DTGuard: A Lightweight Defence Mechanism Against a New DoS Attack on SDN

Jianwei Hou, Ziqi Zhang, Wenchang Shi(✉), Bo Qin, and Liang Bin

Renmin University of China, Beijing 100872, People's Republic of China
{houjianwei,zhangziqi,wenchang,bo.qin,liangb}@ruc.edu.cn

Abstract. The decoupling of the control plane and the data plane in Software-Defined Networking (SDN) enables the flexible and centralized control of networks. The two planes communicate via the southbound interface. However, the limited communication bandwidth on the southbound interface is exposed to potential denial of services (DoS) threats that may compromise the functions of southbound interface and even affect the whole SDN network. Some research has already focused on DoS attacks on the southbound interface and explored some countermeasures. Most of them are primarily concerned with the risk of malicious uplink traffic from the data plane to the control plane while few work expresses concern about downlink traffic from the control plane to the data plane. However, the threat of downlink traffic is also severe. In this paper, we reveal a DoS threat of amplified downlink traffic and implement a novel DoS attack, called control-to-data plane saturation attack, to demonstrate the threat. To mitigate such threats, we propose a lightweight defence mechanism called DTGuard that can monitor and identify abnormal ports based on a random forest classifier and migrate abnormal traffic along with a low-load link timely. The design of DTGuard conforms to the OpenFlow protocol without introducing additional modifications on the devices. The experimental results show that DTGuard can effectively mitigate the control-to-data plane saturation attack with a minor overhead on the controller.

Keywords: SDN · Network security · DoS attack · Mitigation

1 Introduction

Software-Defined Networking (SDN) is a networking architecture that decouples the control logic from the forwarding logic in a network providing high flexibility

This work is partially supported by the Natural Science Foundation of China under grant No. 61472429, Natural Science Foundation of Beijing Municipality under grant No. 4122041, and National High Technology Research and Development Program of China under grant No. 2007AA01Z414. The first two authors contributed equally to this research.

J. Zhou et al. (Eds.): ICICS 2019, LNCS 11999, pp. 503–520, 2020.
https://doi.org/10.1007/978-3-030-41579-2_29

and programmability. A typical SDN architecture consists of a logically centralized controller in the control plane, a set of network devices (such as switches) in the data plane and various applications in the application plane [7]. The control plane dictates the whole network behavior. It issues control messages to switches to specify their actions via the southbound interface.

The limited bandwidth of the southbound interface could be a bottleneck of an SDN network, which opens a new venue for attackers [11]. Attackers can launch a Denial of Service (DoS) attack against SDN by overloading the southbound interface. Since the core control information of an SDN network is delivered via the southbound interface, such as device management, link discovery and topology management, the dysfunction of the southbound interface may significantly downgrade the performance of the whole SDN network.

Previous work on DoS threats to the southbound interface focused on uplink traffic from the data to control planes [6,10,12,14,15,17,18]. The successful attacks can fingerprint match fields of the flow rules [19], thus to craft massive table-miss packets that may trigger massive Packet-In messages from the switch to the controller, exhausting computing or networking resources. However, to our knowledge, few work is concerned with the threat of downlink traffic from the control to data planes, which also needs to be taken seriously.

In the paper, we focus on the DoS threat of amplified downlink traffic that may overload the southbound interface and even eventually lead to the whole SDN network dysfunction. To demonstrate that the threat does exist in SDN, we propose and implement a new SDN-aimed DoS attack, called control-to-data plane saturation attack. The attack can leverage a handful of crafted packets that can trigger the Flood operation of the controller to generate amplified downlink traffic, which will increase the load on the southbound interface and may finally compromise the whole network performance. This raises a serious alarm because it is an effective attack that can leverage a small amount of traffic to incur significant performance degradation on an SDN network.

To mitigate the DoS threat of downlink traffic to SDN networks, we present an efficient and lightweight defence mechanism, called DTGuard, to provide automatic and real-time detection of control-to-data plane saturation attack. Utilizing a pre-trained random forest classifier, DTGuard can classify normal and abnormal ports of the switch based on traffic-based features. It can detect the attack timely under the low attack rate and migrate the abnormal traffic on the southbound interface to the data plane to effectively mitigate the attack.

The main contributions of this work can be summarized as follows:

- We reveal the risk of denial of services introduced by amplified downlink traffic that it can overload the southbound interface and even paralyze the whole network.
- We present a novel DoS attack, control-to-data plane saturation attack, to demonstrate the threat we have revealed. The attack can leverage a few crafted packets to generate amplified traffic, overloading the southbound interface and exhausting the bandwidth in the data plane. We implemented the attack on an SDN simulation environment under different SDN controllers to demonstrate the feasibility and generality of it.

– We propose a lightweight defence system against the downlink DoS attack based on a random forest model, called DTGuard. DTGuard is protocol-independent without additional modifications on switches. Experimental results show that DTGuard can efficiently identify the abnormal port of the switch and migrate malicious traffic with a negligible performance overhead.

The rest of the paper is organized as follows. We discuss the related work in Sect. 2. In Sect. 3, we illustrate the design and implementation of the control-to-data plane saturation attack to demonstrate the threat exists indeed. To mitigate the DoS threat of downlink traffic, we propose a defence system, called DTGuard. The detailed design of DTGuard is presented in Sect. 4. The implementation and evaluation of DTGuard are illustrated in Sect. 5. Finally, the paper is concluded in Sect. 6.

2 Related Work

The southbound interface that supports the interaction between the control and data planes may pose a potential new attack surface, leading to severe threats to SDN security [16]. Attackers can launch DoS attacks to exhaust the limited bandwidth on the southbound interface. There is already much research on the DoS threat of uplink traffic to explore mitigation methods. The mitigation methods mainly fall into three groups: (i) those based on traffic caching/migration (ii) those based on traffic feature extraction (iii) those based on traffic filtering.

The core idea of the mitigation methods based on traffic migration is to migrate malicious traffic to non-critical links or caches to reduce the load on the southbound interface. AVANT-GUARD [17] introduced a proxy to sift failed TCP sessions in the data plane prior to being sent to the control plane, which can reduce interaction times between the data and control planes. AVANT-GUARD is a protocol-dependent defence system against SYN flood attacks. FloodGuard [18] prevented the controller from overload by installing proactive flow rules and temporarily caching table-miss packets in a data-plane cache. Table-miss packets in the cache would be sent as Packet-In messages to the controller later at a low rate. FloodGuard breaks the protocol-dependent limitation in AVANT-GUARD but may lead to long delay and high packet loss rate for some flows. FloodDefender [6] proposed to offload malicious traffic to neighbor switches to migrate the load of the compromised link, and employed an Support Vector Machine (SVM) model to identify the attack traffic.

Some mitigation mechanisms of DoS attacks try to distinguish malicious traffic from normal traffic based on traffic-based features. Hu *et al.* [14] proposed an entropy-based detection scheme to identify whether a DDoS attack occurs by calculating the entropy of the IP address for each new stream. Mousavi *et al.* [10] also proposed an entropy-based method to measure the change of network features (include source and destination IP addresses, source and destination IP ports) and used an SVM classifier to classify the network traffic. Peng *et al.* [15] utilized DPTCM-KNN algorithm to measure the difference between abnormal traffic and normal traffic to identify the anomalies.

There are also some research efforts on defending SDN-aimed DoS attacks by filtering abnormal traffic. Kotani and Okabe [12] proposed a defence mechanism that it could filter out less important Packet-In messages without dropping important ones to keep a low level of load on switches. The switches record the values of the header fields before sending the packets to the controller. When the following packets with the same header fields arrive, the switch would temporarily cache the packets before the corresponding flow rule of these packets was installed on the flow table, which can prevent a large number of Packet-In messages from being sent to the controller in a short time.

All of the above work focused on the mitigation strategies to SDN-aimed DoS attacks of uplink traffic. However, the severe DoS threat of downlink traffic to the southbound interface also needs much attention. We implement a novel attack to reveal this threat and explore countermeasures to mitigate the threat.

3 Control-to-Data Plane Saturation Attack

In this section, to illustrate the DoS risk introduced by downlink traffic on the southbound interface, we present a new DoS attack on SDN networks, called control-to-data plane saturation attack. We first introduce the basic knowledge on "OpenFlow" [13], which is the most widely accepted southbound protocol. In this paper, we focus on the SDN networks that use OpenFlow protocol. Then we present the adversary model and details of the attack. We implement and evaluate the feasibility and effects of the attack in an SDN simulation environment under different SDN controllers.

3.1 Packet Processing in SDN

SDN separates the control and data planes by defining an open and standardized southbound interface and a protocol (e.g., the OpenFlow protocol) to access such interface. All traffic between the two planes passes through this interface.

Each OpenFlow-enabled switch maintains one or more flow tables and handles flows depending on the flow rules in the flow table. When a packet arrives, if there is a matched rule, the switch will directly deal with the packet according to the rule. If there is no matched rule, the packet will be sent to the controller in the form of a Packet-In message. The controller parses the Packet-In packet to make appropriate decisions and installs a corresponding flow rule on the switch. Controllers usually deliver the following three types of decisions via the southbound interface:

- Forward: forwarding the packet along a path to the specified port. The controller calculates the forwarding path based on the information of source and destination hosts, and then sends Flow-Mod messages to all the switches on the path to establish a connection between the source and destination hosts.
- Flood: flooding the packets in the data plane. The controller will send Packet-Out messages to all switches.

– Drop: dropping the packet. The controller sends a Flow-Mod message to the switch that reports this packet and installs a drop rule for packets of this flow on the switch.

3.2 Adversary Model

We assume that an adversary can control one or more hosts or virtual machines to craft packets and generate attack traffic. However, we do not assume the adversary can compromise the controller, applications or switches.

Based on the analysis presented in Sect. 3.1, there remains a possibility for adversaries to craft a few packets that can trigger amplified traffic from the control to the data plane. We analyze the number of triggered packets downward via the southbound interface according to the three decisions of the controller:

– Case-1: when a packet triggers a Drop decision, there are 2 packets generated on the southbound interface, that is, 1 Packet-In packet and 1 Flow-Mod packet.
– Case-2: when a packet triggers a Forward decision, there are $1 + P$ packets generated on the southbound interface, that is, 1 Packet-In packet and P Flow-Mod packets (P is the number of switches on the forwarding path).
– Case-3: when a packet triggers a Flood decision, there will be $N + N$ or $N + 1$ (based on control logic of different controllers) packets generated on the southbound interface. For some controllers (such as RYU, Floodlight), all switches will send Packet-In messages to the controller and the controller will issue Packet-Out messages to all switches to flood the packets to all the hosts in the network. That is, N Packet-In packets and N Packet-Out packets (N is the number of all switches in the network). For some other controllers (such as OpenDayLight), only one switch will send a Packet-In message to the controller and the controller will send Packet-Out messages to all the switches. That is, 1 Packet-In packet and N Packet-Out packets.

In general, Case-1 and Case-2 generate little traffic, having little impact on the network. However, when a Flood decision is triggered, the controller needs to communicate with all switches in the network, which opens a door for attackers to generate overwhelming downlink traffic by crafting a few crafted packets.

3.3 Attack Method

When an adversary is going to launch an attack with overwhelming downlink traffic, the key issue that he/she concerns most is **how to craft a packet that can trigger the Flood decision of the controller**.

On one hand, based on TCP/IP protocol, the controller will send broadcast packets (e.g., ARP request packets, DHCP request) to all hosts in the network by default, which triggers the Flood decision.

On the other hand, the controller maintains a list of known hosts in the SDN network. When a Packet-In message arrives, the controller will extract the

source address and the location of the host (i.e., the ID and port of the switch connected with the host) from the message and add this information into the host list. Therefore, when receiving a packet whose destination address is not included in the host list, the controller need to send packets to all hosts in the network, which also triggers the Flood decision. After the destination host replies to this packet, the controller can learn its location and add the destination host to its host list.

We implemented experiments on three popular SDN controllers (i.e., Floodlight, RYU, and OpenDaylight) to verify whether the controller will make the Flood decision on these two kinds of packets. The results show that all the three controllers make Flood decision on the two kinds of packets and send packets to all hosts in the network.

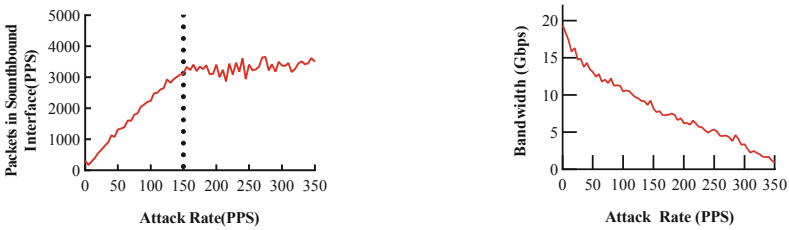
Based on the analysis above, we design a new SDN-based DoS attack, called control-to-data plane saturation attack. The attack crafts the packets that can trigger the Flood decision of a controller to generates amplified downlink traffic, overloading the southbound interface. Compared with existing SDN-based DoS attack on the southbound interface, this attack has two characteristics. One is that adversaries only need to send a handful of packets at a low rate, so it is difficult to detect the active malicious host. The other characteristic of the attack is good concealment because the amplified traffic is generated by the controller which is highly trusted by the devices in the data plane.

3.4 Attack Evaluation

To demonstrate the DoS threat of downlink traffic exists indeed, we implement the control-to-data plane saturation attack on a simulation testbed using Mininet [1]. We adopt the Fat-tree topology, a common topology for data center networks [3], with pod = 4 and host density = 2 (that is, each edge switch connects with 2 terminal hosts). The experimental topology is shown in Fig. 2.

We select a host in the data plane to craft the UDP packets whose destination addresses are not in the host list of the controller by randomizing destination IP addresses of the packets. These packets can trigger the Flood decision of the controller, generating the amplified traffic.

The amplification factors of the attack traffic under the three controllers are shown in Table 1. When there is no attack traffic, the load of the southbound interface keeps a lower state. There is communication between switches and the controller for some normal tasks, such as transferring heartbeat packets and sending LLDP packets for link discovery. When the attack occurs, based on the analysis in Sect. 3.2, the amplification factors of RYU and Floodlight should be two times of the number of switches in theory, that is 40 in our testbed. As shown in Table 1, the practical amplification factors reach about 25, which are lower than the theoretical value, restricted by network environment and network resources. And in the testbed under OpenDaylight controller, because only one switch will send Packet-In message to the controller and all switch will receive Packet-Out message for Flood, the theoretical value of the amplification factor is $N + 1$ (that is 21 in our testbed) while the experimental value is about 13.



(a) Load of the southbound interface (b) Network bandwidth

Fig. 1. Effects of different attack rates on SDN under Floodlight controller

Table 1. Summary of amplification factors in the networks under three controllers

Controller	Attack rate (PPS)	The load of southbound interface (PPS)	Amplification factor
RYU	0	56	25.56
	50	1334	
Floodlight	0	68	24.94
	50	1315	
OpendayLight	0	112	13.32
	50	778	

We use TCPDUMP to evaluate the relationship between the attack rate and the load of the southbound interface while using iperf to measure the network bandwidth of host communication in the data plane under different attack rate. The test results are shown in Fig. 1 (Floodlight as an example).

As shown in Fig. 1(a), the crafted packets have amplification effects on the traffic of the southbound interface. The load of southbound interface multiplies with the increase of attack rate and tends to saturate when the attack rate reaches around 150 pps. It can be seen from Fig. 1(b) that the network bandwidth between hosts decreases from 20 Gbps with the increase of the attack rate. The attack rate at around 50 pps can lead to network fluctuation, and the network bandwidth tends to 0 when the attack rate reaches around 350 pps.

Therefore, we can draw a conclusion that the control-to-data plane saturation attack can overload the southbound interface and also lead to network collapse in the data plane, affecting the normal communication between hosts.

4 Proposed Countermeasure

The control-to-data plane saturation attack is an SDN-aimed DoS attack rooted in the SDN architecture because of the decoupling of the control and data planes. To detect and mitigate this attack, we introduce DTGuard, an efficient, lightweight, and protocol-independent defence mechanism. We present the detailed design of DTGuard in this section.

4.1 System Architecture

The basic idea of DTGuard is to distinguish the abnormal port from the normal by a machine learning method to detect the attack. The switch's port that connected with the attack host is an abnormal port. And DTGuard migrates the overwhelming traffic concentrated on the southbound interface to the data plane when the attack occurs. DTGuard mainly consists of four modules including an attack detection module, a traffic statistics module, a path calculation module, and a flow rule generation module, as shown in Fig. 3.

The attack detection module keeps active after the controller starts up to detect whether there is an abnormal port. When an abnormal port is detected, DTGuard activates the other three modules to handle the attack.

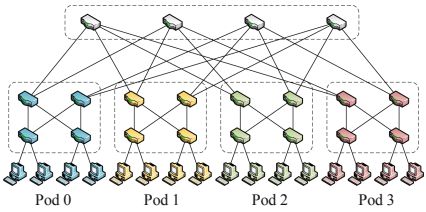


Fig. 2. Experimental topology

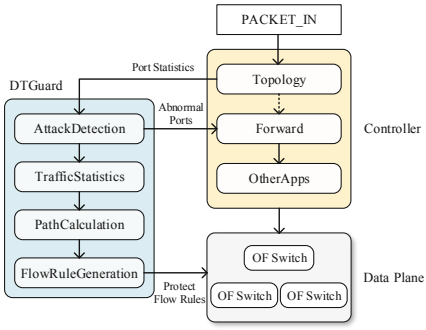


Fig. 3. The architecture of DTGuard

4.2 Attack Detection Module

The attack detection module executes regularly at a period of time T to monitor the traffic on each port of the switch connected with hosts and extract traffic features of the port. It leverages a pre-trained classifier based on a random forest model [9] to classify the traffic on the port either as normal or abnormal based on six traffic-based features. The six features of the traffic on a port will be discussed in more detail below.

Success Rate of Flow Rule Matching (SRf): To generate control-to-data plane traffic, attackers need to craft the packets with no matched flow rules in the switches' flow tables. Therefore, when the attack occurs, the success rate of flow rule matching may decrease. We use Eq. (1) to compute SRf.

$$SRf = \frac{Num_{matched}}{Num_{received}} \tag{1}$$

$Num_{received}$ represents the total number of packets received by a port of the switch in the period T while $Num_{matched}$ represents the number of packets that match the flow rule successfully.

Rate of Bidirectional Flows (RBf): Normally, the dataflow between hosts in the network is bidirectional. That is, one host sends packets to other hosts, and other hosts respond after receiving the packets. When an attack occurs, the number of unidirectional dataflows may increase because attackers send packets with fake destination IP addresses. Therefore, the rate of bidirectional flows may decrease [8]. We use Eq. (2) to compute RBf.

$$RBf = \frac{Num_{pair}}{FlowNum} \tag{2}$$

Num_{pair} is the number of bidirectional flows passing through a port in the period T, and $FlowNum$ is the number of all flows passing through the port.

Trigger Rate of Flood Action (TRf): In a normal network environment, most traffic will be forwarded directly according to the flow rules on switches. Only a small amount of traffic is reported to the controller and triggers the Flood action. However, when the attack occurs, the number of packets triggering the Flood decision increases, resulting in a significant increase in trigger rate of Flood action. We use Eq. (3) to calculated the trigger rate.

$$TRf = \frac{Num_{Flood}}{PacketInNum} \tag{3}$$

$PacketInNum$ is the number of all Packet-In messages received by the controller from a port during the period T, and Num_{Flood} is the number of Packet-In messages that trigger the Flood decision.

The Entropy of Destination IP Address: Usually, the destination address for host communication keeps stable. But when an attack occurs, a large number of packets with random destination IP addresses are generated, which increases the uncertainty of the corresponding destination IP address of a certain source IP address [10]. The entropy is a measure of the uncertainty of random variables in information theory. Therefore, we use the entropy of destination IP address to measure the changes of destination IP addresses that correspond to a source IP address. The calculation method is as follows:

We define the Map L between the destination IP address $dstIP_i$ and the occurrences c_i of this IP on a port in T period as Eq. (4):

$$L = \{(dstIP_1, c_1), (dstIP_2, c_2) ... (dstIP_n, c_n)\} \tag{4}$$

The $dstIP$ is the hash value of the destination IP address, and the c_i is the number of occurrences of the destination IP address. The appearing probability of each destination IP address is as Eq. (5):

$$p_i = \frac{c_i}{\sum_{i=1}^{n} c_i} \tag{5}$$

According to the definition of entropy, the entropy of each destination IP address can be calculated as Eq. (6):

$$H = -\sum_{i=1}^{n} p_i \log p_i \tag{6}$$

Trigger Rate of Flow-Mod ($TRfd$): When an attack occurs, a large number of packets with random destination IP addresses are generated. There may be a decrease in the number of Flow-Mod messages sent to the switch because the controller does not know the location of the destination host. We use Eq. (7) to calculate $TRfd$.

$$TRfd = \frac{Num_{FlowMod}}{PacketInNum} \tag{7}$$

$PacketInNum$ is the number of all Packet-In messages received by the controller from a port during the period T, and $Num_{FlowMod}$ is the number of Packet-In messages that trigger Flow-Mod messages.

Average of Packets per Flow (APf): We define the data flow that can trigger the rule installation as a valid data flow. When an attack occurs, the attack packets are directly flooded to the data plane by the controller without installing new rules on the switches, so that no valid flow is formed. As a result, during an attack, the number of packets $PacketNumber$ may increases while the number of valid flows $FlowCount$ may remain unchanged, which eventually leads to an increase in the average number of packet per flow. The average of packets per flow is calculated as Eq. (8).

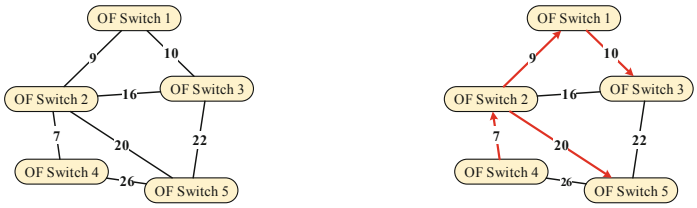
$$APf = \frac{PacketNumber}{FlowCount} \tag{8}$$

These six traffic-based features constitute a 6-tuple to be the input of the classifier to classify network traffic either as normal or abnormal. We use the random forest model to train the classifier. The random forest is an easy-implemented classification method with low computational overhead and strong generalization ability. The implementation of the classifier is described in Sect. 5.1.

4.3 Traffic Statistics Module

The traffic statistics module obtains the traffic information of each port of all switches from the detection module. Leveraging this information, it maps the traffic information of the port to the corresponding link based on the network topology and constructs a weighted graph of the network traffic.

The traffic of each link changes rapidly in a network. To reduce the impact of network fluctuations on the sampling results, we divide the link load into multiple levels. The granularity of the level can be adjusted according to the actual network environment. For example, if the granularity is set as 100, the level of the load ranged from 0–100 can be labeled as level 1, the level of the load ranged from 101–200 can be labeled as level 2, and so on. An example of a weighted graph of the network traffic is shown in Fig. 4(a).



(a) A weighted graph of network traffic (b) The forwarding path in data plane

Fig. 4. A weighted graph of network traffic and the corresponding forwarding path

4.4 Path Calculation Module

Path calculation module aims to select an optimistic forwarding path to migrate abnormal traffic. The path selection follows the principles of low load and no repetition. The forwarding path can be obtained by calculating a Minimum spanning tree (MST) of the weighted graph of the network traffic. When an attack is detected, the controller sends one Packet-Out message to the root node of the MST. The switch at the root node forwards the packet step by step along each edge of the MST to achieve flood function. An example of the forwarding path is shown in Fig. 4(b).

We use Prim algorithm to calculate the MST of the network, and the time complexity of the first calculation is $O(n \log n)$. When the load of some links in the network changes, it would not always bring changes to the MST. In this case, to reduce computational overhead, it is usually not necessary to recalculate the entire MST. Only when the link topology changes or when the change of link load have an impact on the MST, the MST will dynamically update according to the result generated by the previous calculation. Based on four cases of the load update, we detailedly discuss the update process of MST.

Given the weighted graph G, the vertex set V, and the set of edges E, calculate the current MST T. When the weight of an edge e changes, w_{old} represents the weight before the update, and w represents the updated weight.

- Case-1: $w_{old} < w$ and $e \notin T$, the weight of e increases. Since e does not belong to T, the increase of its weight does not affect the current MST, T remains unchanged;
- Case-2: $w_{old} > w$ and $e \in T$, the weight of e decreases. Since e is a part of T, the decrease of its weight does not change the current MST, T remains unchanged;
- Case-3: $w_{old} > w$ and $e \notin T$, the weight of e decreases. A new tree may be constructed with a smaller weight because of the decrease of e. First, edge e can be added to the current MST T to get $T^{'}$. Therefore, there is a loop C in $T^{'}$, as shown in Fig. 5(a). The red edge in the figure is e. According to the loop theorem of the minimum spanning tree, the edge with the largest weight in loop C should be removed from $T^{'}$ to get the new MST. The time complexity of the algorithm for this case is $O(n)$.

– Case-4: $w_{old} < w$ and $e \in T$, the weight of e increases. There may be another edge with a lower weight than e to construct a new MST because of the increase of e. First, the edge e can be deleted from the current MST T to obtain $T^{'}$. Therefore, there are two independent subtrees in $T^{'}$. a and b are vertexes of $e(a, b)$. A is the set of vertexes that can be reached from vertex a in $T^{'}$ and B is the set of vertexes that can be reached from vertex b in $T^{'}$, $A \cap B = \emptyset$ and $A \cup B = V$. C is a cut-set of edges in graph G. As shown in Fig. 5(b), e is the blue edge, $C = \{s_1, s_2, s_3, e\}$ is the set of edges connecting the two subtrees. When the weight of e increases, according to the cut property of the minimum spanning tree, the edge with the smallest weight in C should be added to $T^{'}$ to get the new MST. The time complexity of the algorithm for this case is $O(n)$.

Therefore, the change of load on the link may not always lead to the change of MST. When the load update changes the MST, the new MST is recalculate based on the previous calculation without introducing too much overhead.

(a) $w_{old} > w$ and $e \notin T$ (b) $w_{old} < w$ and $e \in T$

Fig. 5. Dynamic updating of minimum spanning tree (Color figure online)

4.5 Flow Rule Generation Module

Flow rule generation module generates and installs the corresponding flow rules on the switches in the MST based on the forwarding path obtained from the path calculation module. When the controller makes a Flood decision, there are two ways to execute the Flood action for two cases.

– Case-1: if the source port of the packet that triggers the Flood is an abnormal port, the controller will send a Packet-OUT message only to the switch at the root of the MST. The root node forwards the packet along the MST until the packet is delivered to all ports of all switches in the network.
– Case-2: if the source port of the packet that triggers the Flood is a normal port, the controller will flood the packets to all switches directly.

We use a reserved field of the IP header field, ToS, to enable the switch to distinguish whether a packet is from a normal host or an abnormal host. For Case-1, the match field IP_TOS keeps as defaults, IP_TOS = 0b00000000. For Case-2, the controller sets the match field IP_TOS = 0b00000011 before sending out the Packet-Out message to the root switch of the MST.

5 Implementation and Evaluation

In this section, we implement the prototype of DTGuard on Floodlight in a simulated SDN environment to evaluate its performance and overhead.

5.1 Implementation

TestBed. Using Mininet-2.3.0 and Open vSwitch-2.0.2 to simulate the underlying network, we deploy the controller and the underlying network on a single physical machine with 7.7 GB memory, 4 cores at 2.13 GHz. The controller communicates with switches via OpenFlow 1.3 protocol. The experimental topology is a fat tree topology with pod = 4 and host density = 2, as shown in Fig. 2.

We implement the prototype of DTGuard on an open source SDN controller Floodlight without additional modifications on the data plane. The four modules of DTGuard are implemented in Java language. The Attack Detection module is mounted on the Packet-In processing chain of Floodlight and starts up at the same time as the controller startup. The other three modules are activated when an abnormal port is detected by the Attack Detection module.

Traffic Generation. To generate the traffic similar to the real traffic in the real network environment, we have extended the Mininet so that hosts can communicate with each other to simulate normal traffic. The simulated normal traffic falls into two parts, one is the random traffic between hosts, and the other is the traffic of common services.

- Traffic between hosts: The $No.m$ host in the network sends packets to the $No.(m+i)$ host, the $No.(m+j)$ host, the $No.(m+k)$ host with probabilities of P_t, P_a, and P_c, respectively. The legitimate traffic generated during tests is a composition of several different protocols (TCP (85%), UDP (10%), ICMP (5%)) based on the statistics of network traffic in the real world [4,5].
- Traffic of common services: to simulate the network traffic based on the C/S model, we have selected some hosts as servers that are deployed with some common services(e.g., FTP, HTTP) while other hosts as clients request these services at a random probability.

Training Classifier Model. To generate training samples, we set a time window of the controller to 10 s and calculate traffic-based 6-tuple of the 10 s for each port. To generate abnormal traffic, we use Scapy [2] to craft the UDP packets with random destination IP addresses which can trigger the Flood decision of the controller. The attack rate increases at a step of 25 pps, starting from 0 pps to 350 pps. We have collected 1,000 samples at each attack rate from abnormal ports and normal ports, respectively. We totally collected 30,000 samples to train the classifier, 70% of the sample as the training set and 30% as the test set.

5.2 Results and Evaluation

Attack Detection Effects. The effectiveness of our detection mechanism is evaluated through Detection Rate (DR) and False Alarm rate (FA) measurement. DR and FA are defined as Eq. (9):

$$DR = \frac{TP}{TP + FN} \qquad FA = \frac{FP}{TN + FP} \tag{9}$$

TP (True Positives) represents the number of abnormal ports that are classified as abnormal, and FN (False Negatives) represents the number of abnormal ports that are classified as normal. TN (True Negatives) represents the number of normal ports that are classified as normal and FP (False Positives) represents the number of normal ports that are classified as abnormal.

The attack detection effect of the classifier is depicted in Fig. 6. As the attack rate increases, the difference between the features of normal and abnormal traffic becomes more and more obvious, so that the DR increases. When the attack rate reaches 25 pps, the DR keeps stable at 98% or more. The FA is stable below 1%, as shown in Fig. 7. The results demonstrate the classifier can distinguish the abnormal traffic at both high attack rate and low attack rate with high accuracy. Therefore, DTGuard can detect the DoS attack at an early stage under a low attack rate, preventing the network from significant performance degradation.

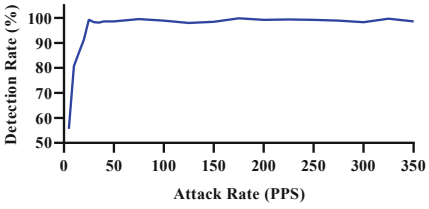


Fig. 6. Detection rate

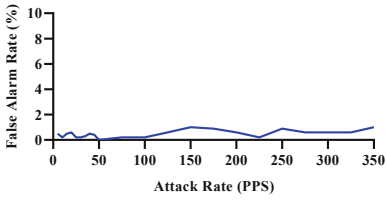


Fig. 7. False alarm rate

Defence Effects. To demonstrate the defence effect of DTGuard, we have launched control-to-data plane saturation attacks in two environments with: (i) an original Floodlight controller, or (ii) a Floodlight controller with DTGuard.

We use TCPDUMP to measure the load of southbound interface in both two environments. As shown in Fig. 8, in (i) environment, as the attack rate increases, the load of the southbound interface multiplies, which tends to saturate at about 150 pps attack rate. In (ii) environment, because the malicious traffic is migrated to the data plane, eliminating the amplification effect of traffic, the load of the southbound interface is significantly reduced. The statistics results of the test in Fig. 8 show that DTGuard reduces the southbound interface load by an average of 74.2%, which demonstrates the effectiveness of DTGuard.

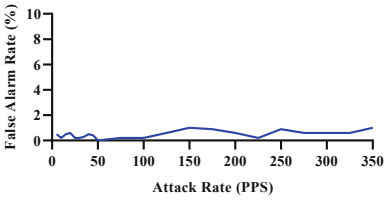


Fig. 8. Load of southbound interface

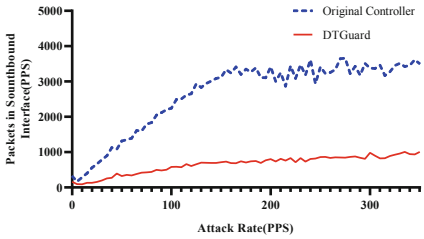


Fig. 9. Network bandwidth

We use iperf to measure the network bandwidth of host communication in both two environments, as shown in Fig. 9. The normal network bandwidth is about 20 Gbps when there is no attack. As the attack rate increases, the malicious traffic gradually overwhelms the southbound interface in (i) environment, so that the network bandwidth is significantly reduced. In (ii) environment, although the network bandwidth also decreases with the increase of the attack rate, the declining trend of network bandwidth slows down. The statistics results of the test in Fig. 9 show that DTGuard prevents network bandwidth against the decline by about 41.7% on average at the same attack rate.

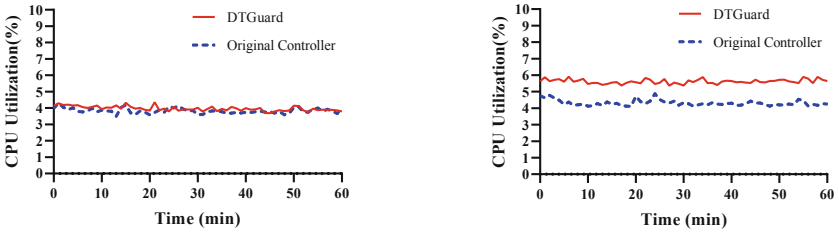
Overhead Analysis. Each module of Floodlight chains together to deal with the received packets. When a Packet-In message arrives, it will be processed along the chain of modules. To evaluate the extra processing time brought by DTGuard, we use Floodlight's PacketIn Processing Time Service to measure the running time of each module.

Table 2. PacketIn processing time of each module

Module	Normal network	Network under attack	
		init-stage	following-stage
DTGuard	4.90%	15.62%	9.71%
Forwarding	80.85%	72.19%	76.25%
DeviceManagerImpl	8.93%	7.92%	9.63%
LinkDiscoveryManager	3.94%	2.71%	2.87%
Others	1.37%	1.56%	1.54%

When there is no abnormal port detected, DTGuard only activates the Attack Detection module. As shown in Table 2, in this situation, DTGuard takes only 4.9% of the time of the processing chain. When an abnormal port is detected, the other three modules are also activated. It takes 15.62% of the time of the processing chain at the initial execution after the attack because DTGuard needs to construct the entire MST. The following cost of DTGuard accounts for 9.71%

of the time of the processing chain to update the MST. Compared with the Forwarding module, DTGuard takes little processing time, which may not bring much delay to the controller.



(a) Normal network (b) Network under attack

Fig. 10. CPU load of original controller and DTGuard

We used JProfiler to measure the CPU utilization of the controllers in environments (i), (ii) before and after the attack, as shown in Fig. 10. Figure 10(a) shows the CPU utilization of the controllers when there is no attack. In this situation, the CPU utilization curve of the controller with DTGuard almost coincides with that of the original controller. Note that there is only the attack detection module of the DTGuard starting up at this point. Figure 10(b) shows the CPU utilization of controllers when an attack occurs. Note that all four modules of the DTGuard start up at this point. The CPU utilization increases by about 1.3% in this situation. Overall, the overhead of DTGuard on the controller is very little.

6 Conclusion

The southbound interface that provides communication between the control and data planes in an SDN faces potential DoS threats. In this paper, we have revealed a severe DoS threat of amplified downlink traffic to SDN security. To demonstrate the threat indeed exists in general SDN networks, we implement a new SDN-aimed DoS attack called control-to-data plane saturation attack on the testbed under three different SDN controllers. The experimental results demonstrate that the attack can leverage a handful of crafted packets to generate more than the 13 times amplification effect of the attack traffic, exhausting the costly network bandwidth and downgrading the network performance.

To mitigate the control-to-data plane saturation threat in SDN, we propose a lightweight mitigation mechanism, called DTGuard. DTGuard can efficiently detect and identify the abnormal ports by a random forest classifier without modifications in the data plane. Once an attack is detected, it can migrate the attack traffic timely to the data plane along a path with a low load. The experimental results show that DTGuard can precisely detect the control-to-data plane saturation attack and effectively mitigate the abnormal traffic with minimal overheads.

References

1. Mininet. http://mininet.org/
2. Scapy. http://www.secdev.org/projects/scapy/
3. Bari, M.F., et al.: Data center network virtualization: a survey. IEEE Commun. Surv. Tutor. **15**(2), 909–928 (2013)
4. Borgnat, P., Dewaele, G., Fukuda, K., Abry, P., Cho, K.: Seven years and one day: sketching the evolution of internet traffic. In: 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2009), 19–25 April 2009, Rio de Janeiro, Brazil, pp. 711–719 (2009)
5. Braga, R., de Souza Mota, E., Passito, A.: Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: Proceedings of the 35th Annual IEEE Conference on Local Computer Networks (LCN 2010), 10–14 October 2010, Denver, Colorado, USA, pp. 408–415 (2010)
6. Gao, S., Peng, Z., Xiao, B., Hu, A., Ren, K.: FloodDefender: protecting data and control plane resources under SDN-aimed DoS attacks. In: 2017 IEEE Conference on Computer Communications (INFOCOM 2017), 1–4 May 2017, Atlanta, GA, USA, pp. 1–9 (2017)
7. Gude, N., et al.: NOX: towards an operating system for networks. ACM SIGCOMM Comput. Commun. Rev. **38**(3), 105–110 (2008)
8. Guo, R., Yin, H., Wang, D., Zhang, B.: Research on the active DDoS filtering algorithm based on IP flow. IJCNS **2**(7), 600–607 (2009)
9. Ho, T.K.: Random decision forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition, vol. 1, pp. 278–282. IEEE (1995)
10. Hu, D., Hong, P., Chen, Y.: FADM: DDoS flooding attack detection and mitigation system in software-defined networking. In: 2017 IEEE Global Communications Conference (GLOBECOM 2017), 4–8 December 2017, Singapore, pp. 1–7 (2017)
11. Imran, M., Durad, M.H., Khan, F.A., Derhab, A.: Toward an optimal solution against denial of service attacks in software defined networks. Future Gener. Comp. Syst. **92**, 444–453 (2019)
12. Kotani, D., Okabe, Y.: A packet-in message filtering mechanism for protection of control plane in OpenFlow networks. In: Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2014), 20–21 October 2014, Los Angeles, CA, USA, pp. 29–40 (2014)
13. McKeown, N., et al.: OpenFlow: enabling innovation in campus networks. Comput. Commun. Rev. **38**(2), 69–74 (2008)
14. Mousavi, S.M., St-Hilaire, M.: Early detection of DDoS attacks against SDN controllers. In: International Conference on Computing, Networking and Communications (ICNC 2015), 16–19 February 2015, Garden Grove, CA, USA, pp. 77–81 (2015)
15. Peng, H., Sun, Z., Zhao, X., Tan, S., Sun, Z.: A detection method for anomaly flow in software defined network. IEEE Access **6**, 27809–27817 (2018)
16. Shin, S., Gu, G.: Attacking software-defined networks: a first feasibility study. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2013), The Chinese University of Hong Kong, Hong Kong, China, Friday, 16 August 2013, pp. 165–166 (2013)
17. Shin, S., Yegneswaran, V., Porras, P.A., Gu, G.: AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In: 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS 2013), 4–8 November 2013, Berlin, Germany, pp. 413–424 (2013)

18. Wang, H., Xu, L., Gu, G.: FloodGuard: a DoS attack prevention extension in software-defined networks. In: 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2015), 22–25 June 2015, Rio de Janeiro, Brazil, pp. 239–250 (2015)
19. Zhang, M., Hou, J., Zhang, Z., Shi, W., Qin, B., Liang, B.: Fine-grained fingerprinting threats to software-defined networks. In: 2017 IEEE Trustcom/BigDataSE/ICESS, 1–4 August 2017, Sydney, Australia, pp. 128–135 (2017)