SD-OVS: SYN Flooding Attack Defending Open vSwitch for SDN

Xinyu Liu(✉), Beumjin Cho, and Jong Kim

Department of Computer Science and Engineering, POSTECH, Pohang, Republic of Korea
{xinyuliu918,beumjincho,jkim}@postech.ac.kr

Abstract. Software defined networking (SDN) is a novel programmable networking paradigm that decouples control and data planes. SDN relies heavily on the controller in control plane that tells the data plane how to handle new packets. Because the entire network may be disrupted if the controller is disabled, many attacks including SYN flooding aim to overload the controller by passing through the ingress switches. In this paper, we propose a security enhanced Open vSwitch (SD-OVS) to protect the controller from SYN flooding. The switch authenticates benign hosts by interchanging cookie packets and generates a short-lived security association (SA). The retransmitted SYN packet from these benign hosts is validated using SA and passed on to the controller. Our evaluation shows that SD-OVS protects the controller from SYN flooding at an acceptable time cost.

Keywords: Software defined networking · OpenFlow · Open vSwitch · SYN flooding

1 Introduction

Software defined networking (SDN) has emerged as a new network paradigm that provides programmability by decoupling the control and data planes [1]. The programmer can manage and control the entire network centrally in a desired manner. The controller of SDN thus becomes the brain of the network because the controller determines the manner in which to deal with the packet; the controller then inserts the flow rules to switches. The switches then handle the packets based on the flow rules stored in the flow table.

Despite these benefits, the rapid emergence of SDN has also generated several types of threats [5–7]. We must ensure the security of SDN when we apply it. In a traditional network, SYN flooding [2] is one of the most common types of attacks that can be launched easily and has powerful effects on its target. The SYN flooding attack exhausts the backlog resource of the destination host by sending SYN packets so that the host generates and stores the half-open connection in a backlog.

To prevent the SYN flooding attack, mitigations such as increasing the backlog, reducing SYN-RECEIVED timer, or using firewalls and proxies must occur [3]. Some researchers have proposed three counter algorithms to ensure that the connection stored in a backlog is benign [4]. However, none of the previous methods can be applied to

D. Choi and S. Guilley (Eds.): WISA 2016, LNCS 10144, pp. 29–41, 2017.
DOI: 10.1007/978-3-319-56549-1_3

SDN directly because SYN flooding can disrupt an entire SDN network rather than just the destination host. For example, if an attacker launches an SYN flooding attack from one compromised host and sets the arrival of new packets to a sufficiently high rate, the attacker can disable the controller by making it busy to respond to packets; which will disrupt the entire network in consequence.

AVANT-GUARD [7] is the first attempt to modify the OpenFlow switch to mitigate an SYN flooding attack in an SDN. However, it has an adaptation problem, as it must modify the controller and switch together. Moreover, it requires additional resources to maintain TCP connections.

In this paper, we propose an SDN switch called SD-OVS, which protects the controller from an SYN flooding attack and enhances the stability of the switch by authenticating benign hosts through security association (SA). When an SD-OVS switch receives an SYN packet and no matching flow exists, the switch sends back an illegitimate SYNACK packet with a cookie. If the switch receives the RST packet with a valid cookie, the switch generates a security association. The SA acts as an identification of the host for the SYN packet and is deleted when the connection is established. The SD-OVS switch utilizes the retry mechanism of the TCP connection establishment. After receiving the retransmitted SYN packet that matches with a stored SA, the switch forwards it to a controller as part of the original process. SD-OVS protects the controller because it does not send a message about a new SYN packet to the controller if the host is not authenticated by the SA. In addition, it does not generate useless flow rules triggered by the SYN flooding attack in the switch.

Our study makes the following contributions:

- We propose a scalable and stable method called SD-OVS, which protects the SDN from an SYN flooding attack in a data plane without requiring any modification to the control plane.
- We evaluate the performance of SD-OVS both in normal operation and when under attack, and the results show that SD-OVS protects SDN from SYN flooding attacks at an acceptable time cost.

The remainder of this paper is organized as follows. In Sect. 2, we provide background information. Section 3 presents the working mechanism of the proposed switch SD-OVS. Section 4 provides details about the implementation of SD-OVS. Section 5 presents evaluation results. Section 6 discusses possible attack issues to be considered in the proposed switch. Section 7 presents related studies and a conclusion.

2 Background

In this section, we briefly introduce the architecture of the software defined networking (SDN), and then describe two components of SDN: OpenFlow Protocol and Open vSwitch (OVS). At the end of the section, we introduce the SYN cookie method.

2.1 Architecture of SDN

SDN consists of at least one protocol and the following three layers: application, control, and data forwarding layers (Fig. 1).

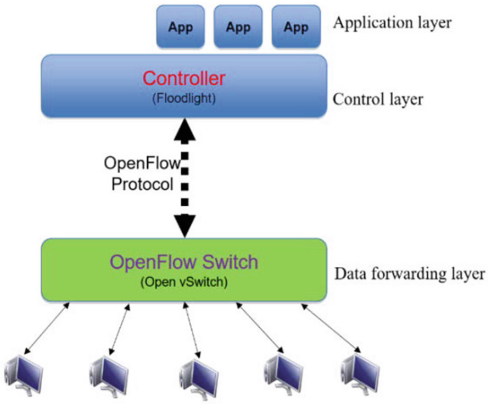


Fig. 1. Framework of SDN.

The protocol used in SDN defines the manner in which the control layer interacts with the data forwarding layer. The innovative applications and protocols are implemented in the application layer. The controller in the control plane communicates with the application layer through its northbound interface. Finally, the controller guides the switches in the data forwarding layer to handle the packets through the secure channel in the switch.

Vulnerability in SDN architecture. Because the network relies heavily on the controller, the entire network could be thrown into chaos or even totally disabled if the controller undergoes a critical attack such as an SYN flooding attack. In addition, because of the limited capacity of the switch, a benign flow rule cannot survive in a switch if several malicious flow rules compete; that is, they either cannot enter or can be dropped from the flow table.

2.2 OpenFlow Protocol

The OpenFlow protocol [1] is a typical and widely used protocol in SDN. The OpenFlow protocol specifies that every time a new packet arrives into the switch, the switch extracts the packet information and determines whether a matching flow exists in the flow table. If no matching flow is present, the switch asks the controller the course of action to take by sending a PACKET_IN message. The controller then creates a flow rule and sends it to all related switches for handling packets associated to the original new packet. Finally, the switch follows the flow rule to handle the packet.

2.3 Open vSwitch

Here, we introduce the OpenFlow switch and Open vSwitch (OVS).

An OpenFlow switch [8] consists of at least a flow table, secure channel, and OpenFlow protocol. The flow table stores the flow rules from the controller, and the secure channel is used for communication between the control and data planes.

Open vSwitch [9] is a widely used OpenFlow switch. Open vSwitch code space is divided into two spaces: kernel and user. The flow in the kernel space is the precise flow, which is set based on the flow in the user space. When a new packet comes into a switch, the switch first looks for a matching flow in the kernel space. If no matching flow is present in the kernel space, OVS sends it to the user space for a second search. If a matching flow exists in the user space, the switch moves the precise flow to the kernel space.

When a new SYN packet exists without a matching flow, the switch performs the action in the "Table Miss." The default "Table Miss Action" is to send the extracted packet to the controller [1]. The controller determines the manner in which to handle the new SYN packet and sends the corresponding flow rule to the user space of all relevant switches.

2.4 SYN Cookie

SYN cookie [10] is one of the mostly used techniques to resist SYN flooding attacks. Our method modifies part of the mechanism of SYN cookie. Thus we explain the working mechanism of it.

SYN cookie is activated if an attack occurs or the backlog queue becomes full. When it is activated, the destination host sends back the appropriate SYNACK packet without storing any information. This is performed to protect the host. When the host receives a subsequent ACK response, it stores the TCP connection into the backlog as a full connection. The sequence number of a SYNACK packet is called a "cookie," which is produced by a hash function with a source IP address, destination IP address, source port number, destination port number, sequence number, maximum segment size (MSS), current time, and random keys.

Drawback of SYN cookie. Although SYN cookie has powerful capabilities, it still possesses a drawback. The destination host can confirm the delivery of SYN cookie when it receives an ACK packet from the source. Otherwise, the destination cannot differentiate whether a SYN cookie from itself or an ACK packet from the source is lost [11]. The destination cannot re-transmit the SYNACK packet because it does not store any information in the backlog. This will produce an unsuccessful connection.

3 SD-OVS

In this section, we provide an overview of our method and explain each stage of SD-OVS in detail.

SD-OVS consists of four stages: auth-cookie generation, SA creation, SA matching, and SA deletion. The first three stages filter malicious SYN packets. The fourth stage provides better performance. The four stages operate as follows. First, a source sends a new SYN packet to a destination through a connected SD-OVS switch. When SD-OVS receives it, it responds with an illegitimate SYNACK packet containing a cookie. The source then responds with an RST packet with the received cookie. When SD-OVS receives the RST packet, it verifies the cookie. If the cookie is valid, SD-OVS creates an SA for this host and stores it. Finally, when the source resends the SYN packet according to the retry mechanism of the TCP/IP protocol, SD-OVS checks whether a matching SA is present. If it finds a matching SA, it sends the PACKET_IN message to the controller. The controller determines whether to set up a flow rule in the switches. SD-OVS then forwards the packet to the destination according to the flow rule to complete the TCP connection (Fig. 2).

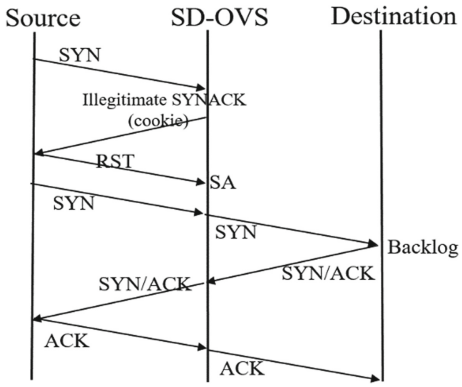


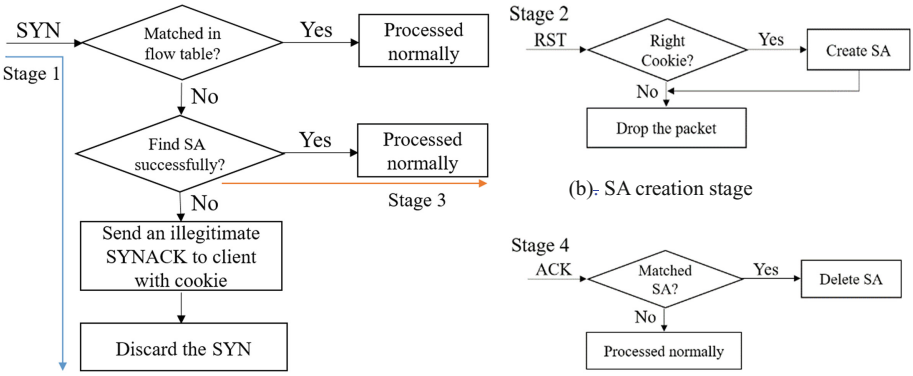
Fig. 2. Workflow of SD-OVS.

We next describe each of the four stages in SD-OVS.

3.1 Auth-Cookie Generation Stage

In Stage 1, SD-OVS sends an illegitimate SYNACK packet that contains a cookie. When the switch receives a SYN packet, the switch looks for a flow that matches it. If a matching flow is found, the switch performs the normal corresponding actions defined in the flow rule. If no matching flow is found, the switch accesses the memory to determine whether a matching SA of the host exists. If no matching SA exists, this means that the sending host (source) has not been authenticated for the packet. The switch then sends an illegitimate SYNACK packet with a cookie to the host (Fig. 3a). The cookie contains TCP/IP information from the SYN packet. In the TCP/IP protocol, the host that receives an illegitimate SYNACK packet must use the ACK sequence number of the SYNACK packet as the sequence number of the RST packet. Thus, in our method, we

set the cookie on the ACK sequence number field of the SYNACK packet. In the meantime, the SYNACK packet becomes illegitimate because of the wrong ACK sequence number.



(a) Auth-cookie generation stage and SA matching stage (b) SA creation stage (c) SA deletion stage

Fig. 3. Four stages of SD-OVS.

3.2 SA Creation Stage

In Stage 2, SD-OVS verifies the cookie to authenticate the host and creates an SA for the authenticated host (Fig. 3b). In the TCP/IP protocol, when the source receives an illegitimate SYNACK packet, it responds with an RST packet. This RST packet assigns ACK sequence number of the received packet as its sequence number to indicate the corresponding illegitimate packet. When SD-OVS receives the RST packet, it verifies whether the sequence number is the valid response for the cookie generated in Stage 1. If the cookie is legitimate, the switch creates an SA and stores it. An SA consists of the source IP address, destination IP address, and source port number. If the sequence number is invalid or has already been used to create an SA, the switch drops this RST packet.

3.3 SA Matching Stage

In Stage 3, when a corresponding SA for the retransmitted SYN packet exists, SD-OVS sends the extracted TCP/IP information of the SYN packet to the user space as the original OVS does (Fig. 3a).

The benign host resends the same SYN packet to the switch after the RST packet based on the retry mechanism in the TCP/IP protocol. At this time, SD-OVS first determines whether a matching flow is present in the flow table. For the resending benign host, SD-OVS finds a matching SA even though it finds no matching flow. This is because the host is authenticated in Stage 2. The SYN packet is then forwarded to the controller as the original OVS does.

Therefore, we can ensure that the benign packet can be processed correctly and that the controller only handles benign SYN packets for a TCP connection.

3.4 SA Deletion Stage

In the last stage, SD-OVS deletes the SA when it receives the last ACK packet in a TCP connection (Fig. 3c). When an ACK packet is present, SD-OVS searches for a matching SA in memory. If a matching SA is found, the switch deletes this SA and then forwards the ACK packet as the original OpenFlow switch does. The reason for this stage is twofold. First, it is for the purpose of memory management. The SA does not hold the memory space for a long time and has negligible effect on memory. Second, it is to prevent the attacker from exploiting the authenticated port after the benign host closes the TCP connection.

4 SD-OVS Implementation

We first describe our implementation environment and our extension of the original OVS. We then discuss the design of the cookie in detail and the specified management for SA.

4.1 SD-OVS Implementation

SD-OVS is implemented in the datapath of Open vSwitch 2.4.0. We extend OVS without any harm to its functionality, and no modification is required for the controller. We use Floodlight as our controller [12].

We add SD-OVS to the kernel space. Four stages are implemented after the "ovs_flow_tbl_lookup_stats" function and before the "upcall" function. When SD-OVS finds an SA, SD-OVS completes the required information for "upcall" and completes processes as the original OVS does. In Stage 4, after deleting the SA, SD-OVS calls "upcall" as the original OVS does.

4.2 Cookie Detail Design and SA Management

Cookie detail design. If we use the original SYN cookie generation method [10], the proposed method can be vulnerable to an RST flooding attack that consumes all SA tables. To prevent this, we must manage SYN cookies carefully.

The original SYN cookie consists of a minute-scale timestamp, MSS index, and a hash [13, 14] of the source IP/port, destination IP/port, timestamp, and security keys. Malicious attackers can reuse the timestamp and MSS index to generate an RST packet with a forged SYN cookie after finding security keys. To prevent a forged SYN cookie, we must make finding secret keys difficult. Thus, we change secret keys frequently and randomly to avoid an exhaustive search of keys by malicious attackers.

SYN cookie generation. SYN cookie is a 32-bit secure sequence number stored in the ACK sequence number field of a SYNACK packet. The original cookie is generated with the following formula as coded in FreeBSD syncookie.c. In the formula, COOKIEBITS is 24 and COOKIEMASK is (((__u32)1<<COOKIEBITS) – 1):

> "cookie_hash(saddr, daddr, sport, dport, 0, 0) + sseq + (count << COOKIEBITS) + ((cookie_hash(saddr, daddr, sport, dport, count, 1) + data) & COOKIEMASK))"

The system generates a 32-bit cookie. The top 8 bits correspond to a minute-scale cookie timestamp (count<<COOKIEBITS) and the remaining 24 bits are "data" plus cookie-hash of source-destination information with a timestamp and secret key index of 1. The "data" is the index of MSS (MSSIND). That value is then included with the overlapping hash value and "sseq." The "cookie_hash(saddr, daddr, sport, dport, 0, 0)" is the overlapping hash value of source-destination information with a setting timestamp of 0 and a secret key index of 0. The "sseq" is the original source-side packet sequence number. Because we are sending an RST packet with an illegitimate sequence number, "sseq" is meaningless in our scheme.

In the cookie hash function, a secret key indexed by 0 or 1 is added. These secret keys (only two 32-bit values) are crucial and maintained only in the switch. Although an attacker can find these keys after an exhaustive search, a considerable amount of time is required. However, to increase their secrecy, we change these keys frequently and randomly before an attacker finds them. Thus, an attacker is unable to generate packets easily using a forged SYN cookie by reusing information in a valid SYN cookie.

SA management. Because the SA table is another resource of a switch, it should be managed properly to prevent the switch from denying incoming connection requests. The entry in an SA table is generated by receipt of an RST packet with an SYN cookie. The corresponding entry is then deleted after receipt of the last ACK packet from the source. If SD-OVS does not receive a corresponding ACK packet necessary to delete the SA, we call this SA an "incomplete SA." An incomplete SA is removed after a timeout to prevent an attacker from first sending a valid RST packet and then exploiting the existing SA later in launching attacks.

5 Evaluation

In this section, we describe our evaluation results and show that SD-OVS protects the SDN efficiently at an acceptable time cost.

We evaluated SD-OVS on the same computer by using a virtual network namespace. First, we tested SD-OVS with a benign host and evaluated the amount of time consumed. Second, we evaluated the performance of SD-OVS and the original Open vSwitch during an SYN flooding attack.

5.1 SD-OVS for Benign Host and Time Cost

For the benign host, we tested whether SD-OVS can function well and evaluated the amount of time consumed in sending the illegitimate SYNACK packet and creating the SA in Stages 1 and 2.

We used a network namespace to create a virtual network topology to be evaluated (Fig. 4). The h1 represents a receiving host and h2 a benign sending host. The switch is connected with the Floodlight controller. Without considering the amount of time to transmit a packet in the network, we evaluated the amount of time consumed for one TCP connection in Stage 3.

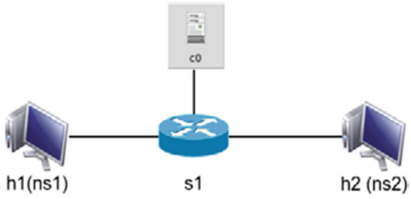


Fig. 4. The virtual network topology used in Sect. 5.1

The benign host (h2) attempts to make a TCP connection with the receiving host (h1). Detailed packet information is captured by Wireshark (Fig. 5). The top circle represents Stages 1 and 2, and the second circle represents Stage 3.

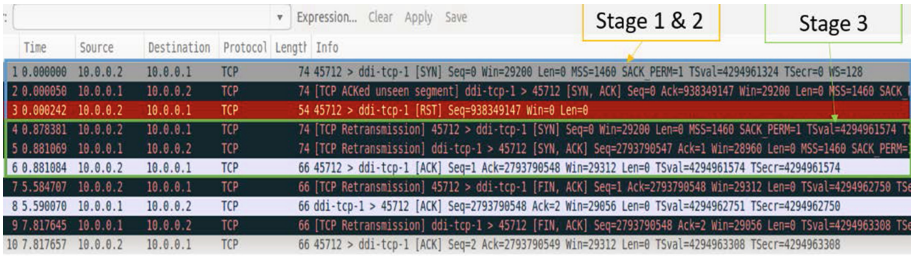


Fig. 5. Packet information when using SD-OVS.

In the top circle, the ACK sequence number of the second packet (the illegitimate SYNACK packet) is the cookie value generated in Stage 1. The sequence number of the third packet (RST packet) sent from the benign host is the same as the sequence number of the SYNACK packet. In this case, the benign host can verify the cookie successfully and create an SA during Stage 2. The second circle shows that the TCP connection is established in Stage 3.

The total amount of time consumed for Stages 1 and 2 is 0.242 ms, and the time for Stage 3 is 2.703 ms. Because the retransmission time gap could be set differently, we evaluated it based on the default time in Linux. The sixth packet shows that the amount of time consumed for one TCP connection is less than 1 s (Fig. 5), which is acceptable.

We recommend setting the transmission time gap to a small value when using SD-OVS to produce better performance.

5.2 SD-OVS Under SYN Flooding Attack

We evaluated the performance for both SD-OVS and the original Open vSwitch during an SYN flooding attack.

A virtual network namespace was used for evaluation. We assumed that the attacker could not compromise the switch but had the ability to compromise the host. The h3 represents the compromised host and, in our experiment, launched the SYN flooding attack (Fig. 6).

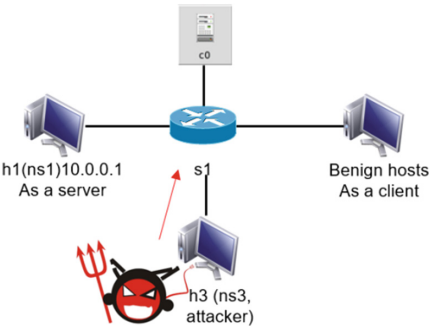


Fig. 6. Virtual network topology used under attack.

When the attack rate was greater than 60 PPS, the original Open vSwitch could no longer respond to the benign new SYN packets. By contrast, the controller connected to SD-OVS could always respond to the new SYN packets with an acceptable time cost (Fig. 7). The time cost incurred during the attack was still less than 1 s (which included the retransmission time gap). This shows that SD-OVS protects the controller at an acceptable time cost (Table 1).

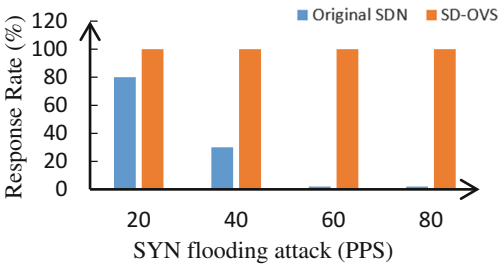


Fig. 7. Response rate compared with original OVS under SYN flooding attack.

Table 1. Time cost comparison for one TCP connection under SYN flooding attack

	TCP connection time (s)
Original SDN	∞
SD-OVS	0.92652

6 Other Possible Related Attacks

In this section, we discuss solutions for some possible attacks that bypass SD-OVS.

First, we defined the incomplete SA in Sect. 4.2 and the management rules for the incomplete SA to prevent the attacker from exploiting the SA by sending a valid RST packet. If the attacker compromises one host and sends an RST with a valid cookie, SD-OVS creates the SA. The attacker then launches an SYN flooding attack instead of following the proposed connection procedure. The malicious SYN packet can complete Stage 3 because of an existing SA. With the proposed timeout scheme for incomplete SA management, SD-OVS deletes the incomplete SA automatically even though the attacker does not send the ACK packet. When the attacker tries to launch the SYN flooding attack, the SYN packet must repeat Stages 1 to 4. Therefore, the SYN flooding attack is not successful.

Second, the Linux kernel can allow a local attacker to obtain sensitive information because of the imperfect secret random value given by the "net_get_random_once" function. Thus, SD-OVS must refresh the secret value frequently to prevent the attacker from acquiring the values, which is most important when we generate the cookie.

Third, if an attacker launches an RST attack by resending the RST packet with valid cookies repeatedly, the switch may create multiple copies of the same SA, which may exhaust the memory. We consider that SD-OVS should check whether the same SA exists before creating a new SA to avoid exhausting memory.

Finally, in the original SYN cookie method, the destination host had an unstable connection when the last ACK packet from the source was lost [10, 11]. The source assumed a connection was established. However, the destination did not have a connection established. In the proposed scheme, TCP handshaking between the source and destination occurs normally without using an SYN cookie. The SYN cookie in our method is used only between the source and ingress switch. The destination receives the SYN packet, which is already authenticated by the SD-OVS switch. The destination stores the information in its backlog. If any packet is lost during the connection establishment, it triggers the retry mechanism as in the original TCP handshaking protocol.

7 Related Work

Our study was inspired by some previous studies that describe the vulnerabilities in SDN [5, 6, 15, 16] and AVANT-GUARD [7]. These studies were the first to attempt to modify the OpenFlow switch and use the SYN cookie method in SDN.

AVANT-GUARD has two shortcomings. First, it must modify both the OpenFlow switch and controller. AVANT-GUARD has two important stages: migration and relay.

These stages are commanded by the controller, and the switch must send a special message, which is defined by AVANT-GUARD for the controller, which is necessary to obtain the command from the controller. Because many different types of controllers exist, AVANT-GUARD requires that all these controllers be modified. Our proposed method does not have this drawback because the SD-OVS switch alone protects against a SYN flooding attack. Thus, our method can provide better scalability. Second, AVANT-GUARD must modify the sequence numbers of TCP packets in an established connection because of the connection relay. AVANT-GUARD must manage two TCP connections as a single connection. Therefore, it must store the value difference between two SEQ numbers of packets as well as change the SEQ number of each packet. Our scheme can avoid this added burden because SD-OVS utilizes the retry mechanism in the TCP protocol. SD-OVS neither needs to manage two TCP connections nor maintain any value to change the packets.

For the same goal of protecting the controller from an SYN flooding attack, SD-OVS is easier to apply with minimal modification to the OpenFlow switch and no modification to the controller. SD-OVS also has a simpler workflow without managing any TCP connection or informing the controller.

8 Conclusion

In this paper, we proposed a SDN switch called SD-OVS to defend against SYN flooding attacks. The mechanism to block malicious SYN packets to protect controller is a host authentication with SYN cookie and TCP retry. We showed a method of establishing a host authentication for an incoming SYN packet and using it for a second attempt with the same SYN packet. Our evaluation results showed that SD-OVS can protect a controller from an SYN flooding attack at an acceptable time cost. The proposed scheme operates effectively against an SYN flooding attack with minimal change to the SDN switch and no modification to the SDN controller.

Acknowledgements. This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-TB1403-04.

References

1. McKeown, N., et. al.: OpenFlow: enabling innovation in campus networks. In: ACM SIGCOMM Computer Communications Review, New York, pp. 69–74 (2008)
2. CERT, TCP SYN Flooding and IP Spoofing Attacks (1996)
3. Eddy, W.: TCP SYN flooding attacks and common mitigations. In: Request for Comments: 4987, pp. 6–10 (2007)
4. Gavaskar, S., Surendiran, R., Ramaraj, E.: Three counter defense mechanism for TCP SYN flooding attacks. Int. J. Comput. Appl. **6**(6), 12–15 (2010)
5. Hong, S., Xu, L., Wang, H., Gu, G.: Poisoning network visibility in software-defined networks: new attacks and countermeasures. In: NDSS Symposium on 2015, San Diego (2015)

6. Braga, R., Mota, E., Passito, A.: Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: IEEE 35th Conference on Local Computer Networks (LCN), pp. 408–415. IEEE, Denver (2010)
7. Shin, S., Yegneswaran, V., Porras, P., Gu, G.: AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In: ACM SIGSAC Conference on Computer and Communications Security, CCS 2013, New York, pp. 413–424 (2013)
8. OpenFlow Switch Specification. http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf
9. Open vSwitch. http://openvswitch.org/
10. Bernstein, D.J.: SYN cookie, September 1996
11. Kim, T., Choi, Y., Kim, J., Hong, S.: Annulling SYN flooding attacks with whitelist. In: 22nd International Conference on Advanced Information Networking and Applications, Okinawa, pp. 371–376 (2008)
12. Floodlight. http://www.projectfloodlight.org/floodlight/
13. Goldreich, O.: Foundations of Cryptography: Volume II Basic Applications, pp. 498–502. Cambridge University Press, New York (2004)
14. PUB, FIPS, Secure Hash Standard. FIPS PUB 180-4, pp. 18–20 (2012)
15. Scott-Hayward, S., O'Callaghan, G., Sezer, S.: SDN security: a survey. In: Future Networks and Services (SDN4FNS), pp. 1–7. IEEE, Trento (2013)
16. Kloti, R., Kotronis, V., Smith, P.: OpenFlow: a security analysis. In: 21st IEEE International Conference on Network Protocols, pp. 1–6. IEEE, Goettingen (2013)