Advance Access publication on 15 November 2018
doi:10.1093/comjnl/bxy117

SDN-Based SYN Proxy—A Solution to Enhance Performance of Attack Mitigation Under TCP SYN Flood

Dang Van Tuyen[1], Truong Thu Huong[1]*, Nguyen Huu Thanh[1], Pham Ngoc Nam[1], Nguyen Ngoc Thanh[1] and Alan Marshall[2]

[1]*School of Electronics & Telecommunications, Hanoi University of Science and Technology, Hanoi, Vietnam*
[2]*Department of Electrical Engineering and Electronics, University of Liverpool, Liverpool, UK*
[*]*Corresponding author: huong.truongthu@hust.edu.vn*

Recently, TCP SYN flood has been the most common and serious type of Distributed Denial of Service attack that causes outages of server resource of Internet Service Providers. In another aspect, Software Defined Networking (SDN) has emerged as a new networking paradigm to increase network agility and programmability. SDN is also a promising architecture to deal with the network security issue where we can flexibly change security rules and control incoming flows. In this article, we design an Openflow/SDN network remedy to combat specifically TCP SYN flood. We show security threats for the SDN architecture and exploit SDN capabilities and features to design a SDN-based SYN Proxy (SSP) paradigm to mitigate such TCP SYN threats. Our SSP is proved to be a network-based solution to protect application servers in terms of decreasing number of Half-Open Connections at an application server and increasing probability of successful establishment for a TCP flow connection under TCP SYN Flood attack. Using SSP to support application servers is shown to outperform the case where the servers adopt only the protection scheme of Microsoft Windows server reference model without utilizing SSP. SSP also shows that it can reduce the time a flow entry occupies the switch resource by 94% in comparison with the Avant-Guard solution. In addition, SSP improves the successful connection rate and average connection retrieval time in comparison with the standard Openflow solution.

Keywords: OpenFlow; SDN; DDoS attack; TCP SYN flood; SYN Proxy

Received 22 February 2018; revised 6 May 2018; editorial decision 11 June 2018
Handling Editor: Steven Furnell

1. INTRODUCTION

Denial of Service (DoS) and Distributed DoS (DDoS) [1, 2] attacks continue to be complex and unpredictable, making them more challenging for companies to mitigate. According to Verisign [3], the highest intensity flood attack observed in Q3 2016 was a TCP SYN flood that peaked at approximately 60 Gbps and 150 Mbps. This flood attack is one of the highest packets per second attacks ever observed by Verisign. TCP SYN flood [4] is a type of DoS attack that relies on abusing the TCP three-way handshake [5] of a TCP connection establishment process in order to consume resources on the targeted server and render it unresponsive. Typically, a client sends a SYN packet to an open port on a server asking for a TCP connection. The server then acknowledges the connection by sending a SYN-ACK packet back to the client, storing the connection information in a transmission control block (TCB) and maintaining a state called half-open connection (HOC). The client then responds to the server with an ACK packet to establish the connection. This process is commonly known as a 'three-way handshake'. During a SYN flood attack, the server is bombarded with huge amount of SYN requests often using spoofed source IP addresses. The server, being unaware of the attack, creates a lot of HOCs and the machine's resources are quickly exhausted due to occupation by useless TCBs before these HOCs can time out. The resource exhaustion not only prevents the target machine from serving legitimate connections from a benign Client but also has the following negative effects: (i) occupying the

server resource, (ii) making flow-state-based security schemes ineffective since managing useless flows (i.e. flows of having no second packet until timeout) and (iii) searching in TCB table will take longer. Moreover, the victim will not be able to identify which SYN packets belong to legitimate or malicious connections. Based on the aforementioned facts, TCP SYN Flood is one of the most difficult attack types to be prevented and defensed. Various defense schemes have been proposed to mitigate its impact on systems [6, 7].

In the aspect of new networking paradigm, software-defined networking (SDN) [8] is a new computer networking approach that decouples the control plane, which modifies the behavior of network devices from the data plane (underlying system), which forwards traffic to the selected destinations. This allows network administrators to manage network services through abstraction of higher-level functionality more flexibly than the traditional network architecture. Within this context, Openflow [9] is considered one of the first SDN standard that defines a communication protocol to directly access to the forwarding layers of network devices such as switches or routers in a SDN architecture. Openflow is a potential candidate for future large-scale network deployment. In fact, SDN was born to bring the flexible networking capability for the system in order to improve the system performance. It then becomes a promising paradigm for the domain of network security as well. However, besides its advantages of a flexible networking paradigm and flexible capacity of network security control, from the network security perspective, Openflow protocol still exposes some weakness, for example flow entries remaining in any Openflow Switch (OFS) based on the time out mechanism cause wasted resource and this property makes OFS a new target for DDoS attacks. To apply the SDN mechanisms, applications of network management and security install more and more flow entries into OFSs. Therefore, when a SDN system is under attack such as TCP SYN Flood, the number of flow entries may increase beyond the capacity and the OFS itself can become a victim [10].

Although there have been several SDN-based schemes proposed to prevent TCP SYN flood such as Avant-Guard [11], LineSwitch [12], these solutions basically add on a packet-processing module at OFS. That fact makes the OFS lose its originally-designed nature in which the switch works just as the data plane of forwarding data without packet processing.

In this article, we design a SDN-based SYN Proxy framework (**SSP**) to mitigate TCP SYN flood attack in a SDN-based network. SSP is simply based on the Openflow mechanisms including: capability of matching to TCP Flags fields and pipeline processing. Therefore, different from Avant-Guard or LineSwitch, SSP can be applied to all Openflow-supported switches without requirement of any modification since SSP requires to build no additional hardware module in the currently-standard OFS. In fact, SSP combines the OFS and controller to work as a stateless SYN Proxy. SSP monitors the TCP three-way handshake process only, not the whole course of TCP connections. With SSP, the lifetime of flow entries are remarkably decreased. Moreover, SSP introduces a method to adaptively adjust the time outs of flow entries according to the number of HOCs at the server. In this way, flow entries are removed much quicker during TCP SYN Floods.

Our performance measurement shows that during the normal traffic period, applying SSP can dramatically reduce the lifetime during which a flow entry occupies an OFS's resource by up to 94% in comparison with Avant-Guard [11]. This not only increases the packet-processing performance of the OFS but also makes the switch more endurable during TCP SYN floods. SSP is shown to be an appropriate approach to replace the current solutions that are used in application servers, such as Microsoft Windows server [13]. One single SSP deployment can protect all application servers within the SDN-based network, we do not need to install the protection scheme at every single server. SSP is proved to decrease the number of HOCs at an application server by 86% in comparison with the case the application servers deploying only the common protection approach of Microsoft Windows Servers, without utilizing SSP in the network, provided the attack rate of 500 SYN packets per second.

The rest of the article is structured as follows: Section 2 describes the related work. Section 3 introduces some background in the research field. Section 4 analyzes the characteristics of the common TCP traffic. In Section 5, we describe our SSP design and implementation in detail. The SSP testing environment is described in Section 6 and the performance is analyzed in Section 7. And finally, the conclusion is presented in Section 8.

2. RELATED WORK

DoS attacks are grouped into the three following categories [6, 7]: Direct, Spoofed-based and Distributed attack; the source IP address spoofing and distributed attack are the most difficult to be detected and prevented. Different from other attacks, TCP SYN flood is mainly based on spoofing source IP address and only needs a low traffic rate but can overwhelm a server in a short time. Due to its simplicity, effectiveness and ease of attack, TCP SYN flood has been currently receiving a great concern from many research groups around the world.

In the traditional networking, the solutions proposed for TCP SYN flood can be found in two manners: host-based and network-based solutions. Host-based solutions focus on hardening the end-host TCP implementation at the server side, including limiting the number of HOCs such as rate limiting [14, 15], increasing TCP backlog [7] or altering the algorithms and data structures used for connection lookup and establishment such as SYN cache [16], SYN Cookie

[17], Reducing the SYN-RECEIVED timer [13]. The common characteristics of these solutions are that they do not differentiate legitimate and malicious traffic, but are gadgets to improve tolerance capacity of the server during attack. Some solutions actually consume computing resource of servers such as SYN cookie [17], increasing TCP backlog [7]. The network-based solutions address security with two main techniques: Filtering in the Firewall and SYN proxy. Most commercial firewall solutions such as [18, 19] filters TCP SYN flood attacks, preventing SYN packets coming from suspected sources to head to a server. The SYN proxy solution [20] is implemented in an intermediate device to monitor the three-way handshake process of TCP connections between a client and a server. The major drawbacks of this solution are that it needs a robust system capacity and it splits the TCP connections during the three-way handshake monitoring process.

To date, the ways to deploy the SDN architecture to detect and mitigate DDoS attack can be found in various proposals such as [21–26]. The common idea of those approaches is to run a security application in the SDN controller, that installs flow entries in the SDN switch to retrieve traffic-related parameters of flows. These flow entries then periodically send traffic information to the controller for analysis and attack detection. Upon detecting attack traffic, the controller installs flow entries at the switch to stop suspicious flows. The problem arising when SDN is applied to mitigate DDoS attack is: malicious flows usually arrive alongside legitimate flows. So to detect and mitigate attack effectively, there is a need to control each individual flow; and the controller has to install a flow entry corresponding to each incoming flow in the switch. This operation dramatically increases the number of flow entries installed in the switch. On another side, when a flow has terminated, the corresponding flow entry is not deleted right afterwards but remains in the switch until the flow entry timeout. This causes wasted OFS resources; and attackers may make use of this vulnerability to consume the SDN switches' resources. Among the SDN-based security approaches, specifically coping with TCP SYN flood, the authors in Avant-Guard [11], LineSwitch [12] propose the schemes that make OFS act like a SYN Proxy. In these solutions, a SDN switch controls TCP flows from the Internet and allow only flows connecting directly to servers after the three-way handshaking procedure has completed. However, Avant-Guard has its own weakness: it exploits SYN cookies in combination with a 'stateful SYN proxy' in the switch that requires high processing capacity from the SDN switch. During TCP SYN floods, the switch resource will become exhausted quickly due to:

- SYN cookie uses a complex algorithm for hashing, encoding and packet processing.
- Due to the split of TCP connections between the client and server, for each TCP connection, the OFS has to maintain a memory block to store and handle sequence number and acknowledgment number of packets exchanged between two parties. Furthermore, each flow entry created corresponding to a TCP flow will dramatically increase the number of flow entries in the switch, thereby affecting the searching time during the matching process; or it may overwhelm the memory when TCP SYN floods occur. LineSwitch provides an improved solution for Avant-Guard in which it monitors only the first SYN packet of each source IP address, thus preventing port scan attacks or repeated use of forge IP addresses. However, during attack, hackers usually make use of random source IP addresses, so that the probability of duplicate IP addresses is actually very low. LineSwitch has not provided a solution to solve the problem of maintaining states in the OFS.

In this article, we propose SSP that combines the OFS and SDN controller to work as a stateless SYN Proxy. In SSP, the OFS does not need to maintain the states of TCP flows after the TCP three-way handshake process has successfully completed. This means the flow entries created inside the switch to serve the three-way handshake process will be deleted immediately after the process has successfully completed. The advantage of SSP is that we can deploy SSP in any Openflow-supported switch without further modification or hardware add-on. This method helps to reduce the average lifetime of flow entries that occupy the switch resource by 94% in comparison with the other SDN-based SYN Proxy solutions Avant-Guard [11], LineSwitch [12]. Our proposed scheme can be considered a method to solve the aforementioned common weak point of SDN-based security scheme: the saturation of the OFS resource.

Moreover, as described in Section 1, the main goal of TCP SYN flood is to make a target machine to open many HOCs until all of the machine's resource are consumed. Traditionally, a popular mitigation method is to monitor for such situations and close useless HOCs right at each application server. A typical scheme like that can be found in a well-known Windows Server. According to a report in 2015, as commercial products, operating systems 'Windows Server' of Microsoft use the function 'TCP SYN attack protection' [13] based on the combination of Rate Limiting techniques [14, 15] and Reducing the SYN-RECEIVED timer. When the number of HOCs and HOCs with retransmitted SYN ACKs exceeds a predefined threshold (**TcpMaxHalfOpen** and **TcpMaxHalfOpenRetried**), the SYN protection mode is activated and the OS will reduce the number of times to retransmit SYN-ACK, which normally is equal to **TcpMaxConnectResponseRetransmissions**, before discarding the HOCs, and not answer any incoming SYN packets. The weak point of this solution is that during the SYN protection mode, all SYN packets going to a server will be discarded no matter if the sources are malicious or legitimate.

In this article, SSP is exploited as a centralized scheme and forefront component to protect all servers within its network. Provided with a SYN packet rate of 500 SYN packets per second, SSP can reduce 86% of the HOCs at a victim server in comparison with the performance of a network that deploys only the traditional approach at a victim server like Windows server.

3. BACKGROUND

3.1. SYN proxy

SYN Proxy is a network-based solution for detecting and mitigating TCP SYN Flood. It is an intermediate device on the network that verifies the three-way handshake process of TCP connections. If this process is successful, the connections between the client and server for data exchange will remain. Conversely, if the process is not verified, the proxy prevents establishing connections to the server or removes HOCs created on the server to free up the server's resource for other benign connections. Based on the packet processing mechanisms of the three-way handshake process, there are two types of SYN proxy: SYN-ACK spoofing and ACK spoofing. Figure 1 describes the operation of these two types.

Upon receiving a TCP SYN request from a client, SYN-ACK Spoofing Proxy does not forward the SYN packet to the server but create a spoofed SYN-ACK packet to respond to the client with a random ISN (Initial Sequence Number) value. If within a predefined time out period, the proxy does not receive an ACK packet from the client to validate the three-way handshake process or the proxy receives an invalid ACK packet (i.e. Acknowledgment number is different from ISN +1), the TCP connection is considered abnormal and discarded. If a valid ACK packet is returned, the proxy creates a spoofed SYN and a spoofed ACK packet to forge the client in order to establish a TCP connection to the server. When the connection is successful, the proxy plays the role

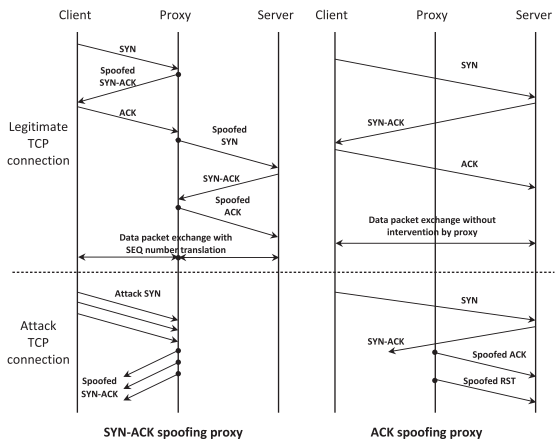

FIGURE 1. The operation of the two types of SYN Proxy.

of an intermediate device to process and forward packets of the two TCP connections between the client and the server. The advantage of the SYN-ACK Spoofing proxy is that IP spoofed TCP connections are prevented by the proxy from attacking directly to the server. However, this solution has a downside in the fact that the proxy must spoof many packets and maintain the connection states of the two TCP connections between the proxy and the client and server for translating the pairs of Acknowledgement/Sequence Number (SEQ). This leads to a risk that the proxy itself becomes a victim of a TCP SYN flood attack.

ACK Spoofing Proxy handles the TCP connection in a different way, it passes the SYN request directly from the client to the server and waits for the ACK response packet from the client after sending the SYN-ACK response of the server.

If during a predefined timeout period, the proxy does not receive a valid ACK packet to complete the three-way handshake process, the proxy generates a spoofed ACK packet sent to the server to complete the three-way handshake process. Soon after, a spoofed RST packet is also created and sent to the server to close the TCP connection. The advantage of this approach is that there is no need to maintain a state on the proxy during the exchange of data between the client and the server. The downside of this solution is that attack traffic still reaches the server directly. If the proxy does not remove malicious flows on time, malicious TCP connections will occupy the backlog and prevent other benign connections to the server.

3.2. Pipeline machanism of packet processing in SDN Openflow

Openflow is one of the first standard schemes that has been broadly researched for the SDN architecture. Openflow defines the communication protocol between the data plane and control plane. The protocol also introduces a standard for processing incoming packets at the data plane such as OFS. Packets which go through OFS are categorized into flows based on header fields of packets; and are identified, processed by flow entries that are installed in the flow tables of OFS. Each packet going through the switch is matched to a flow entry and ruled by the instruction defined in that flow entry. If the packet can not be matched to any available flow entry, OFS creates a packet-in event to demand the controller to install a new flow entry which is applied to process the following packet. Flow entries can be proactively installed by the controller (e.g. proactive flow entry) or installed on demand of a packet-in event (e.g. reactive flow entry).

In general, an OFS can work under a pipeline mechanism [27] in which the OFS contains multiple flow tables indexed from 0 to N. Each flow table comprises multiple flow entries. The pipeline processing starts at the first flow table where incoming packets are matching to every flow entry in order of

priorities. If a flow entry is matched, the incoming packet will be processed in accordant with the instructions of the flow entry. The packet may be forwarded to a specific port, sent to the controller or continued to be matched to flow entries of other subsequent flow tables. Also, each flow table has a Table-miss flow entry (which is the fourth flow entry in the flow table, called FE4) to process packets which are unmatched to any flow entries in the flow tables. The instructions of the table-miss flow entry can send packets to the controller, to drop packets or to direct packets to continue being matched in a subsequent table.

Packet processing, based on pipelining with multiple flow tables, allows packet processing rules to be based on the header fields of packets, the priorities of flow entries, and also on other conditions which are created by arranging the flow entries in the flow tables. If the packet processing at OFS is considered as the implementation process of a sequential computer program, arrangement of flow entries in a single flow table will allow the switch to process incoming packets as a SELECT CASE instruction according to the cases that are arranged in the priority order (Fig. 2). Therein each case is a corresponding criterion described by matched fields in each flow entry. Arrangement of multiple flow tables allows the ruling of packet processing more flexible as a nesting SELECT CASE or IF instruction (Fig. 3). This makes the packet processing faster and more flexible. Besides, the pipeline mechanism also allows accumulating multiple instructions from matched flow entries in flow tables in which the process passes through.

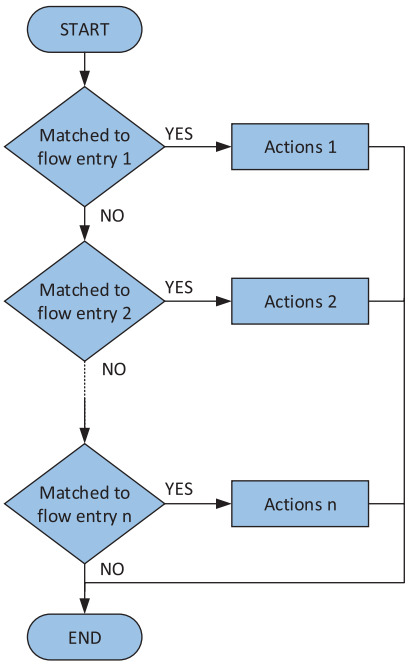

FIGURE 2. Flow chart of the ruling of packet processing at OFS in the case of using one flow table.

4. TRAFFIC ANALYSIS

As stated above, in SSP, we design the switch-controller to follow the TCP three-way handshake process at the switch so as to remove attack HOCs at the victim server as fast as we can. In order to control the three-way handshake process, to detect TCP SYN flood, we need to know the typical timeout period of a common flow coming from the Internet. We analyze the traffic of 100 servers being selected randomly and 3 947 883 flows from the CAIDA 2013 dataset [28]—the most popular Internet log file to investigate characteristics of incoming non TCP-SYN-attack flows. We measure the waiting time from the point the Openflow Proxy receives a TCP SYN packet to the point it receives the first ACK packet to finish the three-way handshaking procedure. The statistics of that waiting time are illustrated in Fig. 4.

As Fig. 4 shows, 99% of legitimate TCP flows has the three-way handshake processing period smaller than 2 seconds. It means nearly 99% of the present flows have ACK packets that return within 2 seconds. It gives us the value of an appropriate time, we should set for timer of HOCs created at application servers. In fact, when an attack TCP SYN connection arrives, after answering a SYN-ACK packet, the destination server waits for the ACK packet that actually never comes. Under that circumstance, the server will wait until

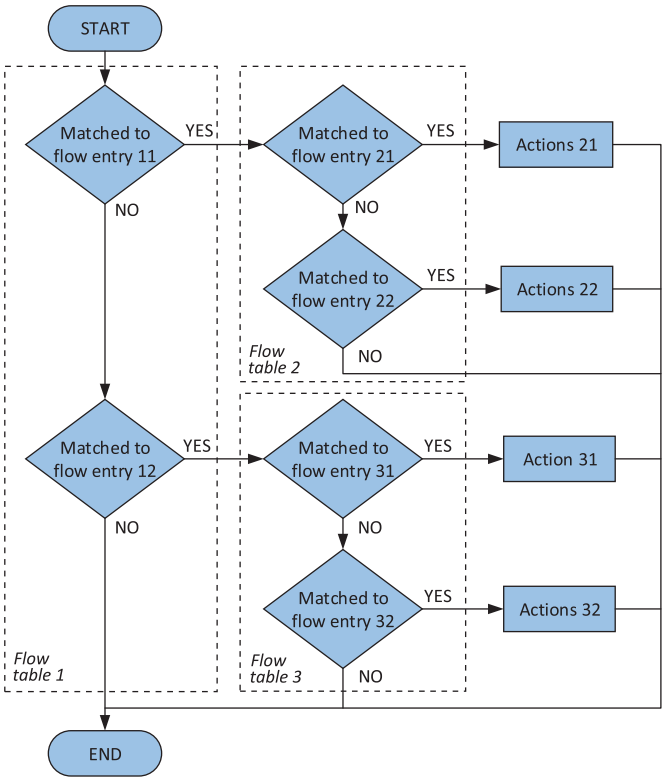

FIGURE 3. A flow chart example of the ruling of packet processing at OFS in the case of using multiple flow tables.

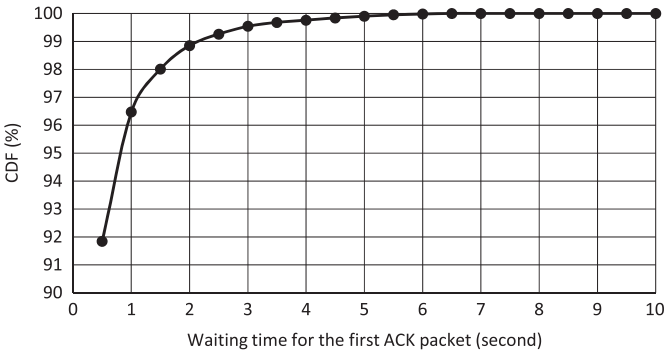

FIGURE 4. CDF of flows having different waiting time for the first ACK packet in TCP flows.

timeout to delete the HOC. The current solution for Windows Server calculates this time out period by waiting three transmissions of SYN-ACK packets that takes 21 seconds (the first transmission after 3 seconds, the second one after 6 seconds, and the third time after 12 seconds). This time out actually should not be a fixed parameter but an adjusted value in order to delete HOCs faster, thereby freeing up the server resource especially during attack.

5. SDN-BASED SYN PROXY ARCHITECTURE AND OPERATION

In this work, we build a so-called SSP (SDN-based SYN Proxy) framework that is intended for edge OFS of a network controlled and demanded by the remote SDN controller to protect application servers from TCP SYN flood attacks (Fig. 5). This work is inherited from our previous proposal [25] in which we proposed a general SDN architecture and a control algorithm to protect application servers within an enterprise network. In [25], we more focused in designing and developing a system that has the capacity of analyzing incoming traffic statistically to find out attack patterns for multiple types of attacks. However, the statistical analysis phase may require a long enough period of data collection in order to acquire accurate analysis. Therefore, in this work, we propose another framework to make the SDN controller and OFS to co-work as ACK Spoofing SYN Proxy which can specifically detect TCP SYN attacks in a shorter time. This new framework is called SDN-based SYN Proxy SSP.

In this SSP framework, the SYN Proxy Module (SPM) is a developed software that configures OFS, arranges multiple flow tables and installs flow entries with predefined instructions at OFS to capture packets during the three-way handshake process. In parallel, the capture process at OFS works with the SPM module in the controller to monitor this three-way handshake process. Note: to make this SSP framework to work, we need commercial Openflow switches with

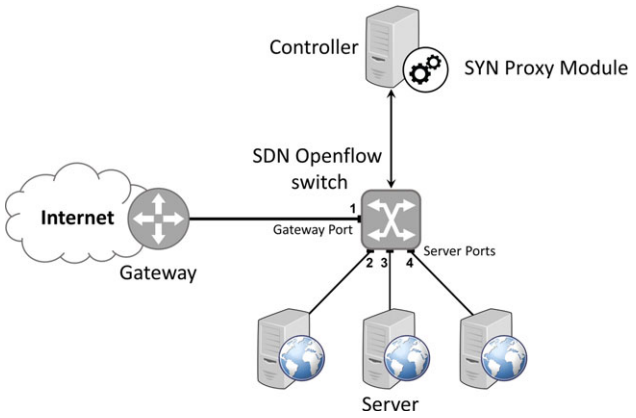

FIGURE 5. SSP system architecture.

Openflow version from 1.5 and later since availability of TCP-Flag-field matching capability and multiple table facility. In the architecture described in Fig. 5, we can see that:

- OFS is used at the edge network that connects directly to protected application servers.
- Traffic coming from the Internet to the servers passes through a gateway port defined in OFS.
- Packet processing policies at OFS are issued by the controller via the secured Openflow protocol.
- The three-way handshake process of TCP connections from the Internet is monitored and validated by the SPM.

The whole framework configures the OFS and SPM modules to work as ACK Spoofing SYN Proxy, enabling the system to quickly detect a harmful TCP SYN request, thereby freeing resource of the switch (i.e. deleting flow entries at the switch flow tables) and resource of application servers (i.e. closing HOCs) as fast as possible. The general principle of the packet processing in SSP is described in Fig. 6.

SSP exploits the pipeline processing of Openflow in which instructions allow packets to be sent to subsequent tables for further processing. SSP installs flow entries with appropriate matching fields, and particular priorities to capture packets exchanging between the client and server during the three-way handshake process (Steps 1–7 in Fig. 6). The SPM module running on the controller validates TCP connections by checking the pair of sequence number and acknowledgment number and issues policies of packet processing for the TCP connections (Steps 7 and 8):

- If the connection is legitimate, the TCP connection is migrated and the following packets of the connection are directly exchanged between the client and server without intervention by the SYN proxy.
- If the connection is regarded malicious, the switch sends a spoof ACK and RST to close the HOC on the server.

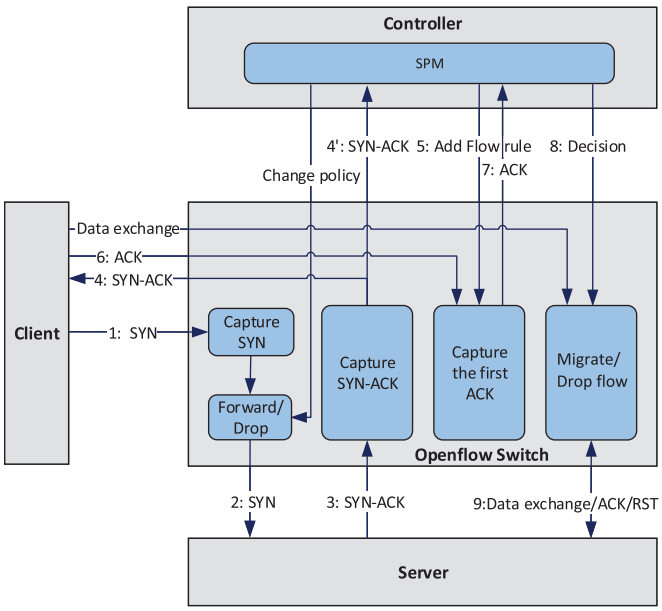

FIGURE 6. General principle of SSP.

In the following subsections, we will elaborate the structure and mechanism of packet processing at the OFS and SPM running at the SDN controller.

5.1. Capturing three-way handshake packets by arrangement of flow entry tables at OFS

The packet processing in the three-way handshake process of a TCP connection at OFS is described in Fig. 7.

In order to implement the aforementioned process, and to make use of the pipeline processing, in SSP, we arrange flow entries and their priorities in the flow tables as described in Table 1.

∗ *Flow Table 0*:

Contains proactive flow entries to capture SYN, SYN-ACK packets of the three-way handshake process:

- Flow entries FE1x are used to capture SYN packets from an Internet client. If the system is under the threshold of normal operation, SYN packets are forwarded to the corresponding server. When beyond the system capacity, SYN packets are dropped. The policy to decide Forward or Drop incoming SYN packets depends on the system status and described in detail in Subsection 5.4.
- Flow-entry FE2 is used to capture SYN-ACK packets responding from servers to Internet clients. If a SYN-ACK packet is received from a server, it will be forwarded to the controller so as to inform the controller to follow up the three-way handshaking process initiated by the server before sending to the client.

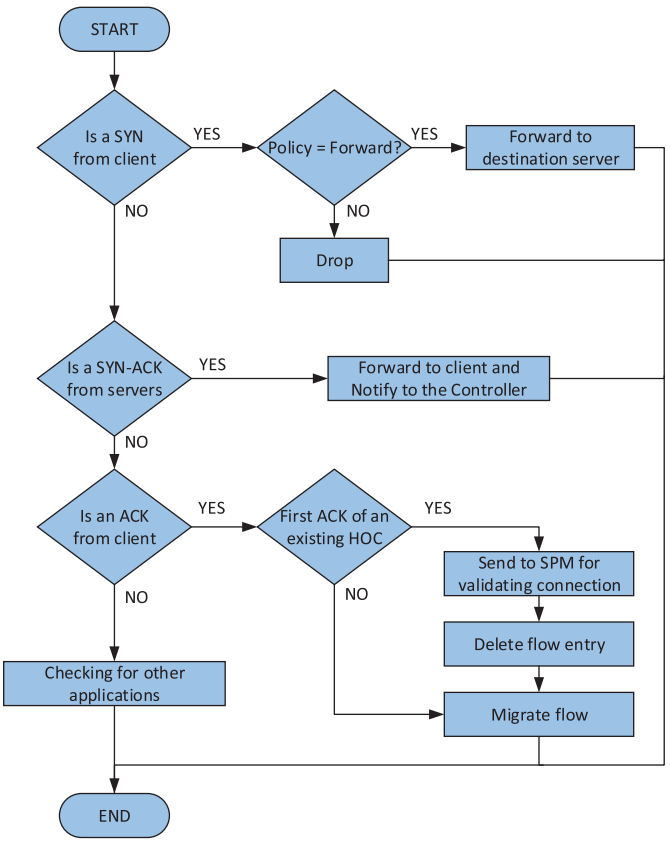

FIGURE 7. Flow chart of packet processing in the three-way handshake process at OFS.

- Flow-entry FE3 is used to direct ACK packets coming from the Internet to *Flow Table* 1. Other incoming packets that differ from the three-way handshake packets will be forwarded to *Flow Table* 2 via the flow entry FE4—the 'table-miss' flow entry.

∗ *Flow Table 1*:

FE5x are reactive flow entries that are installed by the controller for each TCP connection. FE5x are to capture the first ACK packet sent from a client to complete the three-way handshake process. If the ACK packet is return from the client, its corresponding flow entry FE5x forwards it to the SPM module to validate the TCP connection. Concurrently, the flow entry (FE5x) is deleted to free the switch resource.

In each flow entry FE5, a timeout t is set in accordance with the waiting time of the first ACK packet which is calculated from the point the switch forwards the server's SYN-ACK packet to the Internet client until the point it receives the ACK packet (t is calculated by the controller and will be described in Section 5.3). If the switch does not receive an ACK packet after period t, it understands that the connection is likely to be abnormal. Then the flow entry is timed out and this Timeout event is informed to the SDN controller to discard the corresponding HOC at the

TABLE 1. Structure and arrangement of flow entries in SSP flow tables.

Flow Table 0	
FE11	Match fields = {Inport = 1; Type = TCP; Flags = SYN; dstIP = IPd1}; Timeouts = 0; Instructions = {Forward to port 2/Drop}
FE12	Match fields = {Inport = 1; Type = TCP; Flags = SYN; dstIP = IPd2}; Timeouts = 0; Instructions = {Forward to port 3/Drop}
FE13	Match fields = {Inport = 1; Type = TCP; Flags = SYN; dstIP = IPd3}; Timeouts = 0; Instructions = {Forward to port 4/Drop}
FE2	Match fields = {Inport = 2,3,4; Type = TCP; Flags = SYN-ACK}; Timeouts = 0; Instructions = {Send to the Controller}
FE3	Match fields = {Inport = 1; Type = TCP, Flags = ACK}; Timeouts = 0; Instructions = {Go to Table 1}
FE4	Match fields = {Table-miss}; Timeouts = 0; Instructions = {Go to Table 2}
Flow Table 1	
FE51	Match fields = {srcIP = IPs1; srcPort = Ps1; dstIP = IPd1; dstPort = Pd1}; Timeouts = t_1; Instructions = {Send to the Controller}
FE52	Match fields = {srcIP = IPs2; srcPort = Ps2; dstIP = IPd2, dstPort = Pd2}; Timeouts = t_2; Instructions = {Send to the Controller}
FE53	Match fields = {srcIP = IPs3; srcPort= Ps3; dstIP = IPd3, dstPort = Pd3}; Timeouts = t_3; Instructions ={Send to the Controller}
…	…
FE6	Match fields ={Table-miss}; Timeouts = 0; Instructions = {Go to Table 2}
Flow Table 2	
FE7	Match fields = {Inport = 2,3,4; Type = TCP}; Timeouts = 0; Instructions = {Forward to port 1}
FE81	Match fields = {Inport = 1; Type = TCP; dstIP = IPd1}; Timeouts = 0; Instructions = {Forward to port 2}
FE82	Match fields = {Inport = 1; Type = TCP; dstIP = IPd2}; Timeouts = 0; Instructions = {Forward to port 3}
FE83	Match fields = {Inport = 1; Type = TCP; dstIP = IPd4}; Timeouts = 0; Instructions = {Forward to port 4}

server. Via the 'table-miss' flow entry FE6, the other ACK packets not belonging to the monitoring of HOCs are forwarded to *Flow Table* 2 and directly routed to the destination server.

∗ *Flow Table 2*:

This flow table takes responsibility of directing packets of TCP connections which have accomplished the three-way handshake process. It also may contain flow entries for other applications such as routing, load balancing, etc.

- Flow-entry FE7 captures TCP packets other than SYN-ACK generated from internal servers and forward them to Internet clients via the gateway port.
- Incoming TCP packets of a validated three-way handshake flow will be 'table-missed' with *Flow Table 0* and *Flow Table* 1 and matched to a FE8 flow entry then forwarded to the corresponding server through a specific port.

For processing packets other than TCP such as ICMP, UDP packets, we also can add flow entries in *Flow Table* 2. In the case that TCP flows need to be processed by other applications, other subsequent flow tables (such as Flow Table 3, etc) may be added and the Instructions in FE8s can be changed to *'Go to Flow Table 3'*.

5.2. SPM implementation at the SDN controller

The SPM is an application running in the controller to process packets that involves the establishment of TCP connections such as: SYN, SYN-ACK, ACK packets. In order to capture these packets, the SPM, via the controller, requires the OFS to install, modify or delete related flow entries (illustrated in Fig. 8). Depending on the particular characteristics of each server and its current states, SPM calculates a timeout t for each flow entry FE5. For managing HOCs at servers, SPM maintains a flow monitoring table (FMT) that contains details of each connection information including source and destination IP addresses, source and destination ports, initial sequence

TABLE 2. Structure of FMT.

FlowID	srcIP	srcPort	dstIP	dstPort	Sequene number
1	sIP1	sP1	dIP1	dP1	Seq_1
2	sIP2	sP2	dIP2	dP2	Seq_2
3	sIP3	sP3	dIP3	dP3	Seq_3
…	…	…	…	…	…

TABLE 3. Analysis Results of Legitimate Traffic in the CAIDA 2013 Dataset

Number of servers for analysis	100 servers
Traffic duration	45 minutes
Number of flows	3 947 883 flows
Average number of new flows per second per server	14.62 fps
Average of flow lifetime T_{life}	18.26 seconds
Average of three-handshake time $T_{\text{handshake}}$	0.93 seconds

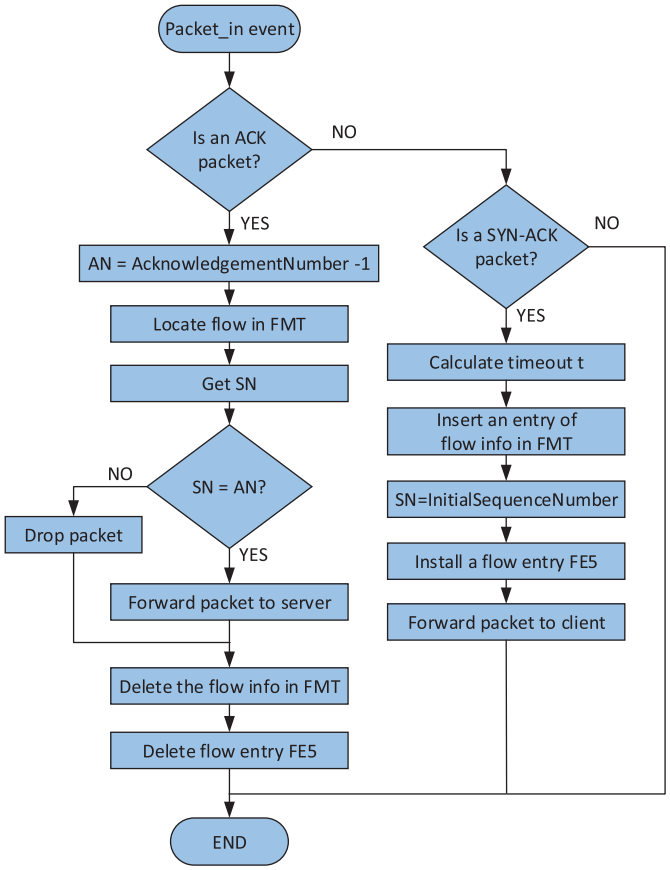

FIGURE 8. SPM flow diagram.

number of the ACK-SYN packets. Each HOC corresponds to a row in FMT. When a HOC is validated, migrated or discarded due to illegitimacy, the corresponding row is erased out of FMT. Table 2 describes the structure of FMT.

Upon receiving a SYN-ACK packet from an internal server that answers a SYN request, SPM extracts attribute information of the flow including initial sequence number Seq_i and inserts a new row into FMT. SPM also installs a corresponding flow entry FE5i to the switch and forward the packet to the client. On receiving an ACK packet from the client that confirms a TCP connection, SPM looks for the entry of the connection in the FMT and validates the connection based on comparing the acknowledgment number extracted from the packet and the Initial Sequence Number stored in connection entry. If the connection is valid, the entry is removed from FMT and SPM command the OFS delete flow entry FE5i in OFS. The whole process of SPM is illustrated in Fig. 8.

5.3. Flow-entry timeout calculation

In general, during TCP SYN flood, since there is no ACK packet from clients to complete the three-way handshake process, flow entries of this type of traffic will be expired

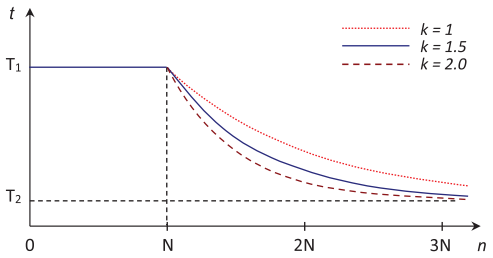

FIGURE 9. The dynamic timeout adjustment of flow entries.

only after a timeout (which is typically fixed 21 seconds as described in Section 4). With the SSP solution, timeout is adjusted according to the number of HOCs to a server at a given time. As described before in 4.1, the timeout of flow entries FE5s strictly relates to TIMEWAIT of HOCs in the server. Playing a SYN proxy role, if timeout t of flow entry FE5 is exceeded while OFS has not got the first ACK packet of the flow, the flow entry FE5 will be discarded since the flow is considered malicious. Subsequently, this event will be reported to SPM in order to delete the corresponding HOCs in the server by a forge ACK packet and a FIN packet. SPM adjusts value t for each flow entry FE5 based on statistical characteristics of a service and the current state of a server. In SSP, timeout is calculated according to the following formula:

$$t = \begin{cases} T_1, \, n \leq N \\ T_2 + (T_1 - T_2)e^{-k\frac{n-N}{N}}, \, n > N \end{cases} \quad (1)$$

where n, number of current HOCs processed by a server. N, average number of HOCs that a server processes during a normal traffic context. In fact, N has different values for each type of service in each particular server. T_1, the maximum time a server can wait for the first ACK packet after sending a SYN-ACK packet. T_2 : the minimum time a server can wait during attack. k, adjustment coefficient.

The adjustment of timeout t is described in Fig. 9. If the number of HOCs to a server is smaller than threshold N, value t is selected to be equal to T_1—the value TIMEWAIT normally used by the server. If the number of HOCs exceeds N, SPM decreases value t proportionally to the excess of n compared to N. With SSP, when TCP SYN flood occurs, the number of HOCs becomes very high, timeout t decreases approximately to T_2, resulting in a decrease in the lifetime of those flow entries. By reducing the default idle time out parameter of flow entries, unnecessary flow entries will be quickly erased from the flow table, thus decreasing occupation of the OFS resource. During a TCP SYN Flood, when the attack continues making the system install new flow entries, quick deletion of those flow entries will mitigate the attack impact on OFS.

5.4. Establishment of thresholds for changing policies to process SYN requests at OFS

As we know, the processing capacity of the switch depends on its hardware configuration. For each SYN request, the OFS consumes an amount of resource (CPU, memory) to process. At each point of time, depending on its capacity, the OFS can process a different number of SYN requests. In general, the switch's hardware capacity is calculated and designed in order to ensure covering a specific number of legitimate TCP connections depending on its applications. When the server is overloaded due to a flash crowd or DDoS attack, the number of SYN requests increases beyond the capacity of the server and the switch. To protect the switch and server from being overrun, SSP operates in two states corresponding to two policies of processing incoming SYN request: Activated and Deactivated. The transition between these two states is based on the two thresholds: M_1 and M_2 in which $M_1 > M_2$:

- M_1: Upper-bound threshold of concurrently processed SYN requests at the OFS. When the number of processing SYN requests reaches over this upper-bound threshold, SSP transits to state Deactivated and incoming SYN packets from clients will be discarded.
- M_2: Lower bound threshold of concurrently processed SYN requests. This is the threshold in which SSP is in state Deactivated. If the system is in the Deactivated state and the number of concurrently processed SYN requests goes lower than this threshold, SSP transits into the Activated state again, and incoming SYN packets from clients will be continuously processed by SSP.

In fact, the values of M_1 and M_2 are defined based on real capacity of the switch and real traffic volume during attack-free situations of the system. The process of monitoring and changing states is carried out by SPM. SPM retains a counter to count the number of concurrently processed SYN requests and compares its value to M_1 and M_2. When SSP transits from Deactivated to Activated, it modifies the instructions of flow entries FE1x in Flow Table 0 to the corresponding behavior. With this threshold mechanism, the system will not be collapsed during TCP SYN Flood or HTTP Flood (Fig. 10).

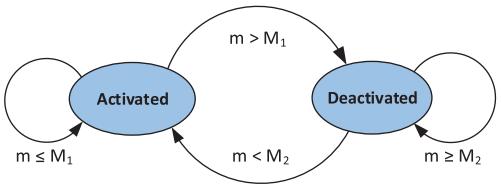

FIGURE 10. State transition of SSP.

With the aforementioned structure and general operation principle, the packet processing of the SSP framework in the Activated state is described in the following sequence diagram (Fig. 11).

6. TESTING ENVIRONMENT

In order to verify the performance of our solution, we built a testbed to emulate a real system as described in Fig. 12. Testbed components include:

- The controller is based on Floodlight controller [29] which is installed in an Ubuntu-14.04-version computer with the following configuration: CPU Intel TM Core i3-2330M @ 2.2 GHz, 500 GB HDD, 4 GB RAM.
- OFS is built based on a NetFPGA card running with OpenVswitch version 2.7.90. The configuration of NetFPGA card: Xilinx VirtexTM-II pro 50; 4 × 1 Gbps Ethernet ports with soft MAC; 4.5 Mb SRAM and 64 Mb DDR2; FPGA Spartan II used to be the Logic Control module for PCI interfaces.
- Legitimate Internet traffic is sent out from BONESI [30] tool while the attack traffic emulated by TCPReplay [31] with series of attack pattern created by BONESI [30] and Wireshark [32].
- FTP server acts as an application server.

7. SYSTEM PERFORMANCE ANALYSIS

To evaluate the performance of SSP, we will compare SSP with the very related work of the same type which deals with a SDN-based solution for monitoring the three-way handshaking process to detect TCP SYN Flood: Avant-Guard. Other SDN-based solutions based on statistically analysis of incoming traffic such as [21–26] are not taken into account since the nature difference of the approaches. As mentioned in Section 1, those papers propose a general way of detecting attacks by learning normal and attack patterns from given data logs. In one side, those schemes possibly can work for different types of DDoS attack but they always require more time for the statistical learning phase. For this specific TCP SYN Flood, we propose another way to handle the TCP attack detection faster by monitoring the three-way handshake process. This method requires less space to store data for the analytical process.

In this section, we will investigate the performance of SSP from multiple perspectives such as: lifetime of flow entries in OFS; how the SDN controller (as a component in our SDN-based SYN proxy) must endure during TCP SYN attack; possibility of successful connections establishment for a benign user to the server during TCP SYN attack; and the capability

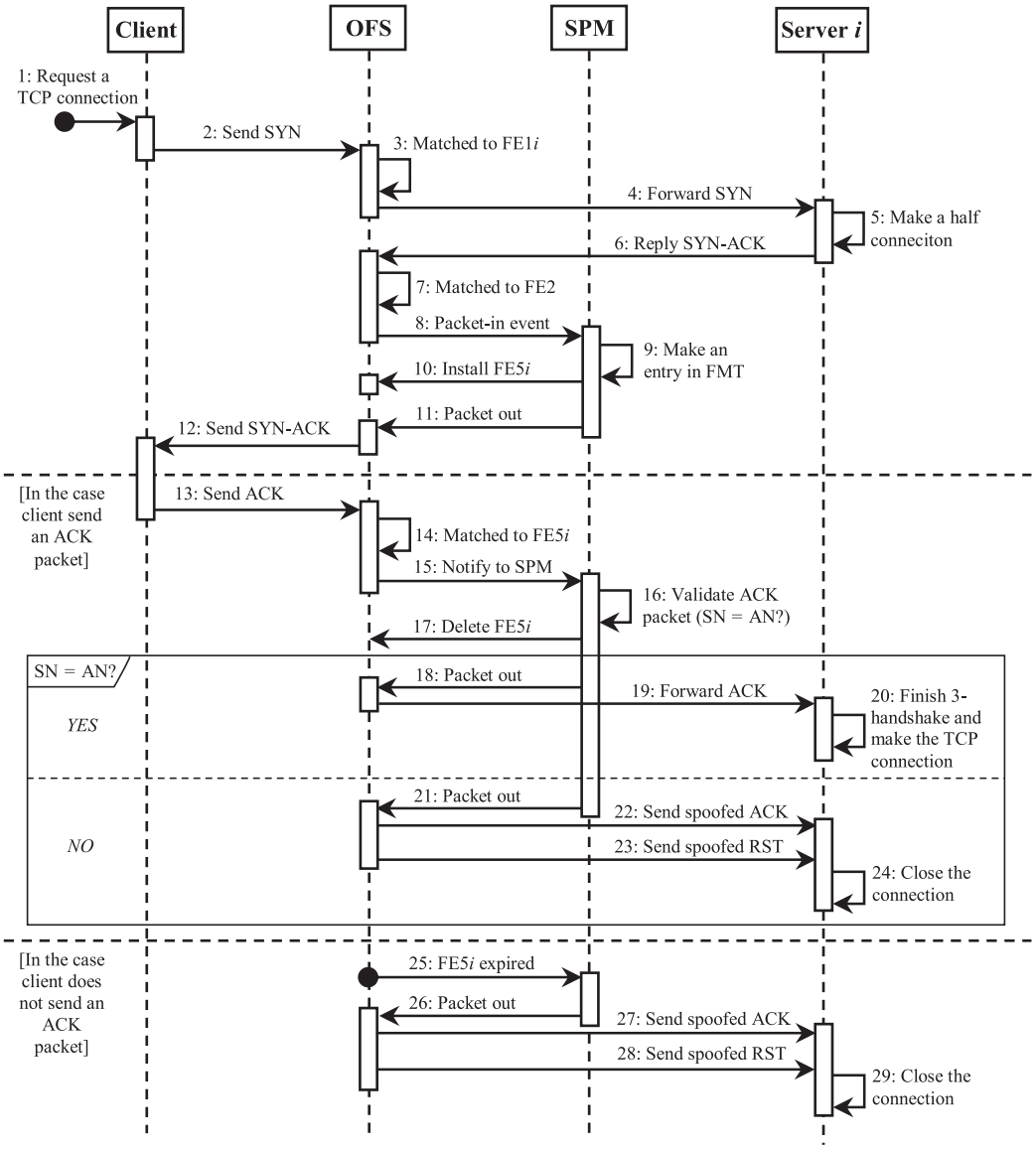

FIGURE 11. SSP framework operation.

of SSP in reducing HOCs in a victim server in comparison with the case the server not being protected by SSP.

7.1. Lifetime of flow entries in OFS

The main impact of SSP is to quickly reduce lifetime of unnecessary flow entries that occupy the flow table in OFS. In another word, flow entries are quicker removed to free up the OFS resource as soon as possible. In order to evaluate this performance of SSP, within the scope of SDN-based approaches against TCP SYN Flood, we compare SSP with Avant-Guard. Theoretically, in the Avant-Guard solution, after the three-way handshaking process has been censored by the Classification stage, the established TCP session is forwarded to the controller. If the controller allows a connection to the server, a corresponding flow entry is installed onto OFS. Therefore, the average lifetime of a flow entry in OFS is calculated by:

$$T_{\text{FE\_life\_AG}} = T_{\text{life}} + T_{\text{idleTimeOut}} - T_{\text{handshake}} \quad (2)$$

where T_{life} is the existing time of a flow and calculated by the time between the first and the last packet of the flow passed by the switch. $T_{\text{idleTimeOut}}$ is the time out of the flow. $T_{\text{handshake}}$ is the time the three-way handshake process consumes.

With the SSP solution, since the lifetime of a flow entry in OFS takes exactly a duration of one successful three-way

handshaking process, the average lifetime of a flow entry in OFS can be calculated by:

$$T_{\text{FE\_life\_SSP}} = T_{\text{handshake}} \quad (3)$$

To quantify the ratio of reducing flow entries of our SSP scheme and the Avant-Guard, we analyze traffic data trace extracted from legitimate traffic of 100 web servers during 45 minutes of the well-known CAIDA data set 2013 [28]. The analysis results of traffic parameters are presented in Table 3.

With the traffic analysis results presented in Table 3, for each benign TCP connection, the average lifetime of flow entries retained in OFS in case of Avant-Guard is $T_{\text{FE\_life\_AG}} = 18.26 - 0.93 = 17.33$ seconds, while in case of SSP is $T_{\text{FE\_life\_SSP}} = T_{\text{handshake}} = 0.93$ seconds. Hence, the average lifetime of flow entries in OFS reduces by $\frac{17.33 - 0.93}{17.33} = 94\%$, resulting in a decrease in the total number of flow entries at the switch at any point in time. This helps the OFS improve speed of packet processing as well as the capacity to endure under SYN flood attack.

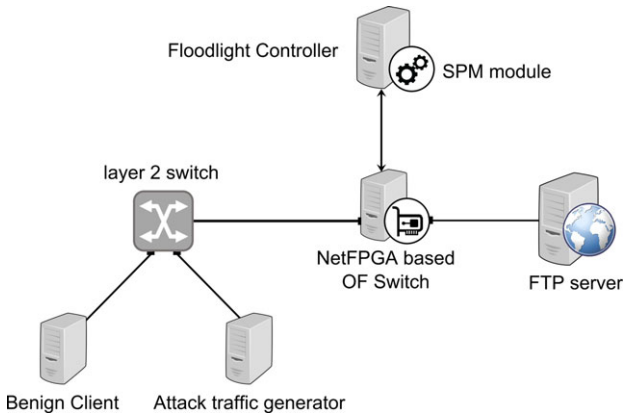

FIGURE 12. SSP testbed.

7.2. Impact on the controller when the system is under attack

In the SSP framework, the SDN controller, with the associated SPM, has to take additional work to handle states of the three-way handshake process of TCP flows. Therefore, we need to evaluate the extra workload the controller has to cope with, especially during TCP SYN attacks. To evaluate the impact of TCP SYN Floods on the controller, we measure resource consumption at the controller in two cases: (i) using the policy of dropping SYN requests to protect the system with setting up thresholds M_1 and M_2 and (ii) not using the policy. M_1 and M_2 are set 7500 and 7000 in accordance with the configuration and capabilities of the testbed system. And the results are shown in Figs 13 and 14.

The results show that in both scenarios during normal traffic, the CPU of the controller is consumed by approximately 20%. There is no difference between the two schemes under the low attack rate such as 100 SYN pps and 200 SYN pps. When under a higher rate of TCP SYN attack, the rate of CPU consumption increases steadily and accounts for 68% in the scheme of not using the thresholds to enable the Drop policy. In the other scheme, CPU consumption increases slower and remain relatively stable at nearly 50% at the rate attack of 800–1000 SYN pps.

The memory consumption is approximately 2 GB, almost the same for the two schemes during the attack free period and under TCP SYN flood attack with different rates. In conclusion, changing the policies of processing SYN requests with the selected thresholds M_1 and M_2 help to improve the capability

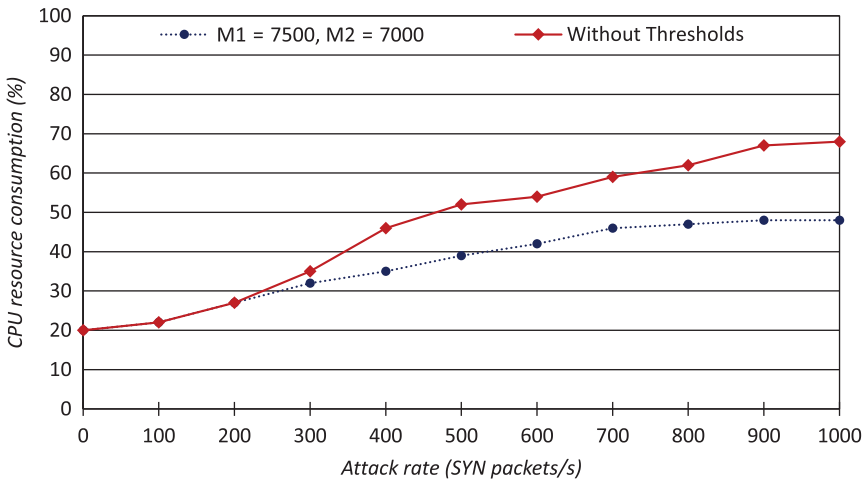

FIGURE 13. CPU consumption of the controller under different attack rates.

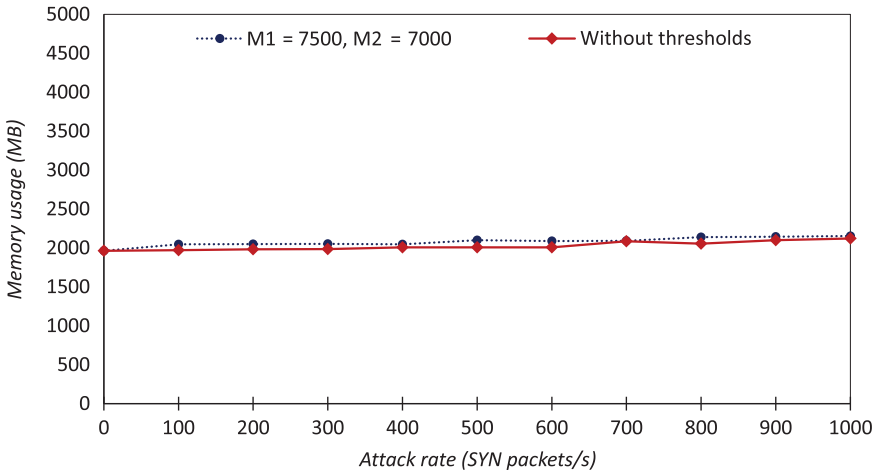

FIGURE 14. Memory consumption of the controller under different attack rates.

and configuration of the system, SSP does not affect the operation of the controller event under TCP SYN flood attacks.

7.3. Possibility of successful connection establishment for a legitimate TCP flow

We also consider the performance of SSP from another perspective, the possibility of successful connection establishment for a legitimate TCP Flow during TCP SYN Flood since the flood can hinder a normal user from connecting to an application server. To evaluate the possibility of successful connection establishment of benign users to a server during attack, we generate legitimate traffic with speed of 50 TCP connections per second along with TCP SYN flood speeds from 100 SYN pps to 1000 SYN pps. The experiment is implemented with two scenarios: Using SSP and using the standard Openflow only.

- In the Openflow scenario, a flow is defined by 5-tuple parameters including: source and destination IP addresses, protocol number, source and destination ports.
- In the SSP scenario, the two thresholds M_1 and M_2 are selected 7500 and 7000, respectively, to suit configuration and capacity of the soft switch OFS.

In the experiment, traffic is generated in 100 seconds in which legitimate traffic is generated later than attack traffic 20 seconds. All traffic at the benign client is captured, processed and analyzed to result in successful connection rate and average connection retrieval time versus attack rates. The results are demonstrated in Figs 15 and 16.

As shown in Fig. 15, when using only the standard Openflow, the successful connection rate decreases quickly from 87% to 5% when the system is under attack rates from 200 SYN pps to 600 SYN pps, respectively. When the attack rates go up to 700 SYN pps or more, the system is paralyzed and no longer be able to respond to legitimate TCP flows.

With SSP (Fig. 16), the successful connection rate begins to decrease to 95% at 300 SYN pps and further down to 58% at 1000 SYN pps. The experimental results show SSP remarkably improves the system's attack tolerance over the pure Openflow standard. As illustrated in Fig. 16, in the pure Openflow scheme, the average connection retrieval time also increases from 4.5 ms at the attack rate of 100 SYN pps to 126 ms at 600 SYN pps. While in the SSP scheme, this average time remains stable under 10 ms.

7.4. Number of HOCs at the server

The capacity of restoring from attack traffic at each server depends on its hardware capability and particular characteristics of each service running on that server. SSP can be considered a network-based approach to protect application servers from TCP SYN Flood without being equipped with a security function like SYN Cookie or SYN Cache. It helps application servers to close malicious HOCs as soon as possible. In our test, in order to evaluate the performance of the SSP framework, we compare the number of HOCs retained at a victim server during attack time when using SSP and not using SSP where servers are presumed to adopt the protection scheme of Windows server [13].

To implement this evaluation, we use the tool BONESI to generate and WireShark to edit the TCP-SYN attack traffic with attack speeds ranging from 100 SYN packets per second (pps) to 1000 SYN pps. Each set of attack traffic is directly transmitted from a client PC, via OFS, to the FTP server during the period of 100 seconds. The number of HOCs are

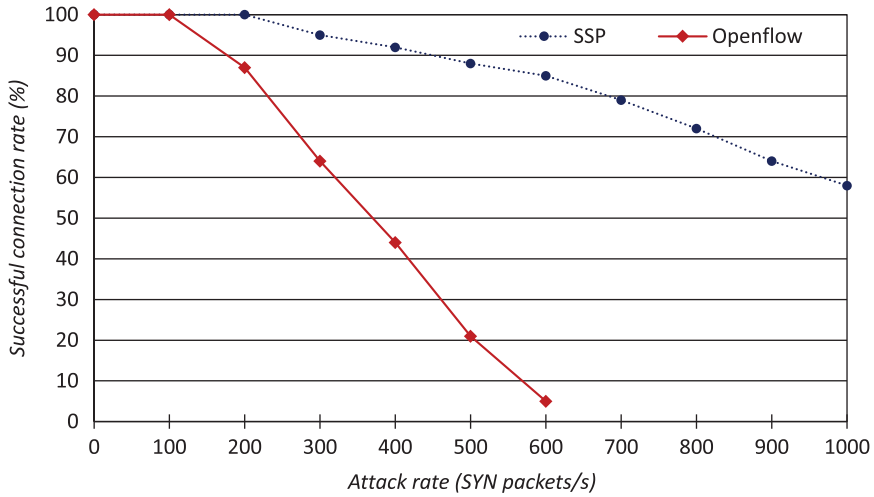

FIGURE 15. Successful connection rate vs. attack rate when using SSP and using standard Openflow.

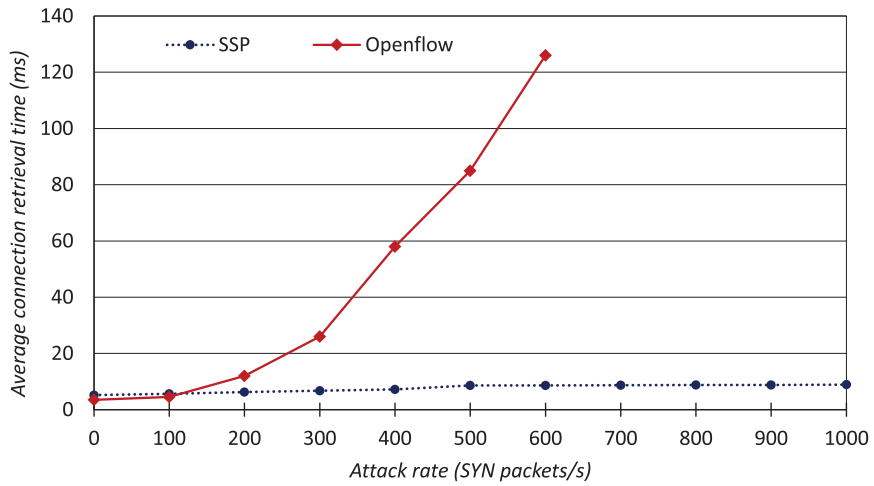

FIGURE 16. Average connection retrieval time vs. attack rate.

calculated by the SYN-RECEIVED command (which is the state statistics command) at the FTP server.

The parameters chosen for our SSP experiment include: $k = 1.5$; $T_1 = 15s$; $T_2 = 2s$; $N = 500$; $M_1 = 7500$ and $M_2 = 7000$. Values for these parameters are chosen stemming from the facts:

- In general, time out of flow entries in OFS is currently set up 15 s, therefore we set the threshold $T_1 = 15s$ (T_1—the maximum time a server can wait for the first ACK packet after sending a SYN-ACK packet).
- According to the CDF results shown in Fig. 4, 99% of legitimate TCP flows has the three-way handshaking processing time smaller than 2 seconds, we therefore set $T_2 = 2s$ (T_2- the minimum time a server can wait during attack).
- k is the adjustment coefficient. The more k increases, the faster T adjusted for each flow entry FE5 decreases. To investigate the performance of this solution, we experiment with $k = 1.5$.
- Average number of HOCs that a server processes during a normal traffic context $N = 500$.
- M_1 and M_2 are set in accordance with the configuration and capabilities of the testbed system.

The results of HOCs at the server corresponding to the three attack speeds are presented in Fig. 17a–c in which SSP outperforms the no SSP solution proportionally as the attack speed

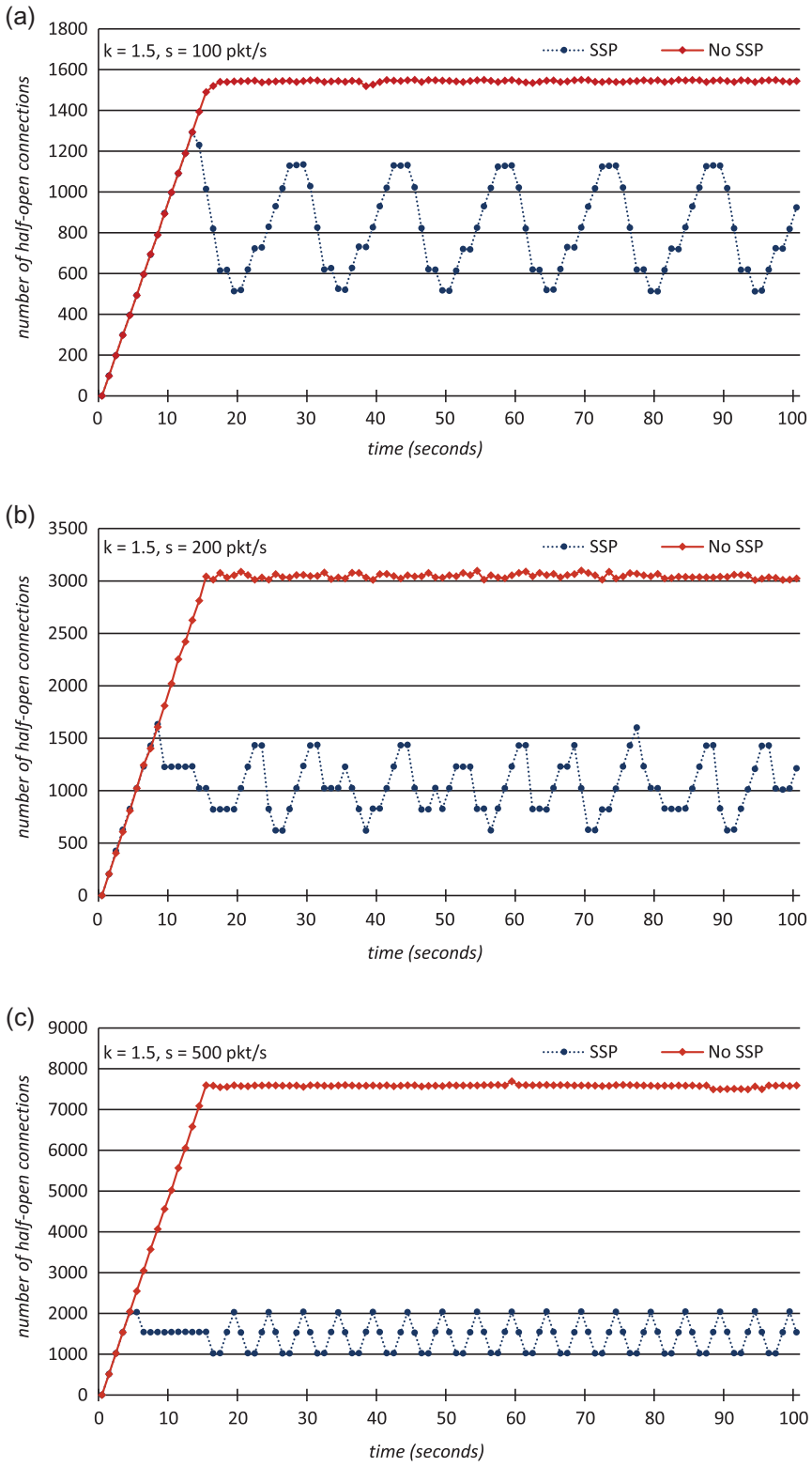

FIGURE 17. Number of HOCs at the server in three attack cases.

increases. For example, SSP can reduce the number of HOCs by 68% in case of 100 pkt/s rate, and by 86% in case of 500 pkt/s.

8. CONCLUSION

In this article, we have designed SSP—a coordination of the SDN Openflow switch and SDN controller to work as a stateless SYN proxy against the TCP SYN Flood merely based on the inherent packet processing mechanism of SDN Openflow.

The advantage of the SSP solution is that it can be exploited in any commerical Openflow switch product with version from 1.5 and later without requiring any futher hardware or protocol extension. We have studied the performance of SSP from multiple perspectives. In SSP, the switch (OFS) is proven to be made robust by reducing the lifetime of flow entries that occupy the switch's resource by 94% in comparison with Avant-Guard. SSP also considers the manner to protect the SDN controller from being a new victim in such a SDN architecture. The controller is proven to be endurable during situations in which CPU consumption of the controller increases but remains stable at a rate of 800–1000 SYN pps. In addition, as a network-based solution, SSP can fee up application servers' resource by removing useless HOCs at all application servers up to 86% in case of 500 pps. SSP also provides higher possibility of successful connection establishment during TCP SYN Flood in comparison with the original Openflow solution.

FUNDING

This work was supported by the Ministry of Science and Technology, Vietnam under the framework of the National Key project [code DTDLCN.38/15].

REFERENCES

[1] Peng, T., Leckie, C. and Ramamohanarao, K. (2007) Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Comput. Surv.*, **39**, 1.

[2] Mirkovic, J. and Reiher, P. (2004) A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Comput. Commun. Rev.*, **34**, 39–53.

[3] Verisign. *Verisign Distributed Denial of Service trends report*, Volumn 3, Issue 3, 3rd Quarter, 2016. https://www.verisign.com/assets/report-ddos-trends-Q32016.pdf, (accessed October 15, 2017).

[4] Mohamed, A.B. and Kandil, A. (2009) Strengthening and Securing the TCP/IP Stack Against SYN Attacks. *Proc. ITI 2009 31st Int. Conf. Information Technology Interfaces*, Piscataway, USA, June, pp. 627–632. University Computing Centre, University of Zagreb J.Marohnica 5, Croatia.

[5] The Internet Society. *Transmission Control Protocol. RFC 793*. https://tools.ietf.org/html/rfc793 (accessed October 15, 2017).

[6] The Internet Society. *TCP SYN flooding attacks and common mitigations*. RFC 4987. https://tools.ietf.org/html/rfc4987 (accessed October 15, 2017).

[7] Eddy, W.M. (2006) Defenses against TCP SYN flooding attacks. *Internet Protoc. J.*, **9**, 2–16.

[8] Open Network Foundation. *Software Defined Networking Definition*. https://opennetworking.org/sdn-resources/sdn-definition (accessed October 15, 2017).

[9] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J. (2008) Openflow: enabling innovation in campus networks. *ACM SIGCOMM Comput. Commun. Rev.*, **38**, 69–74.

[10] Benton, K., Camp, L.J. and Small, C. (2013) Openflow Vulnerability Assessment. *Proc. Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking—HotSDN '13*, Hong Kong, China, August 12–16, pp. 151–152. ACM New York, NY, USA.

[11] Shin, S., Yegneswaran, V., Porras, P. and Gu, G. (2013) AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks. *Proc. 2013 ACM SIGSAC Conf. Computer and Communications Security—CCS '13*, Berlin, Germany, November 04–08, pp. 413–424. ACM New York, NY, USA.

[12] Ambrosin, M., Conti, M., De Gaspari, F. and Poovendran, R. (2015) Lineswitch: Efficiently Managing Switch Flow in Software-Defined Networking While Effectively Tackling DOS Attacks. *Proc. 10th ACM Symposium on Information, Computer and Communications Security—ASIA CCS '15*, Singapore, Republic of Singapore, April 14–17, pp. 639–644. ACM New York, NY, USA.

[13] Kumar, S., Member, S. and Reddy Gade, R. (2015) Evaluation of Microsoft Windows Servers 2008 & 2003 against Cyber Attacks. *J. Inf. Secur.*, **6**, 155–160. doi:10.4236/jis.2015.62016.

[14] Twycross, J. and Williamson, M.M. (2003) Implementing and testing a virus throttle. *Proc. 12th Conf. USENIX Security Symposium, SSYM'03 - Volume 12*, Washington, DC, USA, August 04–08, pp. 20–20. USENIX Association, Berkeley, CA, USA.

[15] Williamson, M.M. (2002) Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. *Proc. IEEE 18th Annual Computer Security Applications Conf.*, Las Vegas, NV, USA, December 9–13, pp. 9–13. IEEE Computer Society Washington, DC, USA.

[16] Lemon, J. (2002) Resisting SYN Flood DoS Attacks with a SYN Cache. *Proc. BSD Conf. 2002 on BSD Conference—BSDC'02*, San Francisco, CA, USA, February 11–14, pp. 10–10. USENIX Association, Berkeley, CA, USA.

[17] Bernstein, D. (1996). *SYN cookies*. http://cr.yp.to/syncookies.html (accessed October 15, 2017).

[18] Juniper Network (2015). *Junos OS—Attack Detection and Prevention Feature Guide for Security Devices, White paper*. http://www.juniper.net/techpubs/en_US/junos/information-products/pathway-pages/security/security-attack-denial-of-service.pdf (accessed October 15, 2017).

[19] Deal, R. (2004) *Cisco Router Firewall Security*. Cisco Press. ISBN:978-1-58705-175-3.

[20] Schuba, C., Krsul, I., Kuhn, M., Spafford, E., Sundaram, A. and Zamboni, D. (1997) Analysis of a Denial of Service attack on TCP. *Proc. 1997 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 4–7, pp. 208–223. IEEE Computer Society, Washington, DC, USA.

[21] Braga, R., Mota, E. and Passito, A. (2010) Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. *Proc. 2010 IEEE 35th Conf. Local Computer Networks—LCN '10*, Denver, Colorado, USA, October 11–14, pp. 408–415. IEEE Computer Society, Washington, DC, USA.

[22] Haopei, W., Lei, X. and Guofei, G. (2014) OF-GUARD: A DoS Attack Prevention Extension in Software-Defined Networks. *White paper In USENIX Open Network Summit*. https://www.usenix.org/sites/default/files/ons2014-poster-wang.pdf (accessed October 15, 2017).

[23] Mehdi, S.A., Khalid, J. and Khayam, S.A. (2011) Revisiting Traffic Anomaly Detection Using Software Defined Networking. *Proc. 14th Int. Conf. Recent Advances in Intrusion Detection—RAID'11*, Menlo Park, CA, USA, September 20–21, pp. 161–180. Springer, Berlin, Heidelberg.

[24] Li, J., Berg, S., Zhang, M., Reiher, P. and Wei, T. (2014) Drawbridge: Software-Defined DDoS-Resistant Traffic Engineering. *Proc. 2014 ACM Conf. SIGCOMM—SIGCOMM '14*, Chicago, IL, USA, August 17–22, pp. 591–592. ACM New York, NY, USA.

[25] Trung, P.V., Huong, T.T., Tuyen, D.V., Duc, D.M., Thanh, N.H. and Marshall, A. (2015) A Multi-Criteria-Based DDoS-Attack Prevention Solution Using Softwaredefined Networking. *Proc. IEEE 2015 Int. Conf. Advanced Technologies for Communications, ATC 2015*, HoChiMinh City, Vietnam, October 14–16, pp. 308–313. IEEE Computer Society, Washington, DC, USA.

[26] Trung, P.V., Toan, T.V., Tuyen, D.V., Huong, T.T. and Thanh, N.H. (2016) OpenFlowSIA: An Optimized Protection Scheme for Software-Defined Networks From Flooding Attacks. *2016 IEEE Sixth Int. Conf. Communications and Electronics, ICCE 2016*, Halong Bay, Vietnam, July 27–29, pp. 13–18. IEEE Computer Society, Washington, DC, USA.

[27] Open Network Foundation (2015). *Openflow switch specification*, version 1.5.1. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.1.pdf (accessed October 15, 2017).

[28] CAIDA (2013). *The CAIDA UCSD Anonymized Internet Traces* 2013-20131121-130000 (2013). http://www.caida.org/data/passive-2013-dataset.xml (accessed July 16, 2015).

[29] Project Floodlight - *Open Source Software for Building Software-Defined Networks*. http://www.projectfloodlight.org (accessed October 15, 2017).

[30] BONESI *A Network Stress Testing Application*. https://github.com/Markus-Go/bonesi (accessed October 15, 2017).

[31] TCPReplay *Pcap Editing and Replay Tool for *NIX*. http://tcpreplay.synfin.net (accessed October 15, 2017).

[32] Wireshark *Used Network Protocol Analyzer*. https://www.wireshark.org (accessed October 15, 2017).