Huang XB, Xue KP, Xing YT *et al.* An efficient scheme to defend data-to-control-plane saturation attacks in software-defined networking. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(4): 839–851 July 2022. DOI 10.1007/s11390-022-1495-0

An Efficient Scheme to Defend Data-to-Control-Plane Saturation Attacks in Software-Defined Networking

Xuan-Bo Huang[1] (黄轩博), *Student Member, IEEE*, Kai-Ping Xue[1,*] (薛开平), *Senior Member, CCF, IEEE*
Yi-Tao Xing[1] (幸一滔), *Student Member, IEEE*, Ding-Wen Hu[1] (胡定文), *Student Member, IEEE*
Ruidong Li[2] (李睿栋), *Senior Member, IEEE*, and Qi-Bin Sun[1] (孙启彬), *Fellow, IEEE*

[1] *School of Cyber Science and Technology, University of Science and Technology of China, Hefei 230027, China*
[2] *College of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan*

E-mail: hxb777@mail.ustc.edu.cn; kpxue@ustc.edu.cn; {ytxing, hdw2016}@mail.ustc.edu.cn
lrd@se.kanazawa-u.ac.jp; qibinsun@ustc.edu.cn

Received April 4, 2021; accepted May 24, 2022.

Abstract Software-defined networking (SDN) decouples the data and control planes. However, attackers can lead catastrophic results to the whole network using manipulated flooding packets, called the data-to-control-plane saturation attacks. The existing methods, using centralized mitigation policies and ignoring the buffered attack flows, involve extra network entities and make benign traffic suffer from long network recovery delays. For these purposes, we propose LFSDM, a saturation attack detection and mitigation system, which solves these challenges by leveraging three new techniques: 1) using linear discriminant analysis (LDA) and extracting a novel feature called control channel occupation rate (CCOR) to detect the attacks, 2) adopting the distributed mitigation agents to reduce the number of involved network entities and, 3) cleaning up the buffered attack flows to enable fast recovery. Experiments show that our system can detect the attacks timely and accurately. More importantly, compared with the previous work, we save 81% of the network recovery delay under attacks ranging from 1 000 to 4 000 packets per second (PPS) on average, and 87% of the network recovery delay under higher attack rates with PPS ranging from 5 000 to 30 000.

Keywords software-defined networking (SDN), saturation attack, fast recovery, linear discriminant analysis

1 Introduction

Software-defined networking (SDN) has emerged as a promising network architecture that has enjoyed great popularity in academia and industry in the past few years. By decoupling the control and data planes and providing great programmability, flexible control, and agile management, SDN has seen widespread adoption by many companies [1].

SDN centralizes the control logic by using open southbound interfaces to allow controllers and switches to interact. Among all the implementations of the southbound SDN protocols, OpenFlow is the de facto standard [2]. OpenFlow introduces flows and flow tables to identify and manage network traffic. Switches use the flow rules to process the packets and forward a `PACKET_IN` request to the controller if they receive a table-miss new packet in the flow table. Controller applications install flow rules, which define the core functions to manage the networks.

Unfortunately, the SDN design itself has serious security problems. The data-to-control-plane saturation attack is a new destructive Denial-of-Service attack which overloads the control plane, the data plane, or both [3]. The saturation attacks exploit the table-miss mechanism in SDN by manipulating lots of table-

Regular Paper
Special Section of MASS 2020–2021
A preliminary version of the paper was published in the Proceedings of MASS 2020.
The work was supported in part by the National Natural Science Foundation of China under Grant Nos. 61972371, U19B2023 and U19B2044, and the Youth Innovation Promotion Association of the Chinese Academy of Sciences under Grant No. Y202093.
*Corresponding Author

miss traffic to overload the switches and the controllers. By changing the matching fields, attackers can manipulate table-miss flows easily with traditional DoS floods, like SYN flood [3–5], UDP flood, ICMP flood, and their combinations [6]. Worse still, the SDN control channel is subject to side-channel attacks that reconnoiter the flow rules and the network parameters [7–10]. With the help of these techniques, intelligent saturation attacks can be more destructive and stealthy [11–13].

Therefore, an efficient detection and mitigation mechanism for the saturation attack is in need. Previous studies have proposed several methods.

In terms of detection, the rate and the duration of `PACKET_IN` messages are widely used to detect the attacks [14–16]. In addition, some studies also leverage the flow's entropy and the flow's self-similarity for detection [6, 17–19]. These methods can determine that the whole network is under attack but cannot locate the bot simultaneously, resulting in complex mitigation actions and unnecessary filtration. Besides, the previous study shows that using only the `PACKET_IN` rate is ineffective because burst traffic and abnormal flooding traffic may have similar `PACKET_IN` arrival rates [20].

In terms of mitigation, it is common to use a centralized network entity and migrate attack flows to it for further filtration [6, 14, 17]. However, centralized mitigation methods may have a long migration path and decrease the throughput along the path. FloodDefender [15] detours the attack flows to the victim's neighbor and filters the attack flows in the controller. Nevertheless, the efficiency is related to the number of switches involved in the detour process, and the control channel is still full of suspicious flows. These attack flows injected into the control plane occupy the resources of benign traffic, forcing the legitimate flows to wait long in the queue. Thus, the network needs a long time to recover [17].

From the discussion above, we propose an LDA-based fast recovery saturation attack detection and mitigation system (LFSDM), which detects saturation attacks accurately while making the network recover faster. Our system contains three modules for different designed objectives: an attack detector, a mitigation manager, and distributed mitigation agents. Specifically, first, the attack detector combines the `PACKET_IN` rate and the control channel occupation rate (CCOR) as inputs of LDA to detect the attacks. Second, the mitigation manager collects a white-list in the controller, coordinates corresponding mitigation agents, and installs migration flow rules. Third, the mitigation agents scrub suspicious flows with the white-list and forward benign traffic to the controller.

In the mitigation manager, we propose a novel module called Force_Checking (FC) to enable the fast recovery of networks. The key idea of FC is cleaning up attack flows that are buffered in the control plane, which is always ignored by previous work [15, 17].

We evaluate our system in an SDN simulation environment with Mininet① and RyuController②. The experimental results show that LFSDM can timely and accurately detect the attack while enabling the network to recover in a short time.

The main contributions of our work can be summarized as follows.

1) We put forward a detection scheme that combines the `PACKET_IN` rate and the distribution of the control channel occupation rate (CCOR) and leverages linear discriminant analysis (LDA) to identify saturation attacks. By adopting these methods together, the accuracy of attack detection is improved. Besides, we introduce decentralized mitigation policies based on distributed mitigation agents and white-list, reducing the unnecessary filtration and the number of involved network entities.

2) We make analysis of the network recovery delay under saturation attacks in SDN and point out that the attack flows buffered in the control plane result in a long network recovery delay during the attacks. We propose a novel module Force_Checking (FC), to solve this problem, enabling the network to recover quickly by cleaning up these buffered packets.

3) We implement a prototype system of LFSDM, which contains the detection scheme and the mitigation functions, and conduct extensive experiments in a simulation SDN platform. The results show that LFSDM can detect the attack timely and accurately while saving more than 81% of the ping round trip time (RTT) compared with the traditional methods without cleaning up buffered packets and reducing 87% of the HTTP request time averagely compared with previous work FADM [17].

This paper is an extended version of our conference paper [21]. They differ in the following aspects. 1) In this paper, we utilize the linear discriminant analysis for replacing the original detection method with fixed thresholds in FSDM [21], improving the detection rate.

① http://mininet.org/, April 2020.

② https://osrg.github.io/ryu/, April 2020.

2) We add a noise reduction process in the detection scheme to improve the accuracy. We find that in our conference paper, the entropy of CCOR is easily influenced by high volumes of background traffic. In this paper, we add a noise reduction process, which significantly improves performance when working in high volumes of background traffic. 3) We conduct more realistic experiments with the real-world traffic captured from the gateway of our laboratory, containing about 40 active hosts and several public servers, and compare the detection efficiency of the scheme proposed in this paper with that in our conference paper. The experiments and results are shown in Subsection 7.3.4.

The remainder of this paper is structured as follows. Section 2 elaborates some necessary background knowledge. Section 3 presents some related work. We discuss the adversary model in Section 4, illustrating what kinds of attackers are considered in this paper. Then, we analyze the network recovery delay in Section 5. We elaborate on our LFSDM system in Section 6. Performance evaluation is presented in Section 7. Finally, we draw our conclusions in Section 8.

2 Background

2.1 Software-Defined Networking

In 2008, McKeown *et al.* proposed OpenFlow [2], which is the de facto standard among all the implementations of southbound SDN protocol. According to OpenFlow 1.5③, the controller installs flow rules, distributes test packets, and collects global information of the networks. Each flow rule, according to which switches process network traffic under the instruction of flow rules in a match and action manner, contains a match field and an action field. Generally, controller applications install rules reactively. As shown in Fig.1, a flow will go through several steps in the reactive mode. Each time when the switch receives a flow, it searches the matching rules in the flow tables. If there is a match, the switch will execute the actions in the rule. If no flow rule matches the current incoming flow (called the table-miss flow), the switch encapsulates a PACKET_IN message containing the first flow packet to the controller. After that, the controller applications install new rules for this flow. In detail, different applications make up multiple processing chains waiting for dealing with flow events. When the controller receives a message, the event dispatcher module dispatches it to the subscriber applications. Each application maintains an event queue to store the events that have not been processed temporarily.

2.2 Linear Discriminant Analysis

Linear discriminant analysis (LDA) [22] is a classical statistical approach for dimensionality reduction, whose goal is to find an optimal transformation by minimizing the within-class distance and maximizing the between-class distance simultaneously. In this case, we find that LDA is a simple and efficient method for attack detection.

2.3 Data-to-Control-Plane Saturation Attack

The data-to-control-plane saturation attack is a more destructive DoS attack than the traditional one. It destroys the data plane and the control plane at the same time. In such attacks, the attacker manipulates numerous table-miss flows to trigger `PACKET_IN` floods to occupy the control channel, exhaust the controller's resources, trigger `FLOW_MOD` floods to exhaust the memories of switches, and destroy the data plane target at the same time.

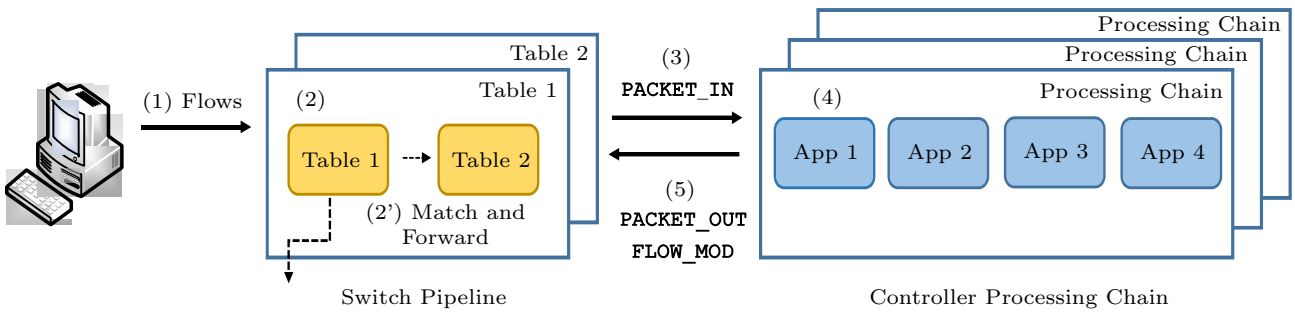


Fig.1. Flow processing example in the reactive mode.

③https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, April 2020.

3 Related Work

The data-to-control-plane saturation attack was firstly introduced by Shin and Gu [23] by triggering flow rules to exhaust the switches' memory and the control plane resources. Then, Shin *et al.* [3] proposed a switch-level SYN proxy to reduce attack flows. Ambrosin *et al.* [24] improved the SYN proxy to enable malicious traffic identification. These methods are not protocol-independent because they can defend only TCP-based attacks. However, saturation attacks may also result from other types of attacks, such as IP spoofing, ICMP flooding, UDP flooding, and so on [6].

Then, protocol-independent methods are proposed to deal with various attack flows. FloodGuard [14] and FADM [17] deploy the centralized middle layer cache between the control plane and the data plane. They migrate the attack flows to the middle layer cache and further filter the benign traffic. However, FloodGuard suffers a high packet loss rate in some cases [15]. FADM has a long network recovery delay under attacks. Unlike the work mentioned above, FloodDefender [15] installs a flow rule to detour attack flows to the victim's neighbor switches. However, the detour process influences lots of switches in the network. In the meantime, the control channels are not protected well because FloodDefender filters the network flows in the controller.

From our perspective, deploying distributed mitigation agents is a more sound approach. The main idea of the mitigation agent is to deal with attack flows before they are forwarded to the control planes. We deploy a distributed mitigation agent in each edge gateway as a virtual functionality module to reduce the number of the influenced network switches, compared with the centralized filtration center.

4 Adversary Model

An adversary model is a description of an attacker in a computer or networked system, which is an integral concept in cyber defense. In this paper, we consider a more powerful adversary than that in the traditional SDN data-to-control-plane saturation attacks [23], where an attacker can reconnoiter some matching fields of flow rules in SDN and then leverage the information to launch more intelligent attacks [7, 25]. We also assume that an attacker can compromise bots with flexible locations, even inside an SDN-based cloud network. These assumptions make the detection and mitigation of the saturation attack a challenging problem. Meanwhile, we assume that the controllers and the switches are secure entities in the network [3], which attackers cannot compromise; thus, defenders deploy security applications in these places.

5 Network Recovery Delay

In this section, we analyze the network recovery delay under attacks. First, we show why and where the network delay comes under attack. The key reason is that the control plane contains many applications to process flows (e.g., ARP Proxy, ACL, Load Balancer), and these applications cannot distinguish attack flows. An event dispatcher receives the packets from the switch and enqueues each packet to the corresponding subscribers' buffer. When a saturation attack happens, the event dispatcher puts lots of attack flows into the event buffer, but the applications still fetch events as usual. Traditional mitigation policies [14, 15, 17] generally install a flow rule to block or migrate the attack flows in the switches while ignoring the buffered attack flows in the controller. Though the security application timely blocks the attack, other applications need a long time to clean up the attack flows. Thus, benign traffic cannot be timely forwarded until the attack flows dequeue. The buffered attack flows result in a long network recovery delay.

Then, we estimate how long the network needs to recover from an attack, even though the attack is timely blocked. We assume that the attacker launches an attack at time t_{a} with the rate of K_{pps} and an average size of b_{a} bytes, and the detector needs a period t_{w} (i.e., a detection cycle) to collect information and determine the attack. Additionally, B_{cs} denotes the control channel bandwidth.

Fig.2 shows an example of the network recovery delay under the worst attack case. We call it "the worst" because the attack happens slightly before a detection cycle starts, meaning that the defenders need more than one detection cycle (the longest detection time), to detect the attack. We depict the longest detection period as t_{d}. During t_{d}, there are about $t_{\mathrm{d}} \times K_{\mathrm{pps}}$ attack packets in the control plane. The number of attack packets N_{a} is:

$$N_{\mathrm{a}} = \min\left\{ t_{\mathrm{d}} \times \frac{B_{\mathrm{cs}}}{b_{\mathrm{a}}},\ t_{\mathrm{d}} \times K_{\mathrm{pps}} \right\}.$$

When the number of attack packets exceeds the bandwidth, there are $t_{\mathrm{d}} \times \frac{B_{\mathrm{cs}}}{b_{\mathrm{a}}}$ attack packets as other packets will overflow the switch buffer, thus being dropped.

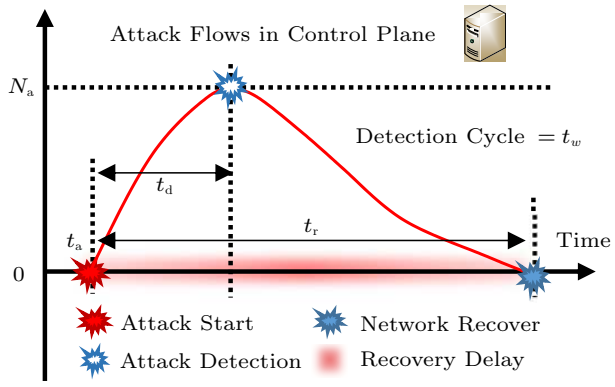


Fig.2. Example of the network recovery delay without cleaning up remaining attack flows under the worst attack case, where the defenders need the longest time to discover the attack.

When the system detects an attack, traditional methods install a defensive flow rule immediately. However, it is not enough. Because there are N_a buffered packets in the control channel, the defensive rules need time for queuing. Once the controller installs the defensive rules, the switches stop injecting attack flows to the control plane. Notice that though these rules block attack flows in the data plane, there are still N_a attack flows buffered in the control plane. Then, when a normal flow comes, it still has to wait for a long time. We assume that applications need time t_c to deal with each flow. Then the number of remaining packets N_r can be calculated as:

$$N_r = N_a - \frac{t_d}{t_c}.$$

Then the time for the controller to process all these attack flows is denoted as

$$t_m = t_c \times N_r.$$

We define the whole network recovery delay as t_r, and we have

$$\begin{aligned} t_r &= t_d + t_m = t_d + t_c \times N_r \\ &= t_d \times \left(1 + t_c \times \max\left\{\frac{B_{cs}}{b_a}, K_{pps}\right\}\right). \end{aligned} \quad (1)$$

From (1), the number of buffered packets N_r and the detection time t_d are the core parameters of the network recovery delay. To reduce the network recovery delay, intuitively, we should decrease t_d or N_r. However, reducing t_d is not generic as it is relevant to the detection algorithm, influencing the decision precision. Thus, we cannot reduce t_d. Instead, reducing N_r is more generic. For this purpose, we propose a novel module FC to reduce N_r effectively.

6 LFSDM System

In this section, we introduce our detection and mitigation system: LFSDM. As Fig.3 shows, there are three

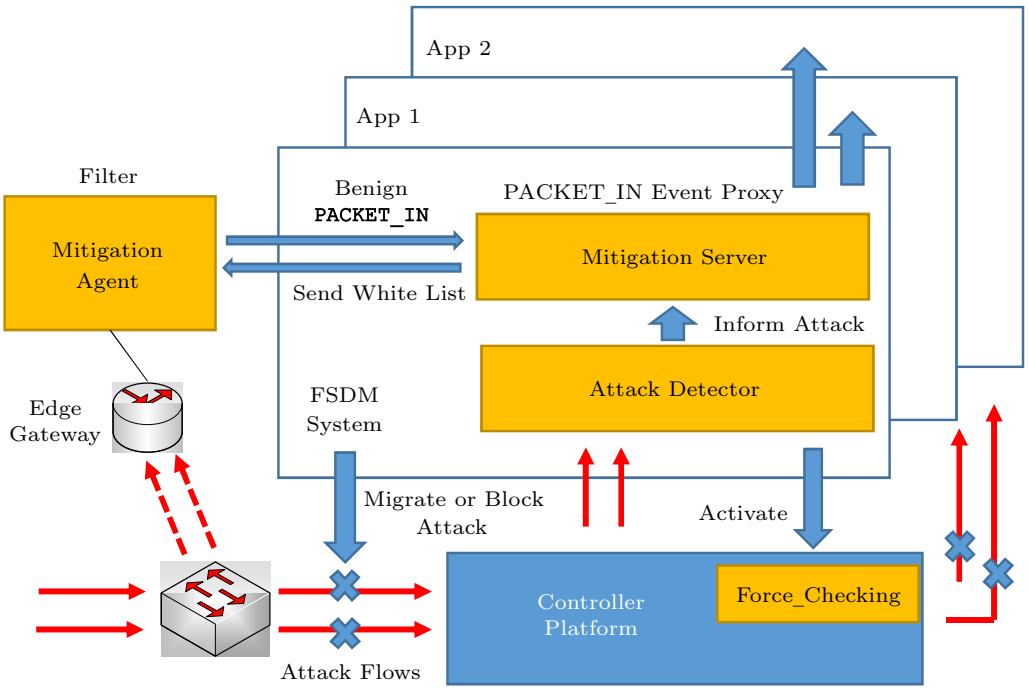


Fig.3. System overview and working process of LFSDM.

modules. In detail, the attack detector collects the CCOR distribution and uses the trained LDA model to measure whether the network is under attack. Moreover, the mitigation server collects a white-list and sends it to each mitigation agent. Once the system detects an attack, the mitigation server stops the white-list collection and blocks or migrates suspicious flows depending on whether the attack flows are mixed with benign flows. If so, we forward suspicious flows to the nearest mitigation agent. Then, the mitigation agent scrubs the attack traffic based on the white-list and forwards the benign traffic to the mitigation server using TCP. Finally, the mitigation server encapsulates the benign packets to a `PACKET_IN` event and sends it to the event dispatcher.

6.1 Attack Detector

6.1.1 Attack Detection Overview

We combine the `PACKET_IN` rate and the control channel occupation rate (CCOR) distribution for attack detection. The `PACKET_IN` rate is a good but not perfect parameter in the detection scheme as it might regard burst traffic as attacks, which increases the false-positive rate. Using the CCOR distribution combined with `PACKET_IN` rate, we decrease the false-positive rate while maintaining a high true-positive rate. The intuition of using CCOR is that a benign network flow from a specific interface is generally in a pair; thus, we can observe its pair flow from another interface. It makes the CCOR distribution look like a uniform distribution. However, when numerous attack flows come, their pair flows can hardly be observed as they exceed the capability the system can afford, and the benign flows from other interfaces will also be congested and queued. That is to say, the monitored CCOR of the specific victim switch interface will be high, which results in a meager entropy value of the whole CCOR distribution. Thus, we use the total `PACKET_IN` rate (depicted as Sum) and the entropy of CCOR distribution (depicted as H) as the input and leverage the machine learning method LDA [22] to solve a binary classification problem.

6.1.2 Setup

The SDN controller offers the link discovery module to find the active switches and their interfaces. We use tuple (`dpid`, `in_port`) to identify each device and its interfaces. Though a flow can trigger multiple `PACKET_IN` messages from different switches, we only monitor the first one as it is enough for attack detection. Considering there are n such interfaces existing in the data plane, we establish a monitor vector as: $\boldsymbol{M}_t = (h_1, h_2, \cdots, h_n)$, where t represents the time window. We set the initial value of $\boldsymbol{M}_0 = (1, 1, \cdots, 1)$ for the sake of entropy calculation.

Assuming in a time window, there come k `PACKET_IN` messages from host h_i. Then we set $h_i = k$. In each time window, we calculate $Sum_t = \sum_{i=1}^{n} h_i$. And the CCOR in this time window can be calculated as:

$$\boldsymbol{M}_t^* = \frac{\boldsymbol{M}_t}{Sum_t}.$$

The entropy of CCOR can be calculated as:

$$H(\boldsymbol{M}_t^*) = -\sum_{i=1}^{n} h_i^* \times \log_2(h_i^*). \tag{2}$$

However, such a simple calculation cannot reach a good classification result. That is because the background traffic could become the noise in the entropy calculation and influence the distribution of CCOR, especially in a relatively high throughput environment. To reduce the noise, also to keep the extracted attack features, we update H in several steps:

1) find the maximum value in $\boldsymbol{M}_t$, denoted as max;

2) pop it out and calculate the average of others, denoted as $mean$;

3) update the value of each h_i from $h_i = h_i - mean$;

4) push the maximum value back, and use (2) to update the entropy $H(\boldsymbol{M}_t^*)$;

5) map H with an exponential function, which is

$$H^*(\boldsymbol{M}_t^*) = \sqrt{10^3}^{H(\boldsymbol{M}_t^*)},$$

because of the range of function $S \gg H$.

Now, the $(H^*(\boldsymbol{M}_t^*), Sum_t)$ pair can be used as input in the LDA model. The idea for these steps is that steps 2 and 3 reduce the noise of the CCOR from other interfaces, and steps 1 and 4 promise that the most likely victim interface will hold the attack features. Also, these four steps make the whole CCOR distribution much more uneven under attacks, resulting in easier attack detection.

To sum up, our detection scheme uses tuple (Sum, H^*) in each time window for detection. To illustrate our idea, there are two easy examples shown in Fig.4. In Fig.4(a), attackers use large-scale botnets outside the network to launch attacks, which can be detected with a low H^* value. While, in Fig.4(b), the

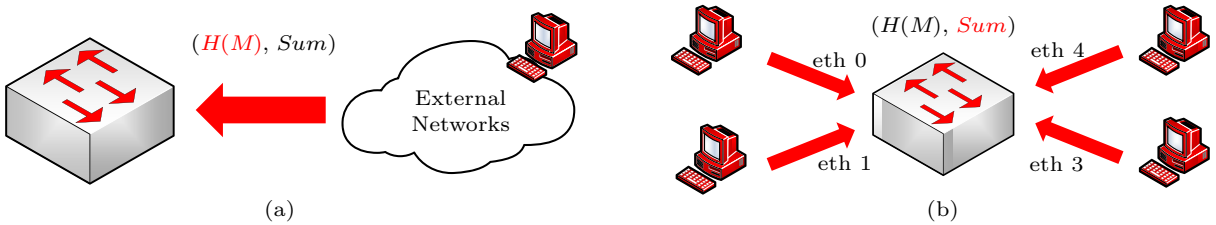


Fig.4. Two attack examples. (a) Attacks with large-scale botnets outside the network. (b) Attacks with several internal hosts. eth represents the "Ethernet Interface" of the device.

attacker launches attacks from multiple network interfaces. In that way, H^* may not be low enough, but Sum can promise that the attack is detected precisely.

6.1.3 Training LDA Model

LDA is a supervised machine learning algorithm that must be trained before use. The training data is made by ourselves, containing realistic benign traffic captured from the gateway of our laboratory and different types and scales of saturation attacks launched by us. The benign traffic has an average rate of 10 Mbps with about 40 active hosts. We use the Tcprewrite④ tool to remap the realistic address to the simulation environment address while maintaining relationships between client and server. Then, we use the Tcpreplay tool to replay the packets on each VM. We collect about 191 normal samples and about 192 attack samples. The LDA training result is shown in Fig.5. We can see that the malicious traffic always comes with a huge Sum value and a meager H^* value.

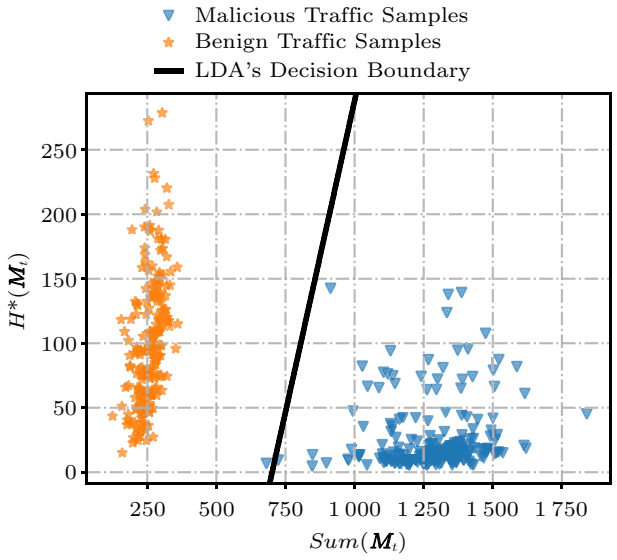


Fig.5. LDA predict result with training data.

6.1.4 Bot Detection

From the CCOR distribution, we find a suspicious interface connected to the bots. In t time windows, if the network state is normal, the attack detector collects a series of monitor vectors as $(\boldsymbol{M}_1, \boldsymbol{M}_2, \cdots, \boldsymbol{M}_t)$. When the attack detector determines that there is an attack during the atk time cycle (denoted as $\boldsymbol{M}_{atk}$), firstly, it checks if there exists an interface that has the malicious CCOR (e.g., larger than 50%). If so, it returns the interface index as the attacker's interface. If not, it then calculates $\boldsymbol{M}_{atk} - \boldsymbol{M}_{atk-1}$ to find the suspicious interfaces that have the highest growth rate of the number of `PACKET_IN` messages. This step can find only one malicious interface in one iteration. If there are multiple attackers, we need several iterations to find them one by one.

6.2 Mitigation Manager

The mitigation manager has a server in the controller and distributed agents deployed in edge gateways. The server and agents synchronize the white-list through TCP. As shown in Fig. 3, when the network is in the normal state, the mitigation server collects a white-list for benign traffic periodically and sends it to each mitigation agent. We adopt the collecting policy for the white-list that is discussed in FADM [17]. The mitigation server updates the white-list in each benign time window according to the following rules.

- The flow is in a pair.
- In a period, the number of flows with the same source IP exceeds a threshold value.

We add the source IP addresses that match these requirements to the white-list in each time window. However, each mitigation agent needs to synchronize the white-list with the mitigation server in our system, which may continuously bring a high cost of bandwidth and memory. Thus, we use a Bloom filter [26] to store the white-list for each mitigation agent and the miti-

④https://tcpreplay.appneta.com/, April 2021.

gation server. Notice that even though the Bloom filter misjudges false-negative samples, it would not influence the system as most attack flows are scrubbed. For example, for a Bloom filter with a false rate of 0.1%, the mitigation agents may forward 10 attack flows in each 10 000 attack flows. However, few attack flows can hardly cause an evident impact on the whole system. Thus, we save a large amount of bandwidth and memory. Once the attack detector determines an attack takes place, the mitigation server processes the attack flows in three steps.

1) The mitigation server installs a defensive rule on the victim switch. The mitigation server installs a block rule for attacks from the single-host interface. For attacks from the multiple-hosts interface (e.g., the edge gateway's interface), as the traffic is hybrid with benign traffic, the mitigation server installs a migration rule for these flows to the nearest mitigation agent for further filtration.

2) If migration happens, mitigation server activates the corresponding mitigation agent, which uses pypcap[5] to capture all the packets delivered by this agent. When activated, the agent extracts all the packet headers and checks if the source IP address is in the white-list. If so, the agent sends the packets to the mitigation server.

3) The mitigation server acts as an event dispatcher under attacks. When receiving a packet from the mitigation agent, the mitigation server uses the victim's (`dpid`, `in_port`) value and encapsulates the packet as a `PACKET_IN` event. The whole protection process is transparent to other network functions with this step. Then, the forwarding application installs the flow rule and avoids table-miss again.

These methods mentioned above ensure timely protection of the control plane. However, we find that the network still needs a long time to process the buffered attack flows (Section 5). We propose the FC Module to solve this problem, which will be discussed in the following part.

6.3 Force_Checking Module

From the discussion in Section 5, ignoring the influence of the buffered attack flows results in a prolonged network recovery delay. We propose the FC module to solve this problem, which achieves this goal in two steps. First, we use a white-list to distinguish benign traffic. This step could be changed to other traffic classification methods. Second, we check each packet in the controller event queue before distributing it to each application. The FC module does nothing when the network state is normal; thus, it will not bring overhead to our system and the network. When the system detects an attack, the FC module checks each packet header in the event queue of the controller. If the event is not in the Bloom filter, the FC module drops it to avoid adding it to the subscribers' event queues. In this way, the buffered attack flows can be efficiently reduced, thus enabling the system to release resources for benign traffic.

7 Evaluation

7.1 Implementation

We implement a prototype of LFSDM system in the simulation SDN platform, with an i5-9400 CPU and 6 GB memory Ubuntu 16.04 virtual machine. Our environment uses the Ryu, the Mininet, and the OpenVswitch[6] to create the controller, the hosts, and the OpenFlow switches respectively. Fig.6 shows our topology with three switches, two legacy routers as gateways, a web-server, and several hosts in the network. In detail, the test hosts send ping messages and record RTT or download a certain page periodically from the web-server and log the request time. Moreover, all the hosts replay the real-world background traffic. We create the attack samples manually with different protocol types and random source addresses. In the Ryu controller, we implement the other two applications. They are ArpHub and EasyACL; both of them are written by ourselves and tested to be functionally well. The former responds to all the ARP requests, and the latter provides essential access control for internal hosts based on IP addresses. These two applications work like a layer-3 forwarding function with access control. The network's control logic is: in `table_0`, the traffic having access in ACL will be set `goto=table_2`, and the table-miss flows will go to `table_1`. In `table_1`, the table-miss ARP, TCP, and UDP messages are forwarded to the controller. In `table_2`, the layer-2 forwarding rules forward the traffic to a certain interface.

As mentioned above, we assume that the attacker has the ability to reconnoiter the flow rules in our system, which means the attacker knows that all the traffic

[5] https://pypi.org/project/pypcap/, April 2021.
[6] https://www.openvswitch.org/, April 2020.

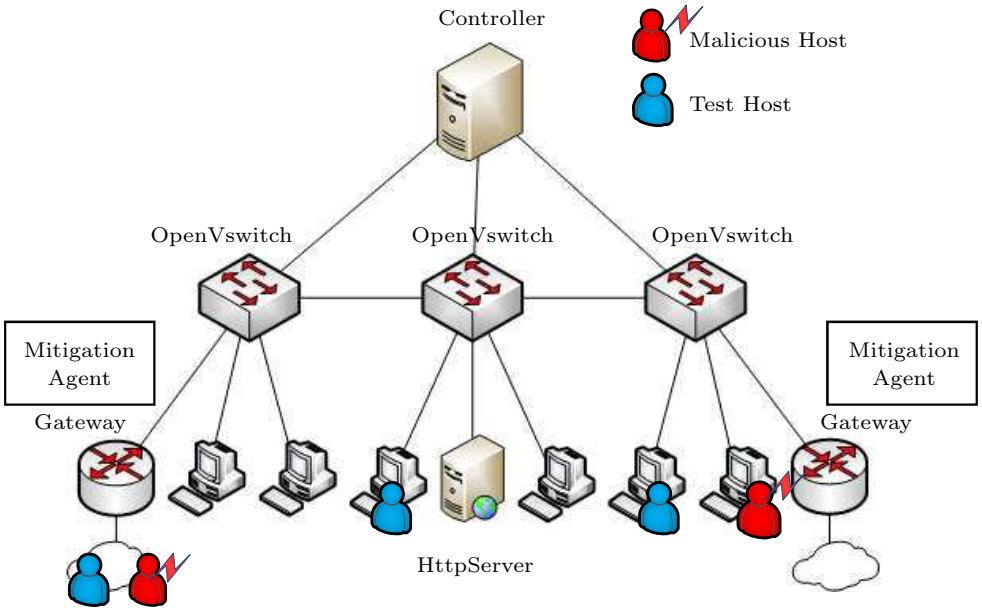


Fig.6. Experimental topology with the prototype system of LFSDM.

with random values of `ip_src` and `ip_dst`=web-server will not be dropped and can trigger `PACKET_IN` floods. The malicious hosts may be outside or inside the network and use the hping3[⑦] to create attack flows with different fields and protocols.

7.2 Experiments

We conduct four experiments to evaluate our system. First, we measure the buffered attack flows to verify the correctness of the network recovery model in practice. The maximum buffered packets show the worst case in the longest detection time, while the average buffered packets reflect generic situations in the random detection time. The ArpHub and the EasyACL count the buffered attack flows with the attack rate ranging from 500 to 2 500 PPS.

Second, we evaluate the network recovery delay. We define the network recovery delay as the first round trip time (RTT) of a new flow under an attack. RTT contains a queue delay in the controller, which indicates how long the controller needs to empty the attack flows. To measure that, test hosts send ping messages under attacks. To get a more convincing value, we send the ping messages slightly after the attacks, making them mixed with the attack flows. The ping messages and the attack flows both trigger `PACKET_IN` messages, and they are mixed in the same queue waiting for processing. We conduct experiments with traditional methods and the FC function, with the attack rate increasing from 500 to 4 000 PPS.

Third, we make a comparison between our work and the FADM [17] system. FADM defines the network recovery delay as the first HTTP request time under an attack. In FADM, hosts download pages from the web server every three seconds, and they use the first record time under attacks to measure the network recovery delay. The FADM definition of network recovery delay and ours reflect the period that the network needs to recover. In this experiment, we use the definition of FADM for evaluation.

Fourth, we evaluate the detection efficiency with attack rates PPS ranging from 1 000 to 5 000, compared with our conference paper FSDM [21].

7.3 Evaluation

7.3.1 *Buffered Attack Packets*

This subsection evaluates the influence of buffered attack packets on the network recovery delay. Fig. 7 shows the number of buffered packets of each application in two methods, with and without the module FC. Besides, we measure the average queue delay of each packet for each application in Table 1. Benign traffic has a very long delay under attacks. For example, under 2 500 PPS attacks, more than 5 628 packets are buffered in the EasyACL and 5 380 packets in the ArpHub. On average, the queue delay for each packet

⑦http://www.hping.org/, April 2021.

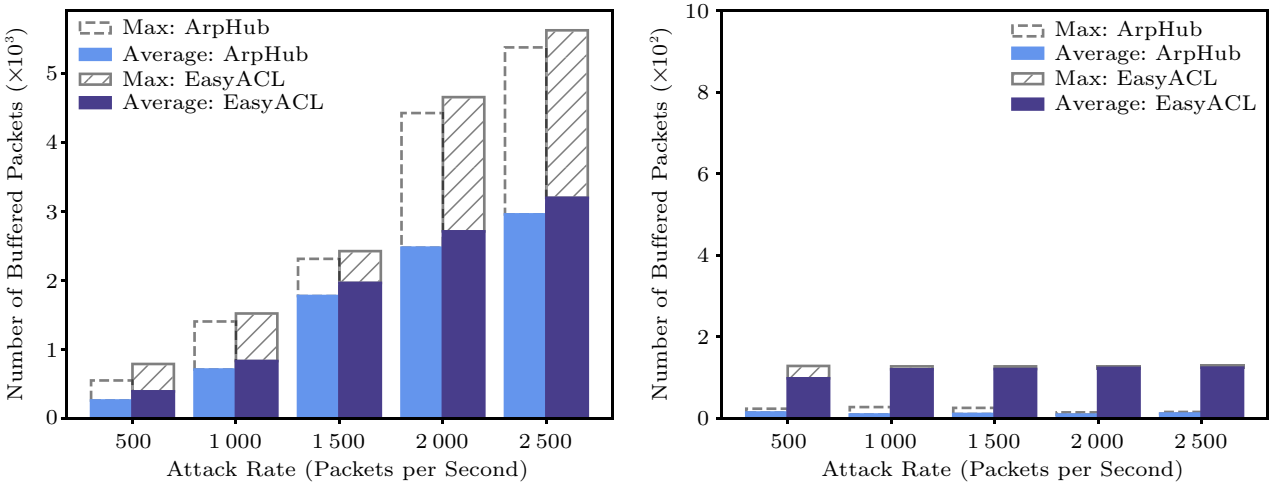


Fig.7. Number of buffered packets under attacks. (a) Buffered attack packets without the FC module. (b) Buffered attack packets with the FC module.

is 30.03 ms in ArpHub and 33.23 ms in EasyACL. Under rough evaluation, the benign traffic queue delays for these two applications are about 162 s and 187 s, respectively. Worse still, these buffered packets will trigger lots of `PACKET_OUT` and `FLOW_MOD` messages, exhausting the limited flow entries of commercial OpenFlow switches. This experiment indicates the importance of emptying the buffered packets.

Table 1. Average Queue Time (ms) for Each Application

Situation	ArpHub	EasyACL
No attack	4.13	7.26
Under attacks	30.03	33.23

The FC module decreases the buffered packets a lot, as shown in Fig.1. With the attack rate increasing, the number of buffered packets does not increase and remains at a low value. Even in the worst case with an attack rate of 2 500 PPS, there are only 128 buffered packets in EasyACL and only 20 in ArpHub. Averagely, FC reduces 98.43% of buffered packets for ArpHub and 89.16% packets for EasyACL.

7.3.2 *Network Recovery Delay*

Then, we compare the network recovery delay of two approaches, with and without the FC module. In this subsection, the network recovery delay is defined as the ping RTT. Fig.8 shows the average ping RTT under different attack rates. RTT suffers an immediate increase without the FC module. The network needs even more than 40 seconds to recover in worse cases. However, with the FC module, the network can empty buffered attack flows to release computational and memory resources, thus responding and dealing with benign flows in a short time. Even under attacks ranging from 3 000 PPS to 4 000 PPS, the network recovery delay is no more than eight seconds. Compared with the traditional mitigation policies that ignore the remaining traffic, we successfully reduce more than 81% of the network recovery delay.

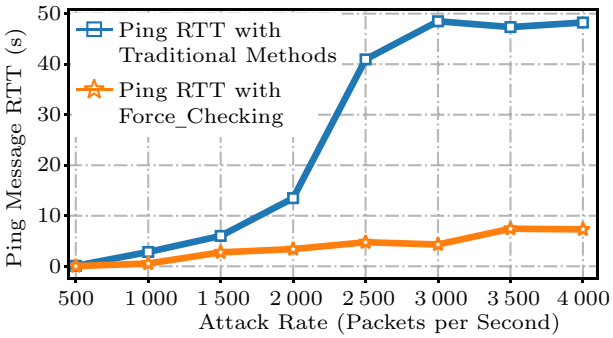


Fig.8. Network recovery delay with ping RTT.

7.3.3 *Comparison with FADM*

In this part, we compare our work with the FADM system under higher attack rates. We use attack range and the definition of network recovery delay in FADM to keep consistency. Fig. 9 shows the HTTP request time under attack. When the attack rate exceeds 5 000 PPS, the FADM system needs more than 30 seconds to recover from those large attack flows. While, in our system, the HTTP request time is less than 6 seconds. Averagely, we reduce 87.17% of the HTTP requests time under many attacks, indicating that LFSDM is robust and practical.

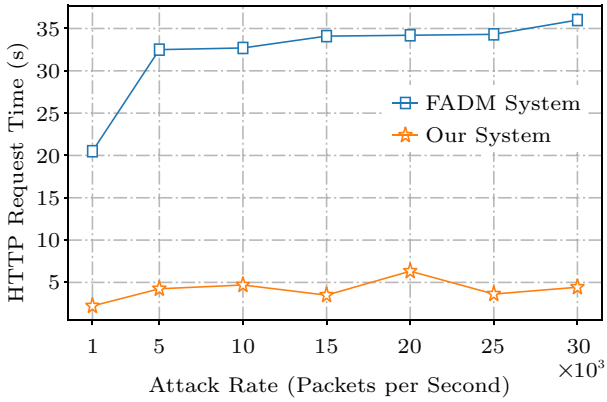


Fig.9. Network recovery delay compared with FADM.

7.3.4 *Detection Efficiency*

In this subsection we compare our detection efficiency with that of our conference paper [21]. We use detection rate (DR) to measure the performance, which is defined as:

$$DR = \frac{TP}{TP + FN},$$

where TP represents the number of the sample that attacks are detected in time (no more than two-time windows). FN denotes the number of attacks that are regarded as normal. We conduct two different experiments under background traffic of 1 Mbps and 10 Mbps and evaluate our work with our conference edition [21] under attack rates of 1 000 to 5 000 PPS.

From Fig.10, we see that the improved method, using LDA and noise reduction, explicitly improves the detection rate compared with our previous work using the fixed threshold. LDA learns from the traffic to find an optimal boundary, while noise reduction decreases the influence of background traffic. In detail, Fig.10(a) shows that with the background traffic of 1 Mbps, the fixed threshold method achieves an average detection rate of 84% while LDA detects all the attacks. Besides, with the background traffic of 10 Mbps, as the previous method does not use noise reduction, the detection rate only exceeds 50%. The high background noise makes the CCOR distribution look uniform and the entropy calculation inaccurate. In this paper, the noise reduction process solves this problem, keeping a high detection rate under high background traffic.

8 Conclusions

SDN-based saturation attack defenses have been extensively studied in the past few years. However, the long network recovery delay is a challenging problem that harms previous work's feasibility. This paper's main contribution is that we performed an in-depth analysis and pointed out that the remaining attack flows and the event dispatching mechanism in SDN are the culprit of the long network recovery delay. To reduce it, we proposed a novel module FC, which enables the event dispatching module to clean up the buffered attack flows and release controller resources, thus allowing the network to recover in a short time. The experimental results showed that LFSDM can accurately detect the attack while reducing 81%–87% of the network recovery delay compared with the previous work. FC also works well in other mainstream controllers and does not harm the throughput when there are no attacks, and thus it is efficient and practical. We also increased the attack detection rate by leveraging the LDA and the CCOR distribution. We will further study the

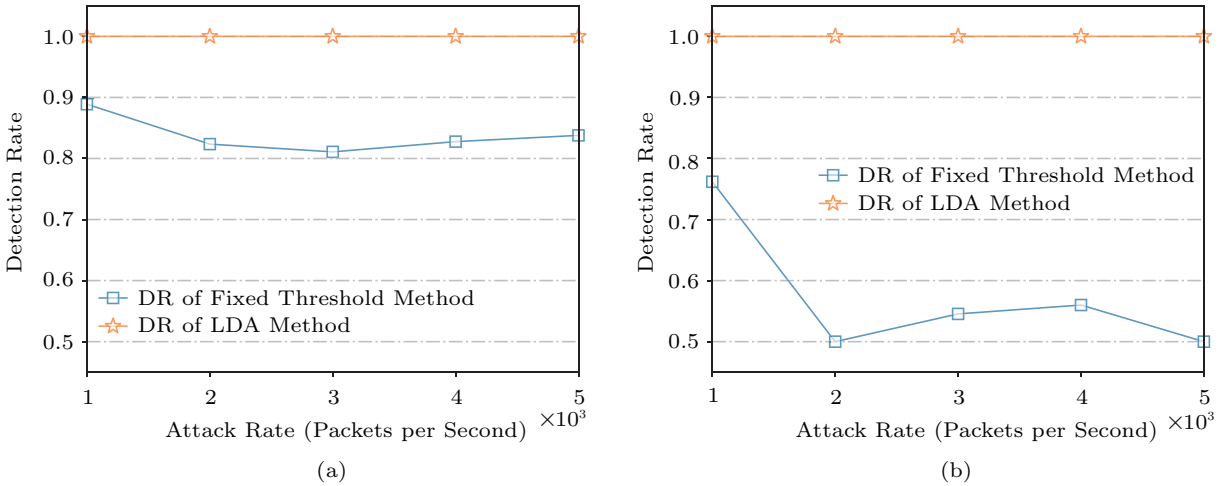


Fig.10. Detection rate under different background traffic. (a) DR under 1 Mbps traffic. (b) DR under 10 Mbps traffic.

quality of service problems of benign traffic under DDoS attacks, such as the network recovery delay, the strong end-to-end communication availability, and so on.

References

[1] Kreutz D, Ramos F, V P *et al.* Software-defined networking: A comprehensive survey. *Proc. IEEE*, 2015, 103(1): 14-76. DOI: 10.1109/JPROC.2014.2371999.

[2] McKeown N, Anderson T, Balakrishnan H *et al.* OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 2008, 38(2): 69-74. DOI: 10.1145/1355734.1355746。

[3] Shin S, Yegneswaran V, Porras P, Gu G. AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks. In *Proc. the ACM Conference on Computer and Communications Security*, November 2013, pp.413-424. DOI: 10.1145/2508859.2516684.

[4] Fichera S, Galluccio L, Grancagnolo S *et al.* OPERETTA: An OPEnflow-based remedy to mitigate TCP SYNFLOOD attacks against web servers. *Computer Networks*, 2015, 92: 89-100. DOI: 10.1016/j.comnet.2015.08.038.

[5] Mohammadi R, Javidan R, Conti M. SLICOTS: An SDN-based lightweight countermeasure for TCP SYN flooding attacks. *IEEE Transactions on Network and Service Management*, 2017, 14(2): 487-497. DOI: 10.1109/TNSM.2017.2701549.

[6] Li Z, Xing W, Khamaiseh S, Xu D. Detecting saturation attacks based on self-similarity of OpenFlow traffic. *IEEE Transactions on Network and Service Management*, 2020, 17(1): 607-621. DOI: 10.1109/TNSM.2019.2959268.

[7] Achleitner S, Porta T L, Jaeger T, McDaniel P. Adversarial network forensics in software defined networking. In *Proc. the Symposium on SDN Research*, April 2017, pp.8-20. DOI: 10.1145/3050220.3050223.

[8] Liu S, Reiter M K, Sekar V. Flow reconnaissance via timing attacks on SDN switches. In *Proc. the 37th IEEE International Conference on Distributed Computing Systems*, June 2017, pp.196-206. DOI: 10.1109/ICDCS.2017.281.

[9] Cao J, Li Q, Xie R *et al.* The CrossPath attack: Disrupting the SDN control channel via shared links. In *Proc. the 28th USENIX Security Symposium*, August 2019, pp.19-36.

[10] Patwardhan A, Jayarama D, Limaye N *et al.* SDN Security: Information disclosure and flow table overflow attacks. In *Proc. the IEEE Global Communications Conference*, December 2019. DOI: 10.1109/GLOBECOM38437.2019.9014048.

[11] Zhang M, Li G, Xu L *et al.* Control plane reflection attacks in SDNs: New attacks and countermeasures. In *Proc. the 21st International Symposium on Research in Attacks, Intrusions, and Defenses*, September 2018, pp.161-183. DOI: 10.1007/978-3-030-00470-5_8.

[12] Cao J, Xu M, Li Q *et al.* Disrupting SDN via the data plane: A low-rate flow table overflow attack. In *Proc. the 13th International Conference on Security and Privacy in Communication Networks*, October 2017, pp.356-376. DOI: 10.1007/978-3-319-78813-5_18.

[13] Yu M, He T, McDaniel P, Burke Q K. Flow table security in SDN: Adversarial reconnaissance and intelligent attacks. In *Proc. the 39th IEEE International Conference on Computer Communications*, July 2020, pp.1519-1528. DOI: 10.1109/INFOCOM41043.2020.9155538.

[14] Wang H, Xu L, Gu G. FloodGuard: A DoS attack prevention extension in software-defined networks. In *Proc. the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015, pp.239-250. DOI: 10.1109/DSN.2015.27.

[15] Gao S, Peng Z, Xiao B, Hu A Q, Ren K. FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks. In *Proc. the 36th IEEE International Conference on Computer Communications*, May 2017. DOI: 10.1109/INFOCOM.2017.8057009.

[16] Buragohain C, Medhi N. FlowTrApp: An SDN based architecture for DDoS attack detection and mitigation in data centers. In *Proc. the 3rd International Conference on Signal Processing and Integrated Networks*, February 2016, pp.519-524. DOI: 10.1109/SPIN.2016.7566750.

[17] Hu D, Hong P, Chen Y. FADM: DDoS flooding attack detection and mitigation system in software-defined networking. In *Proc. the IEEE Global Communications Conference*, December 2017. DOI: 10.1109/GLOCOM.2017.8254023.

[18] Giotis K, Argyropoulos C, Androulidakis G *et al.* Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. *Computer Networks*, 2014, 62: 122-136. DOI: 10.1016/j.bjp.2013.10.014.

[19] Da Silva A, Wickboldt J, Granville L, Schaeffer-Filho A. ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN. In *Proc. the IEEE/IFIP Conference on Network Operations and Management Symposium*, April 2016, pp.27-35. DOI: 10.1109/NOMS.2016.7502793.

[20] Kotani D, Okabe Y. A packet-in message filtering mechanism for protection of control plane in OpenFlow networks. In *Proc. the ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, October 2014, pp.29-40. DOI: 10.1145/2658260.2658276.

[21] Huang X, Xue K, Xing Y, Hu D, Li R, Sun Q. FSDM: Fast recovery saturation attack detection and mitigation framework in SDN. In *Proc. the 17th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, Dec. 2020, pp.329-337. DOI: 10.1109/MASS50613.2020.00048.

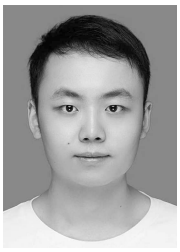
[22] Fisher R A. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 1936, 7(2): 179-188. DOI: 10.1111/j.1469-1809.1936.tb02137.x.

[23] Shin S, Gu G. Attacking software-defined networks: A first feasibility study. In *Proc. the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, August 2013, pp.165-166. DOI: 10.1145/2491185.2491220.

[24] Ambrosin M, Conti M, De Gaspari F, Poovendran R. LineSwitch: Tackling control plane saturation attacks in software-defined networking. *IEEE/ACM Transactions on Networking*, 2017, 25(2): 1206-1219. DOI: 10.1109/TNET.2016.2626287.

[25] Sonchack J, Dubey A, Aviv A J *et al.* Timing-based reconnaissance and defense in software-defined networks. In *Proc. the 32nd Annual Conference on Computer Security Applications*, December 2016, pp.89-100. DOI: 10.1145/2991079.2991081.

[26] Bloom B H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970, 13(7): 422-426. DOI: 10.1145/362686.362692.

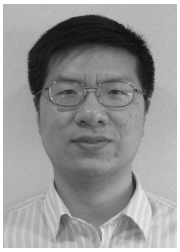


Xuan-Bo Huang received his B.S. degree in information security from the School of Cyber Science and Technology, University of Science and Technology of China (USTC), Hefei, in 2020. He is currently working toward his M.S. degree in information security in USTC, Hefei. His research interests include security issues in software-defined networking and network function virtualization.

Kai-Ping Xue received his Bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), Hefei, in 2003, and received his Ph.D. degree in information and communication engineering from the Department of Electronic Engineering and Information Science (EEIS), USTC, Hefei, in 2007. From May 2012 to May 2013, he was a postdoctoral researcher with Department of Electrical and Computer Engineering, University of Florida, Gainesville. Currently, he is a professor in the School of Cyber Science and Technology and the Department of EEIS, USTC, Hefei. His research interests include next-generation Internet, distributed networks and network security. He serves on the Editorial Board of several journals, including the IEEE Transactions on Dependable and Secure Computing (TDSC), the IEEE Transactions on Wireless Communications (TWC), and the IEEE Transactions on Network and Service Management (TNSM). He has also served as a (lead) guest editor of many reputed journals/magazines, including IEEE Journal on Selected Areas in Communications (JSAC), IEEE Communications Magazine and IEEE Network. He is a fellow of the IET and a senior member of CCF and IEEE.

Yi-Tao Xing received his B.S. degree in information security from School of the Gifted Young, University of Science and Technology of China (USTC), Hefei, in 2018. He is currently working toward his Ph.D. degree in information security in USTC, Hefei. His research interests include future Internet architecture and transmission optimization.

Ding-Wen Hu received his B.S. degree in computer science from School of the Computer Science, The PLA Information Engineering University, Zhengzhou, in 2007. He is currently working toward his Ph.D. degree in communication and information systems in USTC, Hefei. His research interests include security issues in networks.



Ruidong Li received his Bachelor's degree in engineering from Zhejiang University, Hangzhou, in 2001, and received a doctorate of engineering from the University of Tsukuba, Tsukuba, in 2008. He is an associate professor in College of Science and Engineering, Kanazawa University, Kanazawa. Before joining Kanazawa University, he was a senior researcher with the Network System Research Institute, National Institute of Information and Communications Technology (NICT), Koganei City, Tokyo. He is the founder and chair for the IEEE SIG on big data intelligent networking and IEEE SIG on intelligent Internet edge, and the secretary of IEEE Internet Technical Committee. He also serves as the chairs for conferences and workshops, such as IWQoS 2021, MSN 2020, BRAINS 2020, ICC 2021 NMIC Symposium, ICCN 2019/2020, NMIC 2019/2020 and organized the special issues for the leading magazines and journals, such as IEEE Communications Magazine, IEEE Network, and IEEE Transactions on Network Science and Engineering (TNSE). His current research interests include future networks, big data networking, blockchain, information-centric network, Internet of Things, network security, wireless networks, and quantum Internet. He is a senior member of IEEE and a member of IEICE.

Qi-Bin Sun received his Ph.D. degree in information and communication engineering from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), Hefei, in 1997. Currently, he is a professor in the School of Cyber Science and Technology, USTC, Hefei. His research interests include multimedia security, network intelligence and security and so on. He has published more than 120 papers in international journals and conferences. He is a fellow of IEEE.