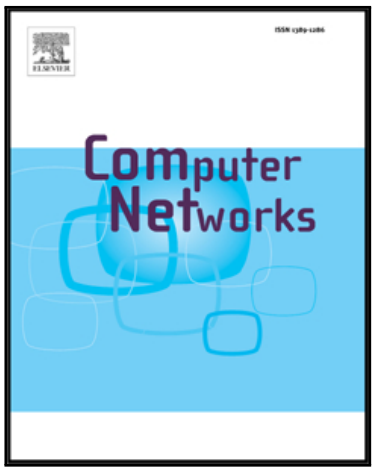
Accepted Manuscript



Woodpecker: Detecting and Mitigating Link-flooding Attacks via SDN

Lei Wang, Qing Li, Yong Jiang, Xuya Jia, Jianping Wu

PII: S1389-1286(18)30972-1
DOI: https://doi.org/10.1016/j.comnet.2018.09.021
Reference: COMPNW 6603

To appear in: *Computer Networks*

Received date: 15 February 2018
Revised date: 23 August 2018
Accepted date: 26 September 2018

Please cite this article as: Lei Wang, Qing Li, Yong Jiang, Xuya Jia, Jianping Wu, Woodpecker: Detecting and Mitigating Link-flooding Attacks via SDN, *Computer Networks* (2018), doi: https://doi.org/10.1016/j.comnet.2018.09.021

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

Woodpecker: Detecting and Mitigating Link-flooding Attacks via SDN

Lei Wang[a], Qing Li[b,*], Yong Jiang[a], Xuya Jia[a], Jianping Wu[c]

[a]*Graduate School at Shenzhen, Tsinghua University, Shenzhen, China*
[b]*Southern University of Science and Technology, Shenzhen, China*
[c]*Department of Computer Science and Technology, Tsinghua University, Beijing, China*

Abstract

Link-flooding attack (LFA), as a new type of DDoS attack, can degrade or even cut off network connectivity of a target area. This attack employs legitimate, low-density flows to flood a group of selected links. Therefore, these malicious flows can hardly be distinguished by traditional defense technologies. In our scheme, we first select M routers and upgrade them into SDN switches to maximize the network connectivity. Then, we propose a proactive probe approach to rapidly locate the congested links. Next, our scheme employs a global judgment algorithm to determine whether the network is under LFA or not. Finally, Woodpecker employs the core defense measure that based on the centralized traffic engineering to make the traffic balanced and eliminate the routing bottlenecks that are likely to be utilized by the adversary. We evaluate our scheme through comprehensive experiments. The results show that the bandwidth utilization of LFA-attacked links can be reduced by around 50% and that the average packet loss rate and jitter can be effectively decreased under LFA attacks.

Keywords: `Link-flooding Attack, DDoS, Software-Defined Networking`

1. Introduction

Recently, distributed denial of service (DDoS) attacks are the biggest threat to the availability of networks, applications and cloud services. The adversary generally explores resource asymmetry between the bots and victim servers, and abuses vulnerabilities of many network protocols to launch DDoS attacks [1, 2]. Many effective approaches have been proposed to detect and defend against the DDoS attacks, including Pushback [3], Ingress filter [4], PacketScore [5] and so forth. These methods all need to identify malicious traffic in advance, but this operation is very difficult for link-flooding attack (LFA) — a new type of DDoS attack.

Different from the traditional DDoS attacks, LFA floods a well-chosen group of links to cut off the network connections of a target area, instead of attacking the target servers directly. To this end, the adversary first detects the paths from bots to the public servers and constructs a link map accordingly. Then, the adversary floods the selected links by employing a large number of bots to send legitimate, low-density flows to the certain public servers. In this way, these congested links will severely degrade or even cut off the network connections of the target area. We show a simple example of LFA in Figure 1.

Over the last few years, LFA has quickly moved from the realm of academic curiosity [6, 7] to real-world incidents. We have already witnessed the real-life demonstration of

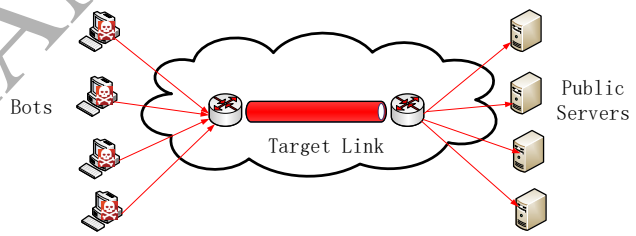


Figure 1: An Example of Link-flooding Attack

LFA in the core of the Internet [8, 9]. The target areas of these attacks include internet exchange points, enterprises and campus. Worth still, such an attack may be more frequent and massive due to inability to resist in reality.

LFA typically has two remarkable characteristics. **Undetectablity:** The target area is not directly attacked. Thus the servers in the target area cannot perceive any suspicious traffic. **Indistinguishability:** The adversary usually employs legitimate, low-rate flows with real IP addresses. Consequently, it is difficult to distinguish malicious flows from legitimate ones.

Because of the above characteristics, the traditional countermeasures, such as local rerouting and flow filtering based on traffic-intensity, have little effect on mitigating LFA. Moreover, LFA can change the selected links or the bot-server pairs periodically. Take a typical LFA — the Crossfire attack [6] as an example. Such attack alternately floods the optimal group of links for 3 minutes and another non-intersecting sub-optimal group of links for 30 seconds.

In summary, there are three significant challenges to de-

*Corresponding author
Email address: `liq8@sustc.edu.cn` (Qing Li)

ACCEPTED MANUSCRIPT

fend against LFA: 1) How to monitor the link congestion that is caused by LFA in time? In current networks, the distributed Internet protocols take a long time to diagnose the congested links as a failure. 2) How to determine the congestion caused by the LFA or other reasons? The lack of a global view of the network makes such judgement extremely difficult. 3) How to deal with the numerous attack flows that are almost the same with normal flows?

In this paper, we present a scheme called Woodpecker that can effectively mitigate LFA by software-defined networking (SDN). First, we upgrade the current routers/switches to SDN-enabled devices. Since the global view of network status and flexible flow manipulation are required in our scheme. Considering the complexity of the actual deployment, we propose an incremental deployment scheme that can maximize the available paths of the network. Then, we design a detection mechanism that can significantly shorten the response time to identify LFA. We use predefined rules as triggers to detect the congestion and probe the congested links with the help of the controller and upgraded devices. We also design an identification algorithm to judge whether the congestion is caused by the LFA according to the global congestion information. Finally, to mitigate the impact of LFA, Woodpecker employs centralized traffic engineering based on the upgraded nodes, which makes the traffic balanced enough to eliminate the routing bottlenecks. Additionally, we design an algorithm to drop some packets according to a blacklist as an emergency measure when LFA is beyond the capacity of the network, which means packet dropping is inevitable.

To demonstrate the performance of Woodpecker, we simulate the scheme by comprehensive experiments based on the real Internet topologies of Rocketfuel [10]. The results show that: 1) the bandwidth utilization of LFA-attacked links can be reduced by around 50%, even though in different network topologies; 2) the average packet loss rate and jitter can be adequately mitigated after Woodpecker has taken effect.

We organize the other sections of this paper as follows. We first introduce background and related work in Section 2. We formalize the threat model for LFA in Section 3 and outline our Woodpecker scheme in Section 4. Then we describe our incremental deployment algorithm in Section 5 and design an approach to rapidly locate the congested links and judge the attack type in Section 6. Next, we illustrate the LFA defense measures in Section 7. Finally, we evaluate Woodpecker by real network topologies in Section 8 and draw the conclusion in the last section.

A preliminary version of this work appeared in a conference paper [11] which briefly discussed ideas of this work. In this paper, we formalize the threat model and analyze the design decisions theoretically. We also redesign our link congestion detection mechanism and develop a novel algorithm to determine whether the congestion is caused by LFA. Additionally, we have added the evaluations of the new LFA detection mechanism and the effectiveness of our prototype under different network conditions.

2. Related Work

Although link-flooding attack (LFA) is one of the latest types of DDoS attacks, there have been some pioneering research works on the characteristics of LFA and how to defend against this attack. Our scheme is inspired by many previous works, and we cover them briefly as follows.

2.1. Background of LFA

In recent years, LFA has become a hot topic in the academia. Coremelt attack [7] which is proposed by Studer et al. is the earliest LFA. It uses a large number of bots and causes significant congestion in the target links through the traffic between these bots. An improved version of the LFA is the Crossfire attack [6]. This attack uses bots as sources and some public servers as destinations and floods the target links through the traffic between the sources and the destinations. This type of LFA is easier to enforce in reality because it only needs to continually probe the public servers in the target area to construct the link map without having strict requirements on the deployment of bots like the Coremelt. Therefore, in this paper, we focus on the Crossfire like LFA.

Besides of the academic curiosity, LFA has already occurred in the real world. The well-known technology bloggers (Ars Technica) have reported such attacks several times, including attacking Internet exchanges on April 2nd, 2013 [9] and an email provider on November 10th, 2015 [8].

LFA is completely different from the brute-force DDoS attacks that aim to exhaust the resources of the end target (e.g., computation, memory, or bandwidth). Instead, LFA creates a large number of low-density attack flows crossing the targeted links to flood and virtually cut off them as described in Figure 1.

Recent research [12] reveals that scalable LFA can be feasible because of the ubiquitous routing bottlenecks. Such bottlenecks are defined as the power-law distributions of link occurrence in the paths of chosen destinations and are caused by the shortest-path-first principle of the current routing protocols.

Lei Xue et al. proposes a network measurement system, called LinkScope[13], to capture abnormal path performance degradation for detecting LFA. This work is an excellent way to detect the heavy congestion caused by LFA, but it is required to deploy many additional probe points. Moreover, this work does not consider the distribution of all the congested links and lacks related countermeasures to mitigate or defend against LFA.

2.2. State-of-the-art Defense Strategies

To defend against LFA, [14] presents a scheme Codef that uses collaborative rerouting and rate control strategies to lead the adversary into a decision-making dilemma. Though this scheme is a good way to identify the malicious traffic, it introduces a specialized server, named as router controller, to maintain a routing control message in each

ACCEPTED MANUSCRIPT

participating AS. This message manipulates a multi-path routing policy in the network. Thus, this scheme is known to be harder and expensive to orchestrate and deploy. G. Dimitrios also uses an SDN framework and similar measures to mitigate LFA, but its local rerouting approach is also difficult to take effect in reality [15].

To improve the feasibility of Codef [14], the team proposed SPIFFY [16]. This improved scheme employs SDN to perform a rate change test, where we temporarily increase the effective bandwidth of the bottlenecked core link and observe the response. It also games with the attacker through the rate or the path selection. However, SPIFFY makes a strong assumption that the adversaries who try to utilize the bots' upstream bandwidth fully will be detected. Since the throughput of attack traffic is unable to increase proportionally with bandwidth expansion. Another assumption is that all attack flows must be TCP-based. Our scheme does not have these restrictions, and we will show the threat model analysis and design decisions in the following sections.

Similarly, [17] proposed a software-defined honeynet to mitigate the LFA. This scheme leverages global network visibility of SDN to infer the potential honey nodes which connect the honeynet. The scheme can increase the attack cost by selecting the furthermost node from the honey node as a final response node in order to force the attacker to fully visit the honeynet topology. This scheme can be integrated with ours as a defensive measure before the global traffic engineering. However, in [17], this scheme is only a preliminary idea, and it needs many details to be supplemented and improved.

LFADefender [18] also uses SDN framework to infer the possible target links. By monitoring these links, this scheme uses traffic rerouting and malicious traffic blocking to mitigate the LFA. Although the defense process of this scheme is similar to ours, there are significant differences in the specific implementation. Our scheme employs a probe-based method of locating the target links and design an algorithm for attack judgment since we cannot consider all link congestion is an LFA. Moreover, we use optimized TE instead of rerouting the traffic in the target links.

2.3. DDoS Defense via SDN

Software-defined networking (SDN) [19], which stems from the project 4D[20] and Ethane[21], has gradually moved from academic research to industrial deployment. Separating the control plane from the data plane and logically centralized control are its main features [22, 23]. Moreover, OpenFlow [24, 25, 26], as the de facto standard of SDN southbound protocol, supports rich and flexible match fields and programmability, which make it easy to implement fine-grained flow control [27, 28, 29]. Owing to these features, SDN offers a promising approach to resolve security issues.

[30] proposes a neural network-based approach to detect DDoS attack using SDN framework. This approach extracts six critical features of traffic and employs an unsupervised Self-Organizing Map (SOM) to classify the pattern of traffic. It is feasible for the traditional DDoS attacks, but not for LFA. Because according to these selected flow characteristics, the attack flows in LFA are almost the same as the normal ones. Therefore, our work, which aims at balancing all the traffic by TE to mitigate the effect of LFA, is orthogonal to the traditional DDoS defense work.

There are many other research works [31, 32, 33] focus on defending against tradition DDoS attacks. Among them, an influential scheme is Bohatei [31], which steers suspicious traffic through the defense VMs while minimizing user-perceived latency and network congestion. It proposes an adaptation strategy to handle dynamic adversaries that can change the DDoS attack mix over time, and a proof-of-concept implementation to handle several traditional DDoS attack types using industry-grade SDN/NFV platforms. The premise of these schemes is to be able to identify DDoS attack traffic, but this is very difficult for LFA. Our scheme is designed without the ability to identify LFA attack traffic.

A realistic problem that we must consider is that it is risky and unrealistic to replace all the network infrastructures with SDN devices on a flag day due to diversity and complexity of the network. Therefore, incremental deployment is required. Dan Levin et al. propose a feasible scheme Panopticon [34] for incremental deployment, and show that the SDN benefits can extend over the entire network.

In our scheme, traffic engineering via SDN is a primary measure in our scheme. There are many related works on traffic engineering in fully deployment SDN, such as B4 [35] and SWAN [36]. They are both extremely solutions for the traffic management across data centers in Google and Microsoft respectively. [37, 38] also implement feasible traffic engineering in the incremental SDN deployment scenario, and they provide an excellent reference for our traffic engineering scheme. The difference is that these works do not emphasize the selection of SDN nodes. While in Woodpecker, we believe that an optimal selection of SDN nodes is crucial for the maximization of network connectivity. [39] shows that only 30% of the SDN nodes are deployed, the traffic engineering can obtain near-optimal performance. In our scheme, our evaluation results also show that even in a low deployment rate, Woodpecker can mitigate LFA effectively.

3. Threat Model

In this section, we describe a typical attack and defense scenario of LFA in this paper and then formally summarize the adversary's goals and constraints.

3.1. Terminologies

To illustrate the threat mode clearly, we first define some terminologies in an attack and defense scenario of LFA shown in Figure 2.

ACCEPTED MANUSCRIPT

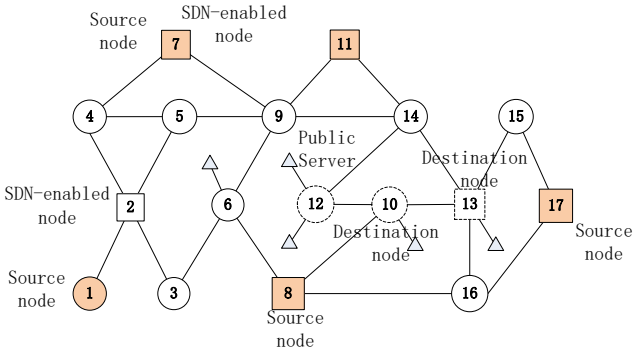


Figure 2: A Typical Attack and Defense Scenario of LFA

Source node and **destination node:** The source node is an ingress switch that connects to other networks, and we color them in Figure 2 (e.g., s_1, s_7, s_8). The destination node (dotted line in Figure 2 , e.g., s_{10}, s_{12}, s_{13}) is the node that connects to the public servers. We use S to denote the set of source nodes and D to denote the set of destination nodes. In this figure, the triangle denotes a public server, which can be accessed by any user. Since the attack traffic is usually from the bots that are outside of the target network, we assume that all attack flows come from S and flow into D to simplify the representation of the flow behavior. That means the topology in our discussion only consists of the switches and the links between them, and the internal traffic is considered as the background traffic. Such assumption only shortens two hops of the actual path and does not affect the flow behaviors in the network. Therefore, it does not affect the correctness of our design.

Ordinary node and **SDN-enabled node:** The ordinary node (circle in Figure 2, e.g., s_1, s_3, s_5) is a traditional router or switch in the network, which speaks traditional protocols such as OSPF or IS-IS. We assume that for each destination IP or subnetwork, the forwarding behavior of an ordinary node will not change, that is, the packet will forward to a fixed next hop. These nodes can be upgraded into SDN-enabled nodes by supporting the south-bound protocol(s). An SDN-enabled node can forward packets to different neighbours even though these packets have the same destination IP since the packets can match other fields (e.g., source IP address or VLAN tag). These SDN-enabled nodes (square in Figure 2, e.g., s_2, s_7, s_{13}), although controlled by a remote controller, can communicate with the ordinary nodes according to the traditional protocols. Based on the information gathered by the SDN-enabled nodes, the controller can get a global view of the whole network.

LFA attack process: To launch an LFA, the adversary is required to enforce three steps: (1) the adversary first detects the paths from bots to the public servers in or around the target area and constructs a link map accordingly. To obtain the paths precisely, the adversary uses the traceroute tool multiple times. If a link that always appears on the paths between a fixed bot-server pair, it will be considered as a persistent link and included in the link map. (2) Some persistent links surrounding the target area are carefully chosen according to the flow density distribution. The flow density of a link is defined as the number of flows passing through between bots and decoy servers. (3) The adversary floods these selected links by employing a large number of bots to send legitimate, low-density flows to the selected public servers. In this way, these congested links will severely degrade or even cut off the network connections of the target area.

3.2. System Assumptions

Threat model: There are two basic assumptions of the threat model: (1) The adversary is rational and has limited resources. That means the adversary expects to maximize the damage to the network connections of the target area using as few resources as possible. (2) The network follows a per-flow fair-share allocation of link bandwidth since this mechanism is already widely applied in today's Internet. If the adversary sends faster than the fair-share rates, the attack flows can be detected by other security mechanisms like [40]. Formally, the LFA adversary pursues three goals:

1) Degradation ratio maximization: Based on the basic assumptions, we define r_{pre} as the per-flow fair-share for the flows between the source nodes and nodes of the target area under normal circumstances. Similarly, we define r_{attack} as the per-flow fair-share under LFA. The degradation ratio is defined as $(r_{pre} - r_{attack})/r_{pre}$. This index shows the degree of decline in the communication of the target area. The adversary would like to maximize the minimum degradation ratio of nodes in the target area or maximize the degradation ratio of the entire target area. To this end, the adversary must solve the generalized maximum coverage problem, which is a well-known NP-hard problem. Instead of finding an exact solution, the adversary usually uses an efficient heuristic algorithm, such as a greedy algorithm in [6]. Because we cannot know details of the greedy algorithm, we cannot accurately determine the attacker's chosen links only based on the network topology.

2) Attack cost minimization: A rational adversary will seek to minimize the cost of the attack. In this paper, we assume that the cost of the attack is proportional to the number of bots necessary for the attack. According to the research on the Pay-Per-Install (PPI) botnet markets [41], bots in the US or the UK are most expensive and cost \$100-\$180 per thousand bots, whereas bots in the rest of the world cost less than \$10 per thousand bots. The survey of [6] presents that the total cost of the Crossfire attack is roughly $\$46K$. That means the adversary will make full use of the bots.

3) Attack persistence: To circumvent detection, the adversary uses legitimate, low-rate flows with real IP addresses. That means the attack flows have the same legitimate traffic patterns as the legitimate flows, and cannot be distinguished from legitimate ones via traffic analysis of headers/payloads at the target links. The adversary

ACCEPTED MANUSCRIPT

accesses the public server and gets the available services. Whether the traffic is TCP or UDP actually is determined by the service of the public servers, although TCP traffic constitutes the majority (90% ∼ 98%) of the Internet backbone traffic. Thus, in this paper, we do not specify the type of attack flows, which is different from the assumption in [16].

Based on this threat model, we develop and evaluate Woodpecker in the following sections.

4. Woodpecker Design Overview

In this section, we present the overall design of Woodpecker. We discuss the high-level idea behind our scheme and a rough introduction to the system design on how to detect and mitigate the LFA.

4.1. Design Decision

Employing legitimate, low-density flows to flood selected links is the hallmark of LFA. Essentially, the attack flows of LFA have no difference from normal flows, no matter in the packet header or the payload. Therefore, identifying the attack flows of LFA is a costly task, and it is likely to result in a high false positive rate.

Min Suk Kang and Virgil D. Gligor's research reveals that ubiquitous routing bottleneck of Today's Internet is the root cause of LFA [12]. This work shows that the routing bottleneck is a fundamental property of Internet design; i.e., it is a consequence of route-cost minimizations.

Our scheme avoids this dilemma ingeniously and expects to eliminate the routing bottlenecks. According to the threat model, the rational adversary will carefully select a group of links that can appear as much as possible in the communication paths of the target area. Moreover, the adversary hopes that this group contains the least number of links. To achieve this, we balance all traffic regardless of whether it is likely to be malicious traffic. That means the chosen links will not be the bottlenecks if there are enough feasible paths in the network to balance the traffic.

Additionally, due to the shortest-path-first principle of current network protocols (e.g., OSPF and IS-IS), a source-destination pair generally has only one path, i.e., the shortest one. This rigid routing principle decreases the connectivity of the network tremendously, and the connectivity affects the number of routing bottlenecks.

Our scheme addresses these challenges through the deployment of software-defined networks. Flows between a source-destination node pair in SDN have many feasible paths since these flows can select different matching fields for forwarding in the SDN-enabled nodes. These added feasible paths are used to balance the traffic. In our scheme, we use the average and the minimum number of paths between the source-destination node pairs to evaluate the ability of the network to balance traffic after SDN deployment.

Another challenge is to determine whether LFA incurs link congestion or not. Slow convergence of diagnosing congested links as a failure is a critical vulnerability in the current Internet. The default time interval of "Hello" packet in OSPF is 40 seconds and the average time for an IGP router diagnosing the congestion as a failure is 217 seconds [42]. The adversary can even take full advantage of this feature to attack another disjoint group of links dynamically[6]. In order to detect this attack quickly and precisely, we need to locate the congestion link first and obtain global congestion information. Taking into account the principle of LFA, we need to consider all the congestion in the network as a whole to analyze. The fast locating of the congested links and the analysis of the whole network of congestion information both require programmability of the data plane and logically centralized control. These features are hard to achieve in the traditional network, but the advantages of SDN. Considering the scalability issue-sour scheme uses an logically overlay SDN. That means the data plane can also enforce the traditional routing protocols independently. The controller only installs the high-priority rules in the SDN-enables nodes to steer the traffic.

Although OpenFlow [24, 25, 26] is the de facto standard of SDN southbound protocol, our scheme does not depend on this specified protocol. The features of SDN including global network view, flexible flow manipulation and rich match fields are the fundamental of our scheme. That means replacing OpenFlow to other southbound protocol, such as Netconf and BGP-LS, will affect little on the effectiveness of our scheme. Additionally, it is risky and undesirable to replace all the network infrastructures with SDN devices on a flag day. Therefore, support for incremental deployment is required. That means both the attack detection and defense measures must consider the case of incremental SDN deployment.

4.2. Framework of Woodpecker

In this subsection, we present the framework of Woodpecker and describe the process with pseudo-code in Algorithm 1. To illustrate the scenario clearly, we also show an example of a network applying Woodpecker in Figure 3.

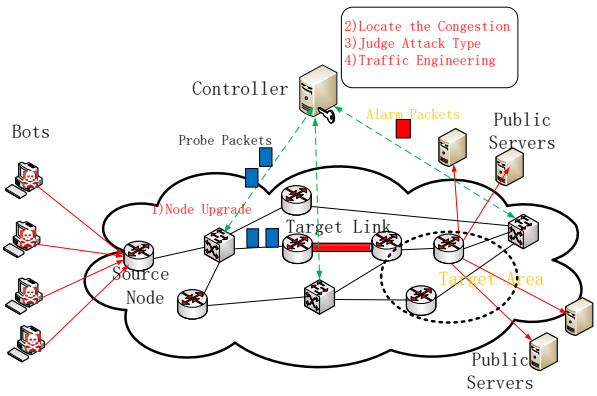


Figure 3: An Example of a Network Applying Woodpecker

The first step of Woodpecker is the SDN deployment, which is the prerequisite for our program. Considering the

ACCEPTED MANUSCRIPT

incremental deployment scenario, Woodpecker chooses a generalized upgrade plan, i.e., choosing M ordinary nodes to upgrade into SDN-enabled nodes while $M \leq N$ and N represents the total number of all switches. Thus, the controller can obtain the global network view and implement centralized control of the SDN-enabled nodes.

Woodpecker installs the rules of the corresponding measurement indicators in the SDN-enabled nodes in advance. When the congestion incurred, the corresponding rules will be triggered, which can coordinate with the controller to quickly locate the congestion link. The congested link will be included in the set E_c, and $\#(E_c)$ denotes the number of links in the set E_c. With the help of E_c and the topology information of the network, the detection module of Woodpecker can determine whether an LFA causes the current congestion.

Once an LFA is ascertained, the defense mechanism based on a centralized traffic engineering will be enforced to balance the attack flows. The controller calculates the ratio of flow distribution on each path and then implements the path reallocation of part of the flows by installing high-priority forwarding rules on the SDN-enabled nodes.

After this step, Woodpecker repeats step 2 to obtain current global congestion information and records them in E'_c. The number of new congested link set ($\#(E'_c)$) is expected to be less than β, ideally zero. The parameter β indicates the critical number of the congested links that the network can tolerate. However, if the LFA still exists, Woodpecker will compare E_c and E'_c and the new flows over the links of E'_c to generate a blacklist. Woodpecker will drop some packets according to the blacklist as an emergency measure. As Woodpecker aims to defend against the LFA, the network will return to the initial configuration when no LFA attack has been detected for a period.

Algorithm 1 Framework of Woodpecker

1: Calculate the most appropriate nodes to upgrade into SDN-enabled nodes, and get the upgrade nodes set $Update[M]$;
2: Locate the congested link e_i between two SDN-enabled switches. Put e_i into the congested links set E_c;
3: bool result = JudgeAttack (G(V,E),E_c,S,D);
4: **if** $result == true$ **then**
5: Enforce the centralized traffic engineering;
6: **else**
7: Use backup path or local reroute;
8: **end if**
9: Repeat 2, Get the new congestion link set E'_c;
10: **if** $\#(E'_c) > \beta$ **then**
11: Drop some packets according to the blacklist as an emergency measure;
12: **end if**
13: After a period T, restore the initial network configuration;

An observation is that the congested links caused by LFA are impossible to be the links between the controller and SDN-enabled devices. Thus, the communication between the controller and the SDN-enabled devices cannot be broken. The reasons are as follows: 1) The adversary detects the same path multiple times to ensure the persistent links, and the flow entry will set up after the first probe. i.e., next time the packet will never be forwarded to the controller. 2) In our scheme, the links between the controller and SDN switches are required to be out of the band. Thus the links that are between the controller and the switches are transparent for probe tools.

To implement our scheme, we are required to tackle a number of challenging issues, including: 1) How to develop an optimized SDN deployment solution when upgrading a limited number of nodes; 2) How to quickly and accurately detect the LFA; 3) How to design a fast and effective traffic balancing mechanism under the incremental SDN deployment. We will discuss the details in the following sections.

5. Optimal Upgrade Nodes Selection Policy

As discussed in previous, Woodpecker is required to be enforced through SDN deployment. We look forward to the full deployment of SDN. However, the one-step SDN upgrade of entire networks is practically impossible since it poses an enormous operational burden, and also raises performance and security risks. Therefore, incremental deployment is required. In this section, we will discuss a more general problem, i.e., how to choose the most appropriate M ordinary nodes to be upgraded into SDN-enabled nodes. This problem can be formally described as follows.

Upgrade nodes selection problem: Given a network (G,V,E), where V is the set of nodes, and E is the set of edges. We assume that all the flows come from source nodes S, and flow into destination nodes D. The problem is choosing M nodes to upgrade to maximize the connectivity and controllability of the network. We can think of a whole network SDN deployment as a special case of $M = \#V$

In this scenario, maximizing the controllability can be transformed into finding a minimum vertex covering set problem. Vertex covering set of a graph means that each edge of the graph has at least one node in the set. As a result, each link contains at least one SDN-enabled node, and all paths are controllable. For example, a node set $\{s_2, s_3, s_5, s_7, s_8, s_9, s_{10}, s_{13}, s_{17}\}$ is a vertex covering set of the graph shown in Figure 2. Moreover, in the network, we should consider the direction of the flow. That means, for a directed edge $i \rightarrow j$, the path can be selected only if the node i is an SDN-enabled node. Thus, we need to upgrade more nodes. In practice, the number of minimum vertex covering set is usually larger than the actual number of selected nodes.

From Whitney's theorem [1], we know that increasing the

[1] Whitney's theorem is a derivation of Menger's Theorem, which

ACCEPTED MANUSCRIPT

minimum number of disjoint paths from node pairs equals to increase the connectivity of the graph. Recalling the motivation of our SDN deployment, we expect to increase more paths between S and D and make these paths available to balance the traffic. Based on this consideration, the problem can be illustrated in Equation (1-4).

$$\max \quad N_{min} \tag{1}$$

$$\text{s.t.} \quad \sum_{i=0}^{n} d_i \leq M \tag{2}$$

$$1 + \sum_{s_i \in V} d_i \Delta P_{i,p,q} = N_{p,q} \tag{3}$$

$$N_{min} \leq N_{p,q} \tag{4}$$

$$\forall p \in S, \forall q \in D, d_i \in \{0, 1\}$$

We use N_{min} to denote the minimum number of feasible (loop-free) paths among all the node pairs which come from the source node set S and the destination node set D respectively. The objective of this optimization problem is to choose M nodes to upgrade, which can make N_{min} reach the maximum value. In constraint conditions, d_i is a boolean value, which indicates whether the node s_i will be upgraded or not. $\Delta P_{i,p,q}$ denotes the number of increased paths between the node s_p and s_q, when the node s_i is upgraded into a SDN-enabled node. Sequently, we discuss the algorithm complexity of this problem.

Lemma 1. *The upgrade nodes selection problem is an NP-Complete Problem.*

Proof. We can prove the complexity of the upgrade nodes selection problem with the following steps [43, 44]:

Step 1 (Problem Transformation): Since this is an optimization problem, there must exist an equivalent decision problem. This decision problem is such that given a path number ($NPath_c$), is there a solution satisfies all of the constraints above, and the increased paths for each source-destination pair are less than or equal to $NPath_c$.

Step 2 (NP problem proof): We assume that the number of nodes of the network is N_G, i.e., $\#G = N_G$. The solution of the decision problem is a boolean vector ($D = [d_i]_{N_G}$). The target function is a polynomial function of the solution. Thus, we can verify whether the increase paths under the selection vector D are less than or equal to $NPath_c$ or not in polynomial time. According to this condition, we can conclude that the problem of upgrade nodes selection is an NP problem.

Step 3 (NP-hard proof): To prove this problem is NP-hard, we show that the integer linear programming (ILP) [45] $\leq_p$ the upgrade nodes selection problem, i.e., we need to show how to reduce any an instance of the ILP to an instance of the upgrade nodes selection problem in polynomial time. We rewrite the formal expression of the upgrade nodes selection problem and get as follows:

$$\max \quad N_{min} \tag{5}$$

$$\text{s.t.} \quad \sum_{i=0}^{n} d_i - M \leq 0 \tag{6}$$

$$-\sum_{s_i \in V} d_i \Delta P_{i,p,q} + N_{min} - 1 \leq 0 \tag{7}$$

$$\forall p \in S, \forall q \in D, d_i \in \{0, 1\}$$

We find that our problem is a typical ILP, since both the objective function and constraints are linear, and the value of all variables can only take 0 or 1. The only complicated problem is how to obtain the value $\Delta P_{i,p,q}$. However, if the solution is given, we can get all the feasible paths between any source-destination pair. In this way, we can also consider the $\Delta P_{i,p,q}$ as a constant value in the decision problem. Therefore, any a valid solution in the ILP can satisfy the upgrade nodes selection problem. This means that ILP can reduce to the upgrade nodes selection problem in polynomial time. As ILP is a well-known NP-hard problem [44], we can infer that the upgrade nodes selection problem is also NP-hard.

Considering all the three steps, we can conclude that the upgrade nodes selection problem is NP-Complete. □

We propose Algorithm 2, which is a heuristic algorithm, to solve this NP-Complete problem. Two parameters are used to measure the benefits of upgrading a certain node. The first one is an array $Coun\_fre[N]$ that records the number of occurrences of the nodes in all the paths between S and D. A matrix N_PATH is constructed to record the shortest path between each node pair based on Dijkstra [46] algorithm, and this matrix can be traversed to calculate the value of $Coun\_fre[N]$. This index demonstrates the importance of the node in all the possible paths of attack flows. In other words, the bigger the value is, the more paths this node (if upgrade) can affect. The other index $degree[i]$ shows the potential of a node for balancing the traffic. That means if a node with a higher degree is upgraded, it may generate more paths. Then, the algorithm normalizes these two indexes, and use a weighted sum ($Ben[i]$) to indicate benefits of the whole node upgrade operation. The parameter α_1 and α_2 indicate the weight of $Coun\_fre[i]$ and $degree[i]$ respectively, and we set $\alpha_1 = \alpha_2 = 0.5$ in our scheme. In fact, the two parameters can be tuned according to the network topology and the selection of source-destination node pairs. Finally, the first M nodes are chosen to upgrade according to Ben. The complexity of this algorithm is dominated by the Dijkstra algorithm. Therefore, the upper bound of the time complexity is $O(nm + n^2 logn)$ if the Dijkstra algorithm

can be illustrated as: Let X and Y be disjoint sets of vertices in a k-connected graph G. Let u(x) for x ∈ X, and w(y) for y ∈ Y be nonnegative integers such that $\sum_{x \in X} u(x) = \sum_{y \in Y} w(y) = k$. Then G has k pairwise internally disjoint X, Y-paths so that u(x) of them start at x and w(y) of them end at y.

is implemented with the Fibonacci-Heap data structure. The parameter n is the number of the vertexes and m is the number of the edges in the network.

We also take the scenario of Figure 2 as an example, and set $M = 5$. We traverse all paths between any nodes in the network and counter the occurrences between the source nodes and destination nodes. For example, node S_{14} appears in 9 of the 15 paths, and the degree is 4. According to algorithm 2, this node is the top 5 nodes in the weighted sum vector (Ben) and should be upgraded to SDN-enabled nodes.

After we choose the M ordinary nodes, we will replace them with an SDN-enabled switch. We do not need to modify anything for ordinary nodes. Therefore, they speak traditional protocols (e.g., OSPF or IS-IS) as usual. In the SDN-enabled node, the controller will add serval permanent flow entries (i.e., hard_timeout = 0) to match the protocol type of traditional protocols and set the action as "Forward to Controller". When a packet from ordinary nodes comes, it will match the flow entry that is set in advance, and be forwarded to the controller. Then controller will calculate the route and add a new flow entry for this flow using a new match field, such as some bits of MPLS. The packet will be forwarded to the specified port according to the new match field. In other words, the operations in SDN-enabled nodes are transparent to ordinary nodes.

Algorithm 2 Upgrade Node Selection Algorithm

Require: G(V,E),S,D
Ensure: $Update[M]$
1: initialize: $Coun\_fre[N] = \{0, 0, ...0\}$
2: **for** each node $s_i \in$ V **do**
3: **for** each node $s_j \in$ V **do**
4: $N\_PATH(i, j) = Dijkstra(i, j)$
5: **end for**
6: **end for**
7: **for** each node $s_p \in$ S **do**
8: **for** each node $s_p \in$ D **do**
9: **if** node $s_i \in$ N_PATH(p,q) **then**
10: $Coun\_fre[i] + +$;
11: **end if**
12: **end for**
13: **end for**
14: **for** $i = 0$ To $n - 1$ **do**
15: $Ben[i] = \alpha_1 * Coun\_fre[i]/maxRecord + \alpha_2 * Degree(i)/maxDegree$
16: **end for**
17: Choose the largest M element from $Ben[N]$ and put them to Update[M]

6. Congestion Location and Attack Detection

In this section, we design an LFA detection module to quickly locate the congested links and judge whether the congestion is caused by LFA.

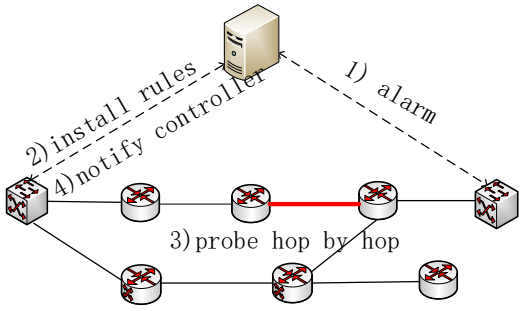


Figure 4: Locate the Congested Link

6.1. Locate the Congested Links

So far, SDN does not have a standard congestion detection mechanism. Since SDN has a centralized controller and the built-in statistic collection features, we develop a congested link location mechanism that combines path analysis with hop-by-hop probing.

To achieve this, we install the predefined flow rules as measurement triggers on each SDN-enabled node. When a packet comes to an SDN-enabled node, the related packet or bit counter will be updated. This operation is a standard function in the implementation of existing SDN switches. Each SDN-enabled node will calculate the rate of some specified flows based on the statistical information, and check whether they meet the predefined trigger condition or not. As congestion caused by LFA aims at cutting off the selected links and leading to a congestion collapse event, the predefined condition is usually set as the lower bound of traffic rate which depends on the link capacity and the maximal number of allowed concurrent flows. We can also set the change of traffic rate as the trigger condition. Although the flow rate that is measured by polling the counters may not be accurate, a significant reduction of this rate can indicate congestion efficiently. In this paper, we use the latter triggers.

When the flow activates the trigger, the SDN-enabled node (e.g., s_2) will send an alarm message to the controller. The alarm message contains the switch dpid, the port rate and the abnormal flow information. Based on the alarm information, we can infer the abnormal path which contains the congested link. An import observation is that, for the congested link, although the rate of the specified flow declines significantly, the port rate is still nearing the maximum value. Additionally, as the centralized control of SDN, the controller knows all the forwarding paths of existing flows. With the help of global information from the controller, the detection module can infer one or more SDN-enabled node pairs, between which the severe congestion is possible to occur.

Then the detection module will implement the hop-by-hop probe to locate the congested links, and we show the process in Figure 4. For instance, the detection module infers that there is likely to be congestion between two SDN-enabled nodes, i.e., $s1$ and $s2$. First, it notifies the

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

controller to add two flow entries(rules) in node s_1. The flow entry consists of six domains: match fields, priority, counters, instructions, timeouts and cookies. The match fields of the two entries both contain IP protocol, and the value is 1 ("1" denotes the ICMP protocol). Additionally, in the first flow entry, destination IP is also included in the match field as a supplementary, and the value is the IP address of s_2. Meanwhile, it sets the instruction as forward to the port which connects to node s_2. In the second flow entry, the instructions domain is set as "forward to the controller". Then, this module makes the controller to construct an ICMP packet whose destination IP is s_2 and TTL is 1, and send it to node s_1 as a PACKET_OUT message. Instruction for the PACKET_OUT message is TABLE, which indicates that the switch should treat the packet as though it had been received on the input port. We set a higher priority for the two flow entries. As a result, the ICMP packet will match the first flow entry, and be forwarded to s_2. The ordinary nodes in a path will treat this packet as a common ICMP packet, and echo the packet.

When the ICMP echo arrives, it will match the second flow entry and be forwarded to the controller. If the reply is an ICMP unreachable message, the controller can get the location of the congested link according to the global topology. Otherwise, the controller will send a new ICMP packet to s_1 with the TTL domain plus 1 automatically. We only need to send a limited number of ICMP packets to locate the congestion because SDN-enabled nodes narrow the probe space tremendously. A special case is that if the alarm node is the first SDN-enabled nodes in the path, it will probe the congested link reversely. Finally, it collects the source IP addresses of all the flows which pass the congested links and record them. These IP addresses over the congested links will be used to generate the blacklist, which will be described in details in Section 7.2.

6.2. Judge the Attack Type

Link failure is a common phenomenon in the current network. Thus, we cannot take all congestion as LFA. To make our design lightweight, the detection module carries out an algorithm to determine whether the network is attacked by LFA. Recalling the threat model in section 3, LFA is not only a link congestion problem but also a group behavior of the congested links. Although the congestion of a single link has no difference from normal congestion, Woodpecker will get all congested links and analyze the damage of these congested links from a global view.

Cutting off the connectivity of the target area is the ultimate goal of LFA. That means if all paths between source nodes and the target area suffer congestion, the detection module can infer that the network is under the LFA. The prerequisite for this detection is that we already know the possible target areas in advance. These target areas are generally the most import areas in the networks, such as internet exchange, bank and other financial institutions, leading enterprises or universities. In our scenario,

Algorithm 3 Attack Judgement Algorithm

Require: G(V,E), S, D, E_c, $T_{area} = [\{s_i, s_i + 1\}, \ldots, \{s_k, \ldots, s_{k+t}\}]$
Ensure: Boolean isLFA

```
for each node s_i ∈ S do
    for each node s_j ∈ D do
        Get the current path(s) in edge_Path[i,j]
    end for
end for
for each edge e_i ∈ E do
    if edge e_i ∈ edge_PATH(p,q) then
        Record_edge[i] + +;
    end if
end for
if E_c cut off the connections of a target area in T_area[i] then
    isLFA = true;
    Return isLFA;
end if
Score = Σ_{i=0}^{#E_c} Record_edge[E_c[i]]
if Score ≥ κ then
    isLFA = true;
end if
Return isLFA;
```

we take the core routers or switches to indicate the target area (T_{area}).

In practice, the adversary may choose a part of the critical links and make them congesting. As a result, the communication of the target area will also be significantly degraded. To detect this behavior, Algorithm 3 evaluates the importance of all links in the whole network over current flow paths. Then the algorithm calculates a score of each congested links to show the occurrence of this link in all forwarding paths. To get the score of a link, all current paths from S to D are recorded. These paths are represented as an edge list. The algorithm uses a hash table to record the score of the link. If a link appears in a current forwarding path, the score of the link will be added by 1. The scores of current congested links are accumulated to check if it exceeds the upper bound value κ. The symbol κ is an empirical value, which denotes the threshold to distinguish LFA from normal congestions and depends on the topology and forwarding paths. Since the number of edges in the network is m, the upper bound of current forwarding path and the length of the path is less than m. That means the time complexity of Algorithm 3 is $O(m^2)$. A tighter bound is $O(M_{CP})$, where M_{CP} refers to the sum of the number of edges contained in all current paths.

We take a simple example to illustrate our attack judgement algorithm in Figure 5. Through the fast congestion detection mechanism, the controller can obtain the global view of network congestion information. In our example, there are five congested links (i.e., $s_8 \leftrightarrow s_{10}$, $s_9 \leftrightarrow s_{14}$, $s_{11} \leftrightarrow s_{14}$, $s_{13} \leftrightarrow s_{15}$, $s_{13} \leftrightarrow s_{16}$) in the network. We

ACCEPTED MANUSCRIPT

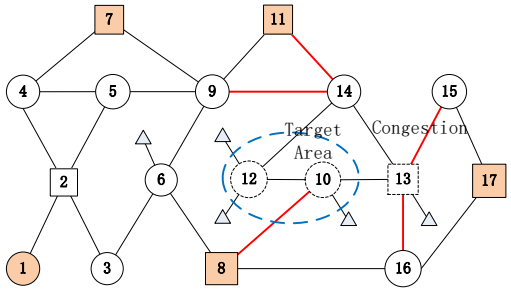


Figure 5: LFA Judgement

can find that the five congested links just cut off all the links between the target area and the source nodes. Thus, we can infer that the network is under the attack of LFA. Actually, even though part of these links (e.g., $s_8 \leftrightarrow s_{10}$, $s_{11} \leftrightarrow s_{14}$, $s_{13} \leftrightarrow s_{16}$) are congested, we can also determine the LFA is launched by the adversary since these congested links get a higher score ($Score = 11$) than the empirical value κ ($\kappa = 10$).

7. LFA Defense Measures

7.1. Centralized traffic engineering

The key module of Woodpecker is a centralized traffic engineering component that is implemented as an application upon the controller platform. This module receives the messages from the detection module and is activated when LFA is detected (i.e.,$isLFA = ture$). Link occurrence following a Zipf-Mandelbrot distribution in current Internet causes the routing bottlenecks that are utilized by LFA[12]. Therefore, the goal of our scheme is to balance the traffic on all links to eliminate the difference in the link utilization of the bottleneck links and other links. To this end, we take up the centralized traffic engineering with the help of the SDN-enabled nodes. We formalize this problem as follows.

$$\min \quad \theta \tag{8}$$

$$\text{s.t.} \quad \sum_{p \in Path_{i,j}} f_{i,j,p} = 1 \tag{9}$$

$$g_e + \sum_{i \in S} \sum_{j \in D} \delta_{p,e}^{i,j} T_{i,j} f_{i,j,p} \leq \theta c_e \tag{10}$$

$$0 \leq \theta \leq 1 \tag{11}$$

$$\forall i \in S, \forall j \in D, \forall e \in E$$

θ denotes the maximum utilization of a link, which is the optimization objective of this problem. Equation (9) shows the constraint of path split ratio for every node pair between S and D. We use p to present a feasible path (loop-free) between a node pair, and $Path_{i,j}$ to denote all the feasible paths between node s_i to node s_j. $f_{i,j,p}$ denotes the split ratio in the path p for the node pair. Inequation (10) is the link capacity constraint. In this inequality, $T_{i,j}$ denotes all the controllable traffic between the node pair s_i and s_j. The controllable traffic means this traffic passes at least one SDN-enabled node. $\delta_{p,e}^{i,j}$ is a link path indicator for the node pair (s_i, s_j), set to 1 if path p includes the edge e, and set 0 for otherwise. Therefore, the expression $\delta_{p,e}^{i,j} * T_{i,j} * f_{i,j,p}$ means the controllable traffic distributed in the path p.

In this condition, g_e denotes other traffic which passes the link e. Generally, g_e contains two kinds of traffic. One is the internal traffic that is generated by other nodes (not nodes in S) or flows into other nodes (not nodes in D). This traffic is considered as steady in this scenario. We can measure this traffic and get the value based on a statistical method. The other kind of traffic is the uncontrollable traffic that does not pass any SDN-enabled node on its path. We can infer the traffic by the total traffic and the controlled traffic. In practice, due to the upgraded node selection algorithm in Section IV, the traffic is expected to pass at least one SDN-enabled switch. We can omit the second kind of traffic, and set g_e a statistical value based on the first kind of traffic.

This optimization problem is linear programming, our module solves this problem by the lib PuLP [2] and gets the split ratio of each path. The complexity of this linear programming is polynomial time order. We also take the approach in [47] to speed up our algorithm.

The split ratio we get from the linear programming is the proportion of the traffic distributed on the entire path. It is required to calculate the split ratio for every port in each switch and discretize this ratio. It is straightforward that the ratio distributed in the last SDN-enabled switch is equal to the ratio of the path. This module can calculate the ratio from the last switch to the first one, and get the flow distribution for each port in every switch. Finally, the module will notify the controller to refresh the flow entries in the SDN-enabled nodes according to the distribution.

7.2. Supplementary Measure

In this section, we propose a supplementary measure in our Woodpecker scheme to handle the severe LFA in the network when it is beyond the capacity of the network. That means packet dropping is inevitable. Flows in LFA are legitimate, low-density and with real source IP addresses. Therefore, the adversary is probably to reuse many bots to generate the attack flows. Based on this analysis, we will drop the packets whose source IP address simultaneously appears in numerous congested links. First, we use a dictionary to record all the source IP addresses of the flows which pass the congested links at the step of "Locate the Congested Links" in Section V. Then we fetch k IP addresses which have the largest value (frequency of

[2] PuLP is a python lib for linear programming, which can support the python language controller like Pox or Ryu to easily solve this traffic engineering problem.

ACCEPTED MANUSCRIPT

occurrence) in the dictionary to generate a blacklist. Finally, we drop the packets according to the sequence in the blacklist, when packet dropping is inevitable.

8. Evaluation

In this section, we run several groups of experiments to demonstrate the effectiveness of Woodpecker, using the real network topologies in table 1. These topologies come from Rocketfuel [10]. Without loss of generality, we randomly choose two groups of nodes as the source node set S and the destination node set D. In our system, the number of the source node set S and destination node set D are $5\% - 10\%$ of the total number of nodes in these topologies.

Table 1: Experiment Topologies

AS Number	Nodes	Edges	Average Degree
AS 1	42	55	1.31
AS 174	45	78	1.73
AS 209	58	108	1.86
AS 577	29	33	1.14
AS 16631	22	32	1.45

8.1. Connectivity Improvement

The first step of Woodpecker is to choose and upgrade the optimal M nodes that can maximize the network connectivity. This step is not online and needs to be deployed in advance. We can evaluate the connectivity improvement by comparing the change in the number of paths after upgrading the selected nodes. For showing clearly, we define a new parameter upgrade rate $\tau = M/N$. M is the number of upgrade nodes, and N is the number of all nodes in the network.

We use two indexes (min_ASxx and avg_ASxx) to demonstrate the effectiveness of Algorithm 1 in Figure 6 and Figure 7. The index min_ASxx (e.g., min_AS1) indicates the minimum number of paths among all the S-D node pairs in the certain AS, and the index avg_ASxx indicates the average number of the paths, i.e., $avg\_ASxx = \frac{\sum_{i \in S, j \in D} path_{i,j}}{(\sum_{i \in S} i * \sum_{j \in D} j)}$. In our experiments, we limit the path within 7 hops, otherwise the situation of path stretch will be serious. We randomly change the source node set and destination node set serval times (20 times in our experiments), and get the average values for the two indexes.

In our experiment, the two value are both 1 before the node upgrade step, since the adversary only chooses the persistent links, and we omit the ECMP paths. Despite the different growth rate of these two indexes, it is clear that the minimum number of paths among all the node pairs is $2 - 5$ after the node upgrade step and the average number of paths varies from 10 to 1500 in different AS at different upgrade rate. These new paths are the foundation of our scheme, and they increase the connectivity of the network prominently. Moreover, the results show that if we upgrade about 40% of nodes, the routing bottlenecks can almost be eliminated, i.e., the adversary of LFA cannot find appropriate target links. In other words, the difficulty and cost of launching LFA will be significantly increased.

8.2. Attack Simulation and Detection

In our simulation experiments, we use Mininet[48] to construct the topologies, and Pox[49] as the controller. Our simulation platform runs at a server of Huawei RH2288 with the CPU of E5-2600 v2, the memory of 16GB DDR3. We calculate the forwarding table for ordinary nodes and write the forwarding rules in the flow tables in advance. Then, we create serval hosts (5 hosts for each source and destination switches in our experiments) and connect them to related switches. The hosts connected to the source nodes are used for generating the traffic to simulate the traffic from bots and legitimate users. The hosts connected to the destination nodes are used to simulate to the public servers. We use iperf tools as the traffic sources and sinks, and generate both the background traffic and attack traffic since both types of traffic have the same pattern. As far as we know, LFA has no public data set. Hence, we simulate this attack according to the paper [6], and flood a group of selective links to simulate LFA. In our experiments, we set all hosts (both bots and legitimate senders) to send 30 long-lived flows to flood the selected links. We do not simulate other internal traffic since Mininet can set the bandwidth of links to show consumption of the background traffic.

Woodpecker detects LFA based on SDN-enabled nodes. The trigger rules are installed in advance to detect severe congestion. Then they cooperate with the controller to locate the congested links and judge whether this congestion is caused by LFA. To evaluate the effectiveness of our LFA detection module, we use three indexes, i.e., precision(P), recall(R), and F1-Measure(F1):

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

$$F_1 = \frac{2TP}{2TP + FP + FN}$$

Here TP means an abnormal event we recognize as congestion that really was, and TN means an abnormal event we recognize as not congestion, and really was not. Similarly, FN indicates an abnormal event we recognize as not congestion, though it really is, and FP indicates an abnormal event we recognize as congestion, though it were not. We list our results in table 2. The results show that Woodpecker can detect severe congestion precisely, and the recall and F1-measure are also beyond 96.5%.

In our scheme, after the detection module collects the global link congestion information, we use algorithm 3 to determine whether the congestion is caused by LFA. To evaluate the detection rate of our algorithm, we congest

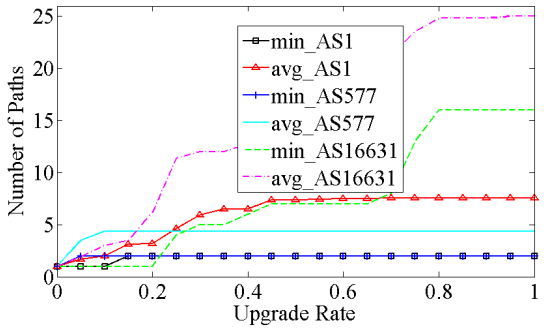


Figure 6: AS 1,577,16631

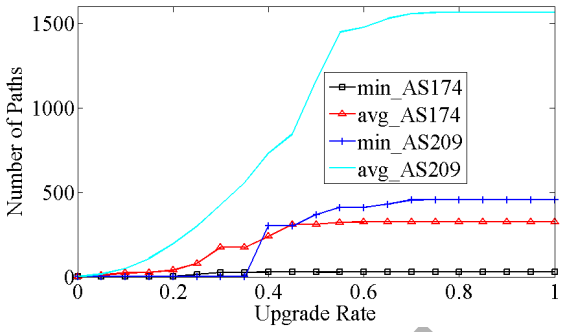


Figure 7: AS 174,209

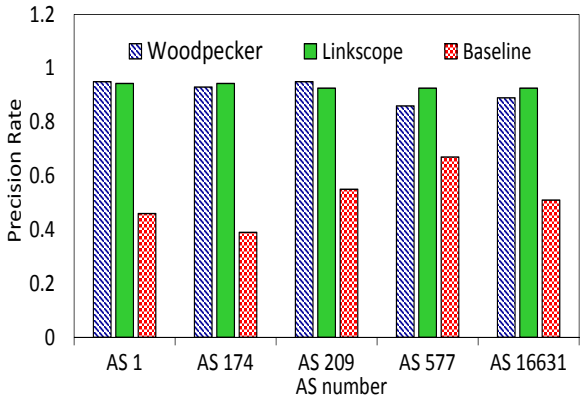


Figure 8: Precision Rate

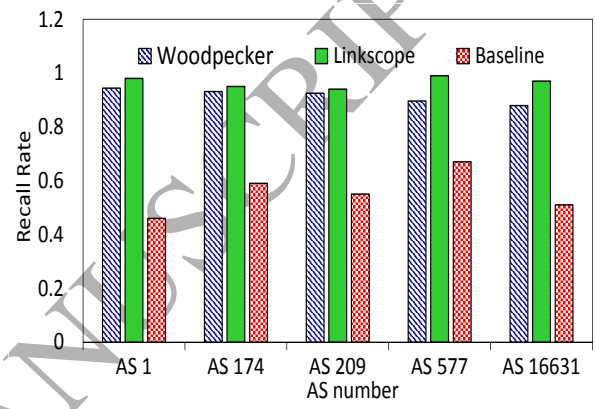


Figure 9: Recall Rate

Table 2: LFA Detection

AS Number	P	R	F1
AS 1	96.88%	98.61%	97.74%
AS 174	96.88%	100%	98.41%
AS 209	93.94%	100%	96.88%
AS 577	100%	98.61%	99.30%
AS 16631	96.88%	100%	98.42%

the target links and other non-critical links and show the results in figure 8 and figure 9. We compare our scheme with LinkScope [13] and the baseline method.

The baseline method is a naive detector method, and it counts the number of severely congested links at the same time. If this number exceeds 20% of all links, the baseline method considers that LFA happens. LinkScope takes topology analysis to select the critical links and employs both the end-to-end and the hop-by-hop network measurement techniques to capture abnormal path performance degradation for detecting LFA and then correlate the performance data and traceroute data to infer the target links or areas. LinkScope has a very high detection rate and false positive of less than 10%. However, it requires the additional deployment of probes and cannot easily adjust the location of these probes. The results show that our scheme has a similar precision rate and recall rate to LinkScope, and higher than the baseline method.

8.3. Evaluation of LFA Mitigation

The previous experiments show that when the upgrading rate reaches 40%, it can generate enough paths to balance the traffic. Therefore, we enforce our simulations at this upgrading rate and get the results in Figure 12. In our simulation experiments, we simulate the LFA, and flood the target links. The maximum link utilization in the target links is nearly 100% under LFA. After the controller detects LFA, Woodpecker will enforce the global traffic engineering. The max utilization in the x-axis in Figure 12 indicates the maximum value of link utilization among all the active links in the networks under different scenarios (i.e., select different target links and different attack area) and the y-axis show the related cumulative distribution function. The result shows that Woodpecker makes the maximum link utilization at about 50% in most cases and under 80% in all cases.

To verify the effectiveness of Woodpecker, we monitor the communication quality of the target area. We use iperf tools to send UDP packets from the hosts connected to source nodes to the certain servers in the target area. We compare the measurement results of the communication under the condition of launching Woodpecker or not in Figure 10—16. In our experiments, we use average loss packet rate (LPR) and average jitter to indicate the quality of the communication. The two indexes are defined as follows:

ACCEPTED MANUSCRIPT

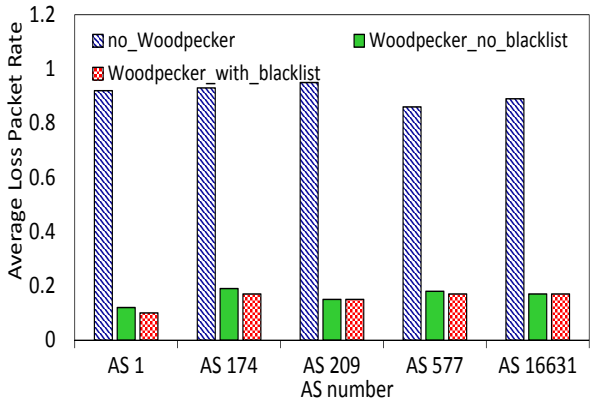


Figure 10: Average Loss Packet Rate, upgrade rate = 30%

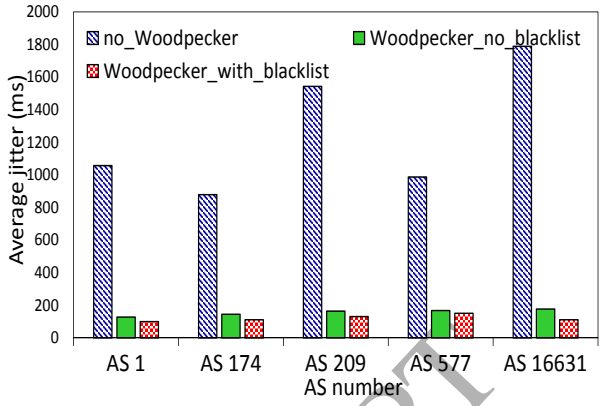


Figure 11: Average Jitter, upgrade rate = 30%

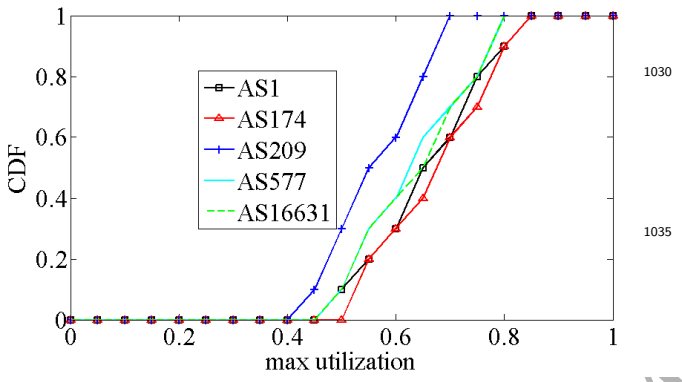


Figure 12: CDF of Max Link Utilization

$$LPR = \frac{send\_packets - receive\_packets}{send\_packets}$$

$$jitter = \frac{(rt(j) - st(j)) - (rt(i) - st(i))}{j - i}$$

In the definition of jitter, $rt(i)$ means the receive time of packet i, and $st(i)$ means the send time. The index jitter describes the variation of the end to end delay. The results show that when the networks are under LFA, the loss packet rate and the jitter are both in high values and they reach normal values after Woodpecker takes effect. We show the average value of the two indexes with the upgrade rate between 30% and 50% in Figure 10—16. In our simulation experiments, Woodpecker (without_blacklist) is good enough for mitigating LFA. Besides, we also evaluate the emergency measure (Woodpecker_with_blacklist). The results show that the blacklist only slightly degrades the two indexes, that means this step is just suitable to be a backup method as we expected.

Since the upgrade nodes selection operation is offline, the evaluation of the react time of Woodpecker does not include this operation. We infer the system execution time by observing changes in link metrics (loss of packets and jitter). While the detection time is obtained from the timestamp of delivering the traffic engineering related rules minus the timestamp of the packet of the first alarm. The results are shown in Table 3. The second column (ADT) indicates the average detection time, and the last column (AET) means the average system execution time. The results mean Woodpecker can significantly mitigate LFA in 5 seconds in most cases, and traffic engineering measure consumes most of the time in the whole defense process.

Table 3: Enforce Time of Woodpecker

AS Number	ADT (ms)	AET (ms)
AS 1	321	1669
AS 174	1773	4428
AS 209	2341	5086
AS 577	775	2509
AS 16631	924	2776

9. Conclusion and Future Work

In this paper, we propose Woodpecker to mitigate a new kind of DDoS attack—LFA. This scheme uses a heuristic algorithm to select a group of switches to upgrade into SDN-enabled switches. With the help of global view and data plane triggers, Woodpecker can fast locate the congestion and determine whether LFA causes the congestion through the global congestion information. To mitigate the LFA, Woodpecker enforces global traffic engineering to eliminate the bottleneck links. We evaluate the effectiveness of our scheme with the real topologies and get inspiring results.

In our future work, we will migrate Woodpecker to a hardware testbed with an industrial-level controller, and evaluate the suitability of our system in real life. Moreover, we will increase the versatility of the system to handle more types of DDoS attacks.

Acknowledgment

The research is supported by the National Natural Science Foundation of China under Grant 61625203,

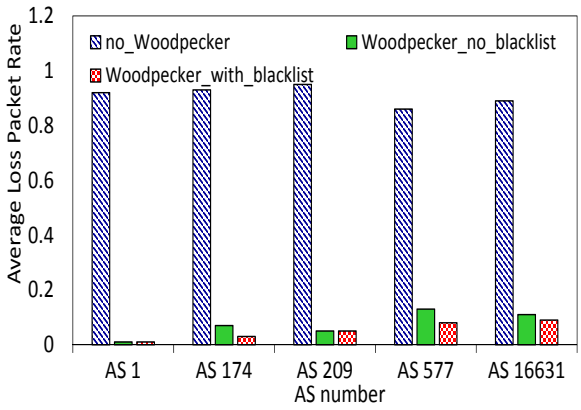


Figure 13: Average Loss Packet Rate, upgrade rate = 40%

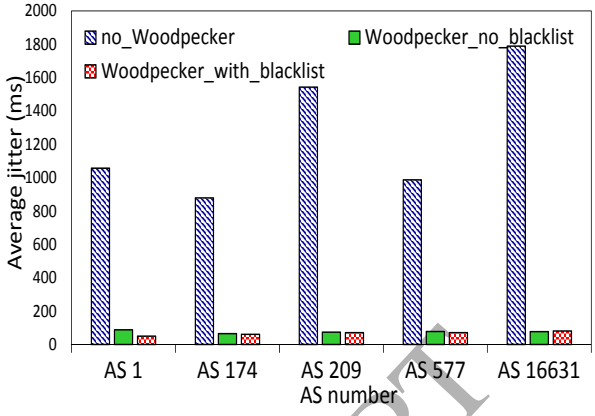


Figure 14: Average Jitter, upgrade rate = 40%

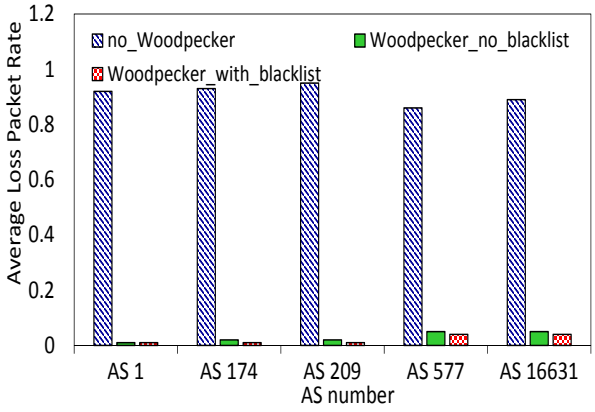


Figure 15: Average Loss Packet Rate, upgrade rate = 50%

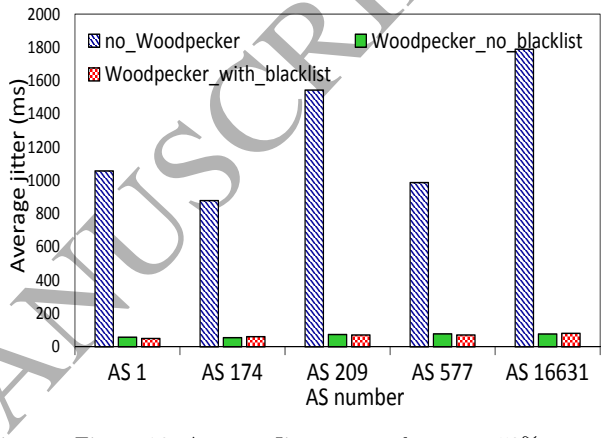


Figure 16: Average Jitter, upgrade rate = 50%

the National Key R&D Program of China under Grant 2016YFC0901605, the R&D Program of Shenzhen under Grant JCYJ20170307153157440 and JCYJ20160531174259309.

References

[1] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, D. Zamboni, Analysis of a denial of service attack on tcp, in: Proceedings of IEEE Security&Privacy, Oakland, USA, 1997.

[2] M. Kührer, T. Hupperich, C. Rossow, T. Holz, Exit from hell? reducing the impact of amplification ddos attacks, in: Proceedings of USENIX Security, SAN DIEGO, USA, 2014.

[3] J. Ioannidis, S. M. Bellovin, Implementing pushback: Router-based defense against ddos attacks, in: Proceedings of USENIX NDSS, San Diego, USA, 2002.

[4] F. Baker, P. Savola, Ingress filtering for multihomed networks, IETF RFC (2004).

[5] Y. Kim, W. C. Lau, M. C. Chuah, H. J. Chao, Packetscore: Statistics-based overload control against distributed denial-of-service attacks, in: Proceedings of IEEE INFOCOM, Hong Kong, China, 2004.

[6] M. S. Kang, S. B. Lee, V. D. Gligor, The crossfire attack, in: Proceedings of IEEE Security&Privacy, Oakland, USA, 2013.

[7] A. P. Ahren Studer, The coremelt attack, in: Proceedings of Springer-Verlag ESORICS, Berlin, Heidelberg, 2009.

[8] D. GOODIN, How extorted e-mail provider got back online after crippling ddos attack, Technica, Ars (2015).

[9] P. Bright, Can a ddos break the internet? sure just not all of it, Technica, Ars (2013).

[10] R. Teixeira, K. Marzullo, S. Savage, G. M. Voelker, Characterizing and measuring path diversity of internet topologies, in: Proceedings of ACM SIGMETRICS, San Diego, USA, 2003.

[11] L. Wang, Q. Li, Y. Jiang, J. Wu, Towards mitigating link flooding attack via incremental sdn deployment, in: Proceedings of IEEE ISCC, Messina, Italy, 2016.

[12] M. S. Kang, V. D. Gligor, Routing bottlenecks in the internet: Causes, exploits, and countermeasures, in: Proceedings of the 2014 ACM SIGSAC CCS, Scottsdale, USA, 2014.

[13] L. Xue, X. Luo, E. W. Chan, X. Zhan, Towards detecting target link flooding attack, in: Proceedings of USENIX LISA, Seattle, USA, 2014.

[14] S. B. Lee, M. S. Kang, V. D. Gligor, Codef: collaborative defense against large-scale link-flooding attacks, in: Proceedings of ACM CoNext, Santa Barbara, USA, 2013.

[15] G. Dimitrios, Cross-domain dos link-flooding attack detection and mitigation using sdn principles, Master's thesis, ETH Zurich (2014).

[16] M. S. Kang, V. D. Gligor, V. Sekar, Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks, in: Proceedings of USENIX NDSS, San Diego, USA, 2016.

[17] J. Kim, S. Shin, Software-defined honeynet: Towards mitigating link flooding attacks, in: Proceedings of IEEE/IFIP DSN Workshop, Denver,CO,USA, 2017.

[18] J. Wang, R. Wen, J. Li, F. Yan, B. Zhao, F. Yu, Detecting and mitigating target link-flooding attacks using sdn, IEEE Transactions on Dependable and Secure Computing (1) (2018) 1–1.

[19] W. Xia, Y. Wen, C. Foh, D. Niyato, H. Xie, A survey on software-defined networking, IEEE Communications Surveys Tutorials 17 (1) (2015) 27–51.

[20] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, A clean slate 4d ap-

ACCEPTED MANUSCRIPT

proach to network control and management, in: Proceedings of ACM SIGCOMM, Philadelphia, USA, 2005.
[21] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: taking control of the enterprise, in: Proceedings of ACM SIGCOMM, Kyoto, Japan, 2007.
[22] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Computer Comm. Rev. 38 (2) (2008) 69–74.
[23] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, J. Van Der Merwe, The case for separating routing from routers, in: Proceedings of ACM SIGCOMM FDNA, Portland, USA, 2004.
[24] O. S. Specification, Version 1.0. 0 (wire protocol 0x01), Open Networking Foundation 1.
[25] O. S. Specification, Version 1.3. 0 (wire protocol 0x04)[electronic resource], Open Networking Foundation 1.
[26] O. S. Specification, Version 1.5. 1 (wire protocol 0x04)[electronic resource], Open Networking Foundation 1.
[27] Z. Guo, S. Hui, Y. Xu, H. J. Chao, Dynamic flow scheduling for power-efficient data center networks, in: Proceedings of IEEE/ACM IWQoS, Beijing, China, 2016.
[28] M. Bredel, Z. Bozakov, A. Barczyk, H. Newman, Flow-based load balancing in multipathed layer-2 networks using OpenFlow and multipath-TCP, in: Proceeding of ACM SIGCOMM HotSDN, Chicago, USA, 2014.
[29] Q. Li, Y. Jiang, P. Duan, M. Xu, X. Xiao, Quokka: Latency-aware middlebox scheduling with dynamic resource allocation, Journal of Network and Computer Applications 78 (2017) 253–266.
[30] R. Braga, E. Mota, A. Passito, Lightweight ddos flooding attack detection using nox/openflow, in: Proceedings of IEEE LCN, Colorado, USA, 2010.
[31] S. K. Fayaz, Y. Tobioka, V. Sekar, M. Bailey, Bohatei: Flexible and elastic ddos defense., in: Proceedings of Usenix Security, Washington, D.C., USA, 2015.
[32] L. Dridi, M. F. Zhani, A holistic approach to mitigating dos attacks in sdn networks, International Journal of Network Management 28 (1) (2018) e1996.
[33] L. Zhou, H. Guo, Applying nfv/sdn in mitigating ddos attacks, in: Proceedings of IEEE TENCON, Penang, Malaysia, 2017.
[34] D. Levin, M. Canini, S. Schmid, F. Schaffert, A. Feldmann, et al., Panopticon: Reaping the benefits of incremental sdn deployment in enterprise networks, in: Proceedings of USENIX ATC, Philadelphia, USA, 2014.
[35] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, et al., B4: Experience with a globally-deployed software defined wan, in: Proceedings of ACM SIGCOMM, Hong Kong, China, 2013.
[36] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, R. Wattenhofer, Achieving high utilization with software-driven wan, in: Proceedings of ACM SIGCOMM, Hong Kong, China, 2013.
[37] S. Agarwal, M. Kodialam, T. Lakshman, Traffic engineering in software defined networks, in: Proceedings IEEE INFOCOM, San Francisco, USA, 2013.
[38] M. Caria, T. Das, A. Jukan, M. Hoffmann, Divide and conquer: Partitioning ospf networks with sdn, Tech. Rep. arXiv:1410.5626, TU Braunschweig (2014).
URL `http://arxiv.org/abs/1410.5626`
[39] Y. Guo, Z. Wang, X. Yin, X. Shi, J. Wu, Traffic engineering in hybrid sdn networks with multiple traffic matrices, Computer Networks 126 (2017) 187–199.
[40] R. Krishnan, M. Durrani, P. Phaal, Real-time sdn analytics for ddos mitigation, Open Networking Summit.
[41] J. Caballero, C. Grier, C. Kreibich, V. Paxson, Measuring pay-per-install: The commoditization of malware distribution, in: Proceedings of USENIX Security, San Francisco, CA, 2011.
[42] A. Shaikh, A. Varma, L. Kalampoukas, R. Dube, Routing stability in congested networks: Experimentation and analysis, in: Proceedings of ACM SIGCOMM, Stockholm, Sweden, 2000.
[43] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to algorithms, MIT press, 2009.
[44] C. H. Papadimitriou, K. Steiglitz, Combinatorial optimization: algorithms and complexity, Courier Corporation, 1998.
[45] B. Korte, J. Vygen, Combinatorial optimization: Theory and algorithms, algorithms and combinatorics 2 (2000) (2006).
[46] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische mathematik 1 (1) (1959) 269–271.
[47] E. Danna, S. Mandal, A. Singh, A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering, in: Proceedings of IEEE INFOCOM, Orlando, USA, 2012.
[48] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, N. McKeown, Reproducible network experiments using container-based emulation, in: Proceedings of ACM CoNEXT, Nice, France, 2012, pp. 253–264.
[49] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, Nox: towards an operating system for networks, ACM SIGCOMM Comput. Commun. Rev. 38 (3) (2008) 105–110.

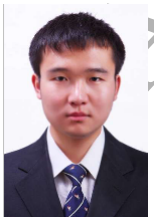
ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Lei Wang received the B.S. degree(2009) from Huazhong University of Science and Technology, Wuhan, China, M.S. degree(2013) from Beijing University of Technology, Beijing, China. He is now a Ph.D. candidate in Graduate at Shenzhen, Tsinghua University, Shenzhen, China. His research interests include Software Defined Networking, Network Function Virtualization and network security.

Qing Li received the B.S. degree (2008) from Dalian University of Technology, Dalian, China, the Ph.D. degree (2013) from Tsinghua University, Beijing, China; all in computer science and technology. He was an assistant researcher in the Graduate School at Shenzhen, Tsinghua University between 2013 2018. He is now an associate professor at the Southern University of Science and Technology, China His research interests include reliable and scalable routing of the Internet, Software Defined Networks and Information Centric Networks.

Yong Jiang received his M.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, P.R. China, in 1998 and 2002, respectively. Since 2002, he has been with the Graduate School of Shenzhen of Tsinghua University, Guangdong, China, where he is currently a professor. His research interests include Internet architecture and its protocols, IP routing technology, etc.



Xuya Jia received his B.S. degree(2014) in Software engineering from Northeast Normal University, Changchun, P.R. China. He is currently working toward his Ph.D. degree with the Department of Computer Science and Technology, Tsinghua University, P.R. China. His current research focuses on Software Defined Networking, fault protection and traffic engineering.

Jianping Wu (Academician of Chinese Academy of Engineering, IEEE fellow): Jianping Wu received the M.S. degree (1982) and Ph.D. degree(1997) in computer science from Tsinghua University, Beijing, China. He is now a Full Professor with the Computer Science Department, Tsinghua University. In the research areas of the network architecture, high performance routing and switching, protocol testing, and formal methods, he has published more than 200 technical papers in academic journals and proceedings of international conferences.

ACCEPTED MANUSCRIPT