = \{n_1,\ n_2,\ \cdots, n_i, \cdots,\ n_t\}\ where\ i \in [1, t]$$

In order to detect the variation of traceroute packets, it is common to make use of statistical methods, such as calculating average value and variance. But considering the uncertainty of traceroute packets in LFA, it is difficult to determine an appropriate threshold. Therefore, we leverage an outlier detection algorithm called LOF (Local Outlier Factor) [18] to identify the time point that the number of traceroute packets increases suddenly. LOF is based on a concept of a local density, which can be estimated by K-nearest neighbor algorithm. By comparing the local density of an object to its neighbors, we divide the objects who have similar densities into one region and identify who have obviously lower density than their neighbors as the suspicious outliers. Before giving the steps of our algorithm, we introduce the following definition first [18]:

Definition 5. (K-distance)

For any positive integer K and a dataset D, the K-$distance$ of object p, denoted as K-$distance(p)$, is defined as the distance $d(p, o)$ between p and an object $o \in D$:

(i) for at least K objects $o' \in D \backslash \{p\}$ it holds that $d(p, o') \leq d(p, o)$, and
(ii) for at most K-1 objects $o' \in D \backslash \{p\}$ it holds that $d(p, o') < d(p, o)$.

Next, we introduce how to detect outliers by computing LOF:

A. Calculate K-distance

For each data n_i, we calculate K-$distance(n_i)$ that is the distance from n_i to its K-th nearest neighbors.

B. Calculate reachability distance

Next, we calculate the reachability distance of data n_i with respect to data n_p, which is given by:

$$reach\text{-}dist_K(n_i, n_p) = Max\{K\text{-}distance(n_i), d(n_i, n_p)\} \tag{10}$$

where $d(n_i, n_p)$ is the Euclidean distance from n_i to n_p.

C. Calculate local reachability density (lrd)

$$lrd_K(n_i) = \frac{1}{\sum\limits_{n_p \in KNN(n_i)} reach\text{-}dist_K(n_i, n_p) \Big/ |KNN(n_i)|} \tag{11}$$

where KNN represents K-nearest neighbors.

Based on the K-nearest neighbors, we calculate lrd of n_i, i.e. the inverse of the average reachability distance.

D. Calculate LOF

Finally, we calculate the $LOF(n_i)$ by the average local reachability density of n_i and local reachability density of data n_p.

$$LOF_{\mathrm{K}}(n_i) = \frac{\sum\limits_{n_p \in KNN(n_i)} lrd(n_p)/lrd(n_i)}{|KNN(n_i)|} \tag{12}$$

We leverage a sliding window to record the number of traceroute packets, which is the data set used by the above algorithm. By detecting increase of the data, we identify suspicious traceroute packets from bots. However, after each interval, the data set will be updated (such added/deleted) and the LOF values of all the data must be computed again. Obviously, under this circumstance, the algorithm has very high time complexity and low efficiency. Hence, we further adopt the incremental dynamic LOF algorithm [19]. It only updates the LOF values of affected data and does not recalculate all the LOF values, which improves the computational efficiency in the dynamic environment.

(I) When the added data p_c affects the K-distance of the original data p_j, that is p_c in the K-nearest neighbors of p_j. Then, the new K-distances of p_j are recalculated as follows:

$$K\text{-distance}^{(new)}(p_j) = \begin{cases} d(p_{\mathrm{j}}, p_c),\ p_c \text{ is the } K\text{-th nearest neighbor of } p_{\mathrm{j}} \\ (K\text{-}1)\ \text{distance}^{(old)}(p_{\mathrm{j}}), \text{otherwise} \end{cases} \tag{13}$$

Because of the change of $K\text{-}distance(p_j)$, $reach\text{-}dist_K(p_j, p_i)$ may be affected where p_i is K-neighbor of p_j. At the same time, lrd values of p_i and p_j have to be updated. And once lrd and K-nearest neighbors have been updated, LOF value of the data must be updated.

(II) When data p_d is deleted from set, we need to recalculate LOF value of data affected by p_d, just like above-updated process after adding data. If K-nearest neighbors of a data include p_d, this data needs to be added to the pending updated dataset.

According to the above algorithm, if the LOF value of p is closer to 1, it means the density of p is almost same as its neighbors and they maybe belong to the same region. Otherwise, p is mostly likely to be an abnormal data. Therefore, by calculating the LOF value, we can identify the time point that the number of traceroute packets increases suddenly. Furthermore, we record the traceroute IPs that appear in the network after that time point, which are probably the bots IPs. After rerouting, adversaries have to coordinate bots to reconstruct link-map by running traceroute, which exposes more and more information about bots. Therefore, through multiple detections, we can identify most of bots.

In this algorithm, we assume the number of flows in the sliding window is a constant N. Hence, the time complexity of $KNN()$ is $O(N)$. As the sliding window moves, the flow is updated accordingly which leads to KNN, lrd and LOF changing. We assume the flow set whose k-distance neighborhood needs be updated is S_{KNN}, whose lrd needs be updated is S_{lrd} and whose LOF needs be updated is S_{LOF}. The time complexity of updated LOF is $O(|S_{KNN}| \times |S_{lrd}| \times NK)$. Obviously, $|S_{KNN}|$, $|S_{lrd}|$ and K are far less than N. Thus, the time complexity of our proposed algorithm is $O(N)$.

In LFA, bots have two functions, which are constructing link-map by running traceroute and flooding target links. Therefore, by leveraging these two aspects, we implement Malicious Traffic Blocking module based on multiple rerouting and traceroute packets analysis. Combining the two methods together, we can identify most of LFA bots. Finally the malicious blocking module can notify the SDN controller to send block rules to drop the malicious flow from LFA bots.

6 Implementation And Evaluation

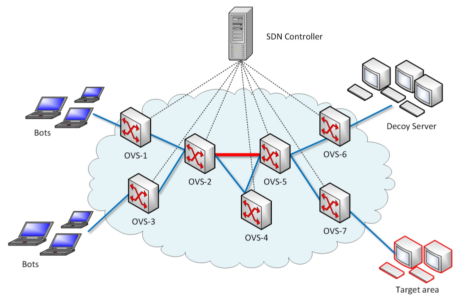
We carried out experiments in a testbed to evaluate the functionality and overhead of our solution. Given the legal and ethical issues surrounding DDoS attacks, rather than renting a botnet and attacking the network, we simulate attacks and deploy our system on CloudLab [11].

6.1 Implementation

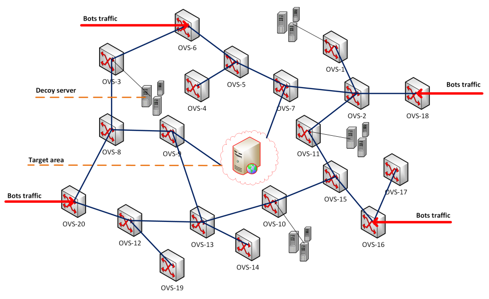
The target link selection module is implemented with Python. We deploy this module on Floodlight, which is a popular open-source SDN controller. Note that the target link selection module runs out of the controller and it gets data through the REST APIs of Floodlight so that it won't introduce additional performance overhead to the controller. The primary REST APIs used by target link selection algorithm mainly include "/core/controller/switches/json", "/topology/links", "/device/" and "/core/switch", to receive all switches DPIDs, the inter-switch links, devices information, and all proactive flows and reactive flows installed in switches.

Based on sFlow, we develop our link congestion monitoring module in Floodlight and link congestion monitor agents. sFlow [9] is a sampling-based network traffic monitoring technology, which is mainly used for statistical analysis of network traffic. It provides complete packet information at Layer 2 to 4 of the OSI model, even including the whole network-wide traffic. By analysis and monitoring, users can obtain real-time network performance and traffic status. Furthermore, sFlow supports OpenFlow protocol and can be deployed in OpenFlow switch easily, which makes it conveniently used in SDN networks. The sFlow monitoring system consists of a sFlow agent and a centralized collector. The agent is software process embedded in the network forwarding devices (such as switches, routers). It captures packets from monitored network device interface according to a certain sampling rate, then collects traffic and data information. After that, it packages data into sFlow datagrams that are immediately sent to the sFlow collector.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2822275, IEEE Transactions on Dependable and Secure Computing



(a) Topology with 7 OVSes



(b) Topology with 20 OVSes

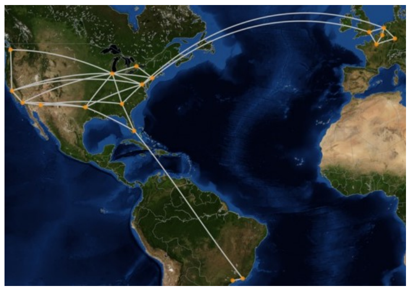

(c) Topology of Highwinds network

Fig. 2: Topology of testbed

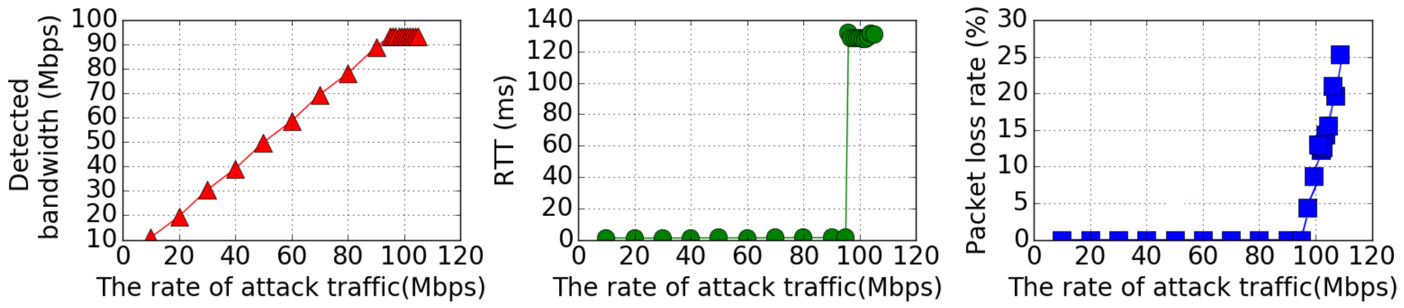


Fig. 3: Detected bandwidth, RTT and packet loss

Collector gathers and analyses the information to create a network traffic visibility picture. In addition, multiple agents are independent and when you want to add new sFlow agent, just need to configure sample parameters and connect to the active sFlow collector.

Both sFlow collector and Floodlight controller are centralized and provide REST API with which they can communicate in JSON format. Therefore, Floodlight can easily obtain the link information collected by sFlow-rt in real time through REST API. First, the controller retrieves top-k links with max flow density as the target links from the target link selection module. Then, according to the returned switch DPIDs and port IDs, sFlow agents are deployed in monitored Open vSwitches (OVSes) and configured agent IPs to connect sFlow collector. Next, Floodlight reads the link information in sFlow-rt such as the available bandwidth, RTT and packet loss rate to detect network status in real time. Last, by analyzing and setting the threshold, controller can detect target link congestion. Once the congestion occurs, Floodlight will initiate the traffic rerouting module immediately to mitigate link congestion and decrease the impact of the crossfire attack. Furthermore, we install sFlow collector and Floodlight controller on the same host to accelerate communication and reduce unnecessary latency.

In addition, we modify Floodlight and add a new module to process the request messages from the link congestion monitor. Once the link is going to be congested, the controller will reroute some traffic from the target link to a new routing path. We define two data structures: 1) route structure, which stores the shortest path from one switch to another, and 2) flows structure, which stores flows needed to be rerouted. After rerouting several times, we measure whether the target link is still in congestion to decide to continue rerouting or not. If the rerouting needs to be continued, we update the route structure by removing links that have no enough available bandwidth. This operation is repeated until the target link is not congested.

Similarly, we implement the malicious traffic blocking module in the floodlight. By making full use of its favorable features of centralized control, we can conveniently obtain the information of packets such as quantity and IP. Further, we execute LOF algorithm and identify suspicious bots. Once the LFA bots have been identified, the malicious traffic blocking module will notify the SDN controller to install drop flow to block the malicious traffic.

6.2 Testbed

Figure 2 shows three topologies of our testbed to simulate the LFA, detect target link, and mitigate link congestion respectively. Bots are used to generate legitimate traffic. The core of the network is composed of OVSes and SDN controller on an x86 server. Figure 2(a) has 7 OVSes, and we assume that two links between OVS-2 and OVS-4 and between OVS-4 and OVS-5 are backup links. The link between OVS-2 and OVS-5 is the target link, whose bandwidth is 100Mbps. The bandwidth of all links in our topology is 100Mbps. We set up the topology on CloudLab, where we can directly control physical machines and run Ubuntu systems on all hosts. Figure 2(b) has 20 OVSes, and bots' traffic floods decoy severs from OVS-6, OVS-16, OVS-18 and OVS-20. And we set up 4 decoy servers surrounding the target area. In addition, we construct our topology from a global Highwinds network provided by Topology Zoo database [19] depicted in Figure 2(c), which has up to 20 backbone routers and 50 backbone links. We import this topology into Mininet to test our approach.

6.3 Overhead of Target Link Selection

Supposing that there are n OVSes in an SDN network and each OVS has m flows, the time complexity of the target link detection algorithm is $O(nm)$. We emulated two SDN topologies: one of them has 7 OVSes and another has 20, as shown in Figure 2. Figure 4 shows the trend of time delay when flow entries in each OVS increased. However, our target link detection program can be run in any host as long as the host can access the SDN controller's RESTful API. Thus, our system doesn't introduce additional overhead to the controller.

6.4 Detection Rate of Link Congestion

Figure 3 shows the results of detection measured on the congested links. It demonstrates the patterns of packet loss rate, RTT, and bandwidth in the cases of normal and congested states. As it can be seen from Figure 3, when the rate of attack traffic is less than 95Mbps, RTT floats around 0.5ms and packet loss is always zero. When the rate of attack traffic becomes higher than 95Mbps, or even a little, RTT has been greater than 100ms and the loss rate is changed to about 4%. We regard this as the beginning of congestion. When the rate of attack traffic continues to increase and becomes higher than 105Mbps, the detected bandwidth floats around 93Mbps, packet loss rate becomes more than 20% and RTT is also greater than 120ms. Therefore, we can think that the thresholds of bandwidth, packet loss, and RTT are 93Mbps, 4%, and 100ms, respectively, where the target link is transformed from the normal state to the congested state. When the detected value is greater than the threshold value, the link is considered as congestion.

To evaluate the LFADefender detection rate, we generate attack traffic higher than the available bandwidth. During the LFA, the detection rates are always 100%. Because when the attack traffic rate is much higher than the available bandwidth, the path is congested and none response packets can be received from the destination all the time. When the attack traffic rate is close to or a little higher than the available bandwidth, LFADefender can still receive some response packets, compute the measurement results, and determine whether the link is congested.

6.5 Effectiveness of Traffic Rerouting

We simulated LFA in a topology depicted in Figure 2(a) to evaluate how effective LFADefender is used to mitigate the link congestion. We used six bots to generate attack traffic. As we have described in Section 6.2, the target link of our test topology is a link between OVS-2 and OVS-5, whose maximum bandwidth is 100Mbps. In order to clog the target link and test a better rerouting performance, the bandwidth generated by attack traffic is set as 120Mbps. Therefore, the sending rate of each bot is set as 20Mbps. For the convenience of description, we use the number 0,1,...,6 to represent links between OVS-1 and OVS-2, OVS-2 and OVS-3, OVS-2 and OVS-4, OVS-2 and OVS-5, OVS-4 and OVS-5, OVS-5 and OVS-6, and OVS-5 and OVS-7, respectively, depicted as the X-axis in Figure 5.

In order to verify the validity of traffic rerouting, we measured the link utilization every second using sFlow. From Figure 5, we can find that the utilization of *link 3* is close to 120% before rerouting. Because the attack traffic bandwidth is 120Mbps and the target link bandwidth is 100Mbps. After rerouting, the utilization of *link 3* decreases to 60%. The utilizations of *link 2* and *link 4* increase to 60% since we reroute attack traffic from the target link to the new routing paths, which are made up of *link 2* and *link 4*. After rerouting, there are not links whose utilizations exceed 100%. Therefore, we can conclude that the congestion of target link has be mitigated. And Figure 6 shows that the rerouting time is less than 2s.

We also tested the effectiveness of traffic rerouting in a large-scale flow-level simulation. We constructed our topology from a global Highwinds network provided by Topology Zoo database [20] depicted as Figure 2(c). Highwinds network has up to 20 backbone routers and 50 backbone links. We simulated hundreds of attack flows using iPerf, and the bandwidth of each flow is not greater than 1Mbps. Figure 7 shows the variation of link utilization and the congestion is resolved in 4 seconds.

6.6 Accuracy of Malicious Traffic Identification

In order to detect the accuracy of malicious traffic identification, we set 100 virtual machines as bots, and 20 virtual machines as the normal hosts. The 100 bots send traceroute packets to servers before flooding network links, where each bot executes at least 6 times traceroute. During the attack phase, the bandwidth of attack traffic from bots is 1Mbps. At the same time, the remaining 20 normal hosts randomly access the servers in the target area, and send some traceroute packets to the servers in this process. Figure 8 shows the accuracy of the bots identification. The red line in the figure is the percentage of identified bots in the total number of bots, and the blue line represents the percentage of normal hosts that are misjudged as bots. The results show that the accuracy rate of bots identification is more than 60% for the first time, which indicates our proposal is relatively effective. Although it includes some misjudgment of normal hosts, the accuracy rate of bots identification and misdiagnosis rate can reach a satisfactory result with the continuation of the experiment. Finally, we can successfully identify more than 90% bots. The information of these bots can be added into a blacklist so as to prevent the malicious traffic from LFA bots.

7 Discussion

To maintain the effectiveness of attacks in LFA, adversaries can launch adaptive attacks. The adversaries can select different target links and repeatedly launch attacks every 3 minutes. That is called *rolling attack*. Thus, to efficiently prevent the link flooding attack, the detection and mitigation processes should be done as fast as possible. Unfortunately, current Internet's dynamic response to the link flooding attack is too slow for the adaptive adversary [21], [22], [23], [24]. But our link monitor agents are directly connected at the end of target links and can monitor the status of ports in real time. The SDN controller can obtain the entire network

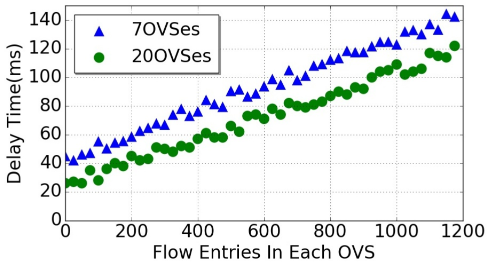


Fig. 4: Time delay of target links detection

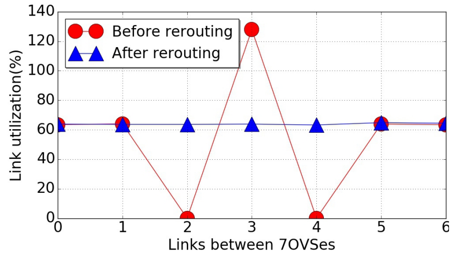


Fig. 5: Link utilization before and after rerouting

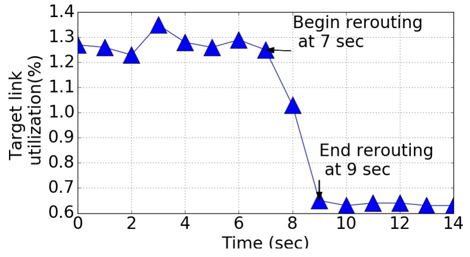


Fig. 6: The overhead of traffic rerouting

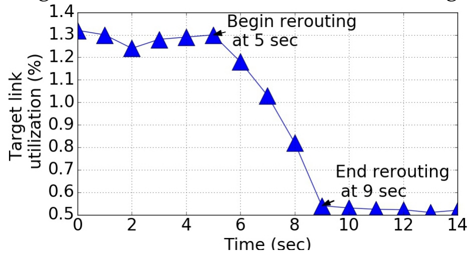


Fig. 7: Effectiveness test in Highwinds network

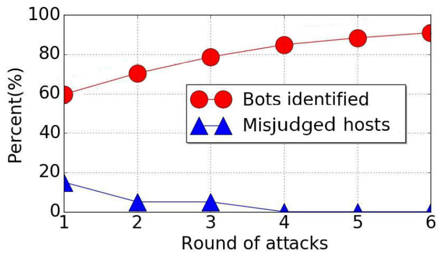


Fig. 8: Accuracy test of malicious traffic identification

topology and help our algorithm to find new routes faster than traditional algorithms. Figure 6 and Figure 7 also show that attacks can be mitigated on a small time scale (only 10 seconds) even on backbone networks.

The target links selected by an adversary could be located in an ISP or between two ISPs. To improve the robustness and stability of the network, redundant links are commonly created between the backbone network routers. When a congestion occurs between two ISPs, we can use the backup links as new routes to recover the network communication. If we cannot find new routes in an ISP, we can forward this flow to adjacent ISPs, which can find new routes for this flow. Thus, this problem can be converted to the first problem. Finding new routes cross different ISPs needs the support of intercommunication between controllers. Distributed controllers, e.g. HyperFlow [25], Onix [26], and ONOS [27], are relatively simple solutions, which only support communication among the same kind of controllers. In contrast, the cross-domain communication protocols could provide a better solution to address the problem. SDNi [28] and West-East Bridge [29] are protocols that already exist and can be used in practice. Fortunately, some other open source controllers, such as OpenDaylight [30], which can exchange data of network topology based on BGP. Hence, we can find new routes cross different domains based on the adjacent domain information, which can be obtained by using the BGP protocol. We also can deploy sFlow on edge SDN routers so as to monitor the target links between ISPs. Once the controller of an ISP has found link congestion, it can reroute its traffic to other links. For bot identification methods, different ISPs can also use it. In this case, one ISP may identify bots based on source IP and another ISP may identify bots based on destination IP.

LFADefender leverages some key SDN features to detect and defend LFA. For conventional networks, our idea may be still useful. But it would be very expensive and the defence response could be much slower than SDNs. For example, in conventional networks, we may need to track the network-wide flows to find target links, and collect all of the route tables and send a server to compute flow paths.

8 Related Work

8.1 SDN-based Security

FRESCO [31] is an OpenFlow security application development framework designed to facilitate the rapid design, and modular composition of OpenFlow-enabled detection and mitigation modules. Avant-Guard [32] implements the connection migration and actuating trigger, which makes SDN security applications more scalable and responsive to dynamic network threats. FlowGuard [33] is a comprehensive framework, which accurately detects and effectively resolves firewall policy violations in dynamic OpenFlow networks. Based on SDN and NFV, Seyed K. Fayaz et al. implement a flexible and elastic DDoS defense system, Bohatei [8]. But it can be only used to mitigate traditional DDoS attacks and cannot be used to defense link-flooding attacks. Therefore, we take advantage of SDN to design LFADefender for the detection and mitigation LFA.

8.2 Link-Flooding Attack Detection and Mitigation

Linkscope [5] employs both end-to-end and hop-by-hop network measurement techniques to capture abnormal link performance degradation for detecting LFA. Compared with Linkscope, our LFADefender leverages the features of SDN to compute flow density and detect possible target links. It does not need to send probe packets from the outside of networks. Furthermore, LFADefender monitors target links dynamically, hence it also does not require complex hop-by-hop network measurement to find the congested links.

Codef [4] can distinguish low-rate attack flows from legitimate flows by modified routers, and protect legitimate traffic during link-flooding attacks. Unfortunately, its effectiveness may be limited because it needs to modify traditional routers and cannot be widely deployed to the Internet immediately. Spiffy [34] leverages the SDN traffic engineering to increase the bandwidth of network links logically, then forces an adversary to choose between two unpleasant alternatives, namely either allowing bots detection or accepting an increase in attack cost. When the bandwidth of network link is expended, the senders that maintain the low rate will be identified as bots. However, LFA are usually made by the low rate web traffic so as to bypass the current security analysis. Hence, when the bandwidth is increased with multiple of M, the traffic rate of normal users is difficult to increase by multiple of M in practice.

VN migration approach [6] can significantly increase the uncertainty about critical links of multiple VNs. However, it brings a larger cost for migrating the target area network and servers. Aydeger et al. propose an SDN-based MTD mechanism to mitigate LFA [35]. They obfuscate the links during the potential link-map creation of the attackers to make it harder to launch the attacks and mutate routes to defend LFA.

Woodpecker [36] present a scheme to defense against LFA based on the incrementally deployed SDN scenario. They add the least SDN switches to obtain the whole network information so as to reduce the cost of deploying SDN switches. Hence, they model the problem of updating M nodes to maximize the network connectivity and design an efficient heuristic algorithm to solve it.

Liaskos et al. [37] propose a framework to detect suspicious bots and target areas in LFA. The detection process is modelled as a problem of relational algebra that represents the association of bots and target areas. Through analysis, they detect and dissolve the malicious association. This work is based on TE, which continuously reroutes traffic to enable bots to reveal their presence. Compared with their framework for theoretically modelling LFA, we propose a more practical and complete system to detect and mitigate link-flooding attacks using the SDN technology.

9 Conclusion And Future Work

In this paper, we have proposed a link-flooding attack defense system, LFADefender that leverages SDN features to effectively detect and mitigate LFA. In LFADefender, we propose a target link selection approach and a congestion monitoring mechanism to effectively detect LFA, in further, propose a multiple traffic rerouting method and a malicious traffic blocking approach to radically mitigate LFA. In addition, LFADefender has been implemented based on Floodlight, and evaluated on CloudLab with real-world network typologies. Our evaluation results have shown that LFADefender could rapidly detect and eliminate LFA with very low performance overhead.

As our future work, we plan to optimize our rerouting method to achieve faster defence for LFA since attackers may change the target links more quickly before the rerouting defence mechanism takes effect. In addition, other LFA bot identification approaches can be considered to improve the accuracy of bot identification in defending LFA.

References

[1] Justine Boucher. Survey report: Majority of service providers experienced ddos attacks, 85 percent experienced customer churn as a result. http://www.businesswire.com/news/home/20150303005304/en/Survey-Report-Majority-Service-Providers-Experienced-DDoS.

[2] Ahren Studer and Adrian Perrig. The coremelt attack. In *Computer Security–ESORICS 2009*, pages 37–52. Springer, 2009.

[3] Min Suk Kang, Soo Bum Lee, and Virgil D Gligor. The crossfire attack. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 127–141. IEEE, 2013.

[4] Soo Bum Lee, Min Suk Kang, and Virgil D Gligor. Codef: collaborative defense against large-scale link-flooding attacks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 417–428. ACM, 2013.

[5] Lei Xue, Xiapu Luo, Edmond WW Chan, and Xian Zhan. Towards detecting target link flooding attack. In *28th Large Installation System Administration Conference (LISA14)*, pages 90–105, 2014.

[6] Fida Gillani, Ehab Al-Shaer, Samantha Lo, Qi Duan, Mostafa Ammar, and Ellen Zegura. Agile virtualized infrastructure to proactively defend against cyber attacks. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 729–737. IEEE, 2015.

[7] Nick Mckeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *Acm Sigcomm Computer Communication Review*, 38(2):69–74, 2008.

[8] Seyed K Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: flexible and elastic ddos defense. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 817–832, 2015.

[9] P. Phaal, S. Panchen, and N. Mckee. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. RFC Editor, 2001.

[10] Ryan Izard. Floodlight openflow controller. http://http://www.projectfloodlight.org/.

[11] CloudLab. Cloudlab. http://www.cloudlab.us.

[12] OpenFlow. Openflow. http://archive.openflow.org/.

[13] Juan Wang, Yong Wang, Hongxin Hu, Qingxin Sun, He Shi, and Longjie Zeng. Towards a security-enhanced firewall application for openflow networks. In *The 5th International Symposium on Cyberspace Safety and Security*, pages 92–103. Springer, 2013.

[14] Peyman Kazemian, George Varghese, and Nick Mckeown. Header space analysis: static checking for networks. pages 113–126, 2012.

[15] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. What do packet dispersion techniques measure? *Proceedings - IEEE INFOCOM*, 2:905–914 vol.2, 2010.

[16] Zhen Zhang, Binqiang Wang, Shuqiao Chen, and Ke Zhu. Mining frequent flows based on adaptive threshold with a sliding window over online packet stream. In *Communication Software and Networks, International Conference on*, pages 210–214, 2009.

[17] Xin Hong Wang, Fu Qiang Liu, and Guang Xing Wang. Te routing algorithm to minimize maximum link utilization. Mini-micro Systems, 2005.

[18] Markus M. Breunig. Lof: identifying density-based local outliers. *ACM SIGMOD Record*, 29(2):93–104, 2000.

[19] D Pokrajac, A Lazarevic, and L. J Latecki. Incremental local outlier detection for data streams. In *Computational Intelligence and Data Mining, 2007. CIDM 2007. IEEE Symposium on*, pages 504–515, 2007.

[20] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[21] Aman Shaikh, Anujan Varma, Lampros Kalampoukas, and Rohit Dube. Routing stability in congested networks: experimentation and analysis. *Acm Sigcomm Computer Communication Review*, 30(4):163–174, 2000.

[22] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Transactions on Networking (ToN)*, 9(3):265–280, 2001.

[23] Ning Wang, Kin Hon Ho, George Pavlou, and Michael Howarth. An overview of routing optimization for internet traffic engineering. *IEEE Communications Surveys & Tutorials*, 10(1):36–56, 2008.

[24] Kyriaki Levanti. Routing management in network operations. *Dissertations & Theses - Gradworks*, 2012.

[25] Amin Tootoonchian and Yashar Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3, 2010.

[26] Teemu Koponen, Martin Casado, Natasha Gude, and Jeremy Stribling. Distributed control platform for large-scale production networks, 2014.

[27] Pankaj Berde, Jonathan Hart, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Guru Parulkar, Guru Parulkar, Guru Parulkar, Guru Parulkar, and Guru Parulkar. Onos: towards an open, distributed sdn os. In *The Workshop on Hot Topics in Software Defined NETWORKING*, pages 1–6, 2014.

[28] Tina Tsou, Pedro Aranda, Haiyong Xie, Ron Sidi, Hongtao Yin, and Diego Lopez. Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains. 2012.

[29] Pingping Lin, Jun Bi, Ze Chen, Yangyang Wang, Hongyu Hu, and Anmin Xu. We-bridge: West-east bridge for sdn inter-domain network peering. In *Computer Communications Workshops*, pages 111–112, 2014.

[30] OpenDaylight. Opendaylight. https://www.opendaylight.org/.

[31] Seungwon Shin, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, Guofei Gu, and Mabry Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS*, 2013.

[32] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424. ACM, 2013.

[33] Hongxin Hu, Wonkyu Han, Gail Joon Ahn, and Ziming Zhao. Flowguard: building robust firewalls for software-defined networks. In *The Workshop on Hot Topics in Software Defined NETWORKING*, pages 97–102, 2014.

[34] Min Suk Kang, Virgil D Gligor, and Vyas Sekar. Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. *Network and Distributed System Security Symposium (NDSS)*, 2016.

[35] Abdullah Aydeger, Nico Saputro, Kemal Akkaya, and Mohammed Rahman. Mitigating crossfire attacks using sdn-based moving target defense. In *IEEE Conference on Local Computer Networks*, pages 627–630, 2016.

[36] Lei Wang, Qing Li, Yong Jiang, and Jianping Wu. Towards mitigating link flooding attack via incremental sdn deployment. In *Computers and Communication*, pages 397–402, 2016.

[37] Christos K Liaskos, Vasileios Kotronis, and Xenofontas Dimitropoulos. A novel framework for modeling and mitigating distributed link flooding attacks. In *IEEE INFOCOM*, 2016.