PBUF: Sharing buffer to mitigate flooding attacks

PBUF: Sharing Buffer to Mitigate Flooding Attacks

Changting Lin*, Chunming Wu*, Yifei Tian*, Zhenyu Wen† and Shouling Ji*
College of Computer Science and Technology, Zhejiang University, China *
Email: {linchangting, wuchunming, tianyifei, sji}@zju.edu.cn
School of Informatics, University of Edinburgh, UK †
Email: zwen@inf.ed.ac.uk

***Abstract*—Software defined networking (SDN) is a promising network architecture, which decouples the control plane and data plane of a network. However, SDN opens some security challenges, such as man-in-the-middle attacks, spoofing attacks, flooding attacks and so on. In this paper, we focus on flooding attacks which consume the switch buffer and controller resource resulting in SDN framework resource overloaded. To prevent SDN framework from flooding attack, we present a defense approach called PBUF (Packet forwarding based on BUFfer sharing), which pools the idle switches to mitigate threat issues. This approach consists of *buffer management* and *packet forwarding* modules. The *buffer management* module gleans the statistics of incoming packets and then analyzes these statistics to estimate the buffer size by network calculus. Considering that a lot of `table-miss` packets will be generated and stored in buffer when the flooding attack is happening, the *packet forwarding* module is designed to forward these `table-miss` packets to idle switches to prevent the switch or controller to be overloaded. These `table-miss` packets will be buffered in idle switches and then sent to controller in a limited rate by generating `packet_in` messages. The simulation results show that PBUF is effective and only introduces a little overhead in SDN framework.**

***Index Terms*—Flooding Attack; Security; Performance; SDN**

I. Introduction

Software defined networking (SDN) [1] has become one of the important network architectures for simplifying network management and enabling innovation in communication networks. There are some vulnerabilities and limitations on SDN framework on account of the principle of separating control plane from data plane. For example, the controller can access the entire SDN, therefore it brings a disaster if the controller is compromised. Moreover, the OpenFlow switch (data plane) also attracts attacks by some potential security vulnerabilities, e.g., a limited buffer size may lead to buffer overflow and further consume the computation resource of the controller in a short time when the switch is attacked by a large number of new packets[2]. All of these new packets, with all or part of header fields are spoofed as random values, should be processed by controller in a short time. We call these new packets `table-miss` packets.

Meanwhile, some researchers have started to explore the security threats and propose some possible defense methods in SDNs [2] [3] [4] [5]. AVANT-GUARD [3] enables the control plane and the SDN network to be more resilient and scalable against control plane saturation attacks. FloodGuard [2] is an advanced defense approach of AVANT-GUARD, which proposes a proactive flow rule method and designs an extra cache to reduce the amount of `packet_in` messages and therefore restricts the abilities of an attacker to be successful. Occurring `packet_in` messages are stored in an extra cache and served using a limited rate to further reduce the impact of the attack. However, it is expensive to design an extra cache for storing `table-miss` packets in FloodGuard. Moreover, the way how to set a threshold, indicating attack occurs or ends, is not detailedly discussed in FloodGuard.

In this paper, we attempt to defense the flooding attack which causes the switch buffer overflowed and controller overloaded attack. Unlike FloodGuard, our proposed defense method does not require to design an extra cache for storing `packet_in` messages and is attack driven. We leverage the idle buffer in other switches to store `table-miss` packets. These `table-miss` packets will be served in switches at a limited rate to reduce the controller overloaded. In addition, we use network calculus to estimate switch buffer size and further trigger PBUF that avoid the buffer overflow. PBUF can retard the time of the buffer overflow happening at least 5 times, compared with normal OpenFlow solution. The retarded time is increased with the increasing number of switches.

To summarize, the contributions of our paper include the following:

- We design the PBUF, an attack driven and efficient defense approach, for SDN networks to prevent switch buffer overflow and controller overloaded attack by using *buffer management* and *packet forwarding* modules. *buffer management* module estimates the buffer size and trigger PBUF to defense attack.
- We implement PBUF algorithm in *packet forwarding* module, which can effectively forward these `table-miss` packets to idle switches without overflow.
- We implement PBUF in SDN framework and test its defense effectiveness and performance. The evaluation results show that PBUF can achieve an efficient and a low overhead when the flooding attack is happening.

The remainder of this paper is organized as follows. In Section II, we will introduce the background including SDN, network calculus, motivation and research challenges. Some related works about SDN security will be introduced in Section III. The design of PBUF is detailed in Section IV. The implementation and evaluation of PBUF is introduced in

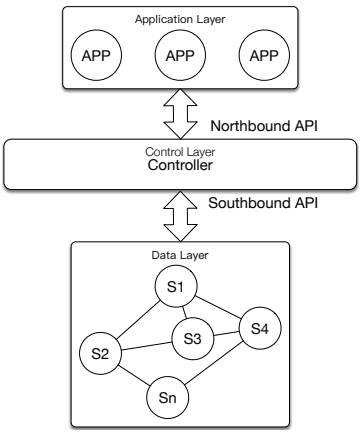


Fig. 1. SDN Framework.

Section V. At last we discuss and conclude our work in Section VI.

II. Background

A. Vulnerabilities of SDN

SDN [6] is currently attracting significant attention from both academia and industry. The framework of SDN is shown in figure 1. However, SDN indeed raises some open security challenges. Then, we summarize some security issues according to figure 1 as follows.

- *Control layer*. A compromised controller can compromise the whole network since a centralized controller is the centralized decision-making entity. This drawback attracts different attacks such as DoS attacks [7]. Due to the controller's limit capacity, it becomes a bottleneck [8].
- *Data layer*. A SDN compatible switch (e.g., OpenFlow switch) is easy to be compromised [9]. Furthermore, a SDN switch has to buffer incoming packets until the controller issues flow rules. However, the switch always has a limited number of flow tables and buffer size that data plane cannot handle a saturation attacks [2] or traffic uncertainty in OpenFlow data plane [10]. These attacks subsequently lead to DoS attacks that render the data plane in an unpredictable state.
- *Application layer*. The application is responsible for providing a set of services and applications. The deployed network applications can manipulate the behavior of the network, and yet they could cause serious security challenges [11]. Likewise, a large number of third-party applications could result in serious security vulnerabilities and challenges because of the lack of a well authentication for third-party applications[12].

B. Network Calculus

Network calculus is a mathematical study of queues and can derive lower bounds (i.e., worst-case) of the system, including backlog, delay and similar performance metrics [13]. Network calculus particular focuses on quality of service guarantee analysis. This theory uses the alternate algebras such as the min-plus and max-plus algebra to transform complex network systems into analytically tractable systems. To complete these analysis, incoming traffic and service provides by a system are all modeled as functions.

Network calculus holds promise as a valuable systematic methodology for the performance analysis of computer and communication systems [14]. In this paper, we hence leverage network calculus to estimate switch buffer size in SDN. Furthermore, a threshold is set, which is used to trigger PBUF. For the sake of discussion, we use OVS (Open vSwitch) as the default SDN switch. To avoid OVS buffer overflow, we should trigger our approach before the buffer is running out. Considering that network calculus is applied to find the lower bound, the estimated value is always larger than the true value. Therefore, the threshold value can be set as an OVS default buffer size value (256 packets), that the overflow will not occur in OVS when the flooding attack is coming.

C. Motivation

Studying on data plane and control plane, we find that they all have bottlenecks. For example, 5406zl switch has a limited TCAM size which only instals about 1500 OpenFlow rules [8]. In addition, some SDN compatible switches can only generate about 150 `packet_in` messages per second (i.e., Pic8 Pronto) [15]. Moreover, Wang [15] found that the bottleneck is at the control plane rather than at the data plane.

As discussed above, a huge number of `table-miss` packets are waiting for processing while the flooding attack is happening on account of the control and data plane has a limitation capacity [2] [15]. Nonetheless, OpenFlow switch cannot always offer sufficient switch buffer to store all of these `table-miss` packets. Consequently, it would attract various security threats, such as buffer overflow, sessions interrupting and information leaking through side-channel attacks [16].

There are some solutions can handle `table-miss` packets when the buffer is full. Generally speaking, these `table-miss` packets can be dropped, forwarded to controller by `packet_in` messages contains whole packet or directed to a subsequent switch by some methods [17] [18]. However, it may cause some significant issues. Firstly, some legitimate requests will be interrupted if the packets are dropped. Secondly, the controller has a limitation capacity to process these `table-miss` packets in a short time. Thirdly, these overflow packets may be directed to a subsequent switch randomly [18] or by other methods like ECMP (Equal-Cost Multi-Path) [19] [17], which will cause link congestion or routing bottlenecks [20].

In this paper, we intend to design an attack driven defense approach called PBUF, which can be triggered timely and further prevent buffer overflow. Then, these `table-miss` packets will be scheduled to idle switches and wait for processing without causing bandwidth congestion. Considering to prevent the controller overloaded attack, these `table-miss` packets will be sent to controller by a limited rate.

D. Research Challenges

To realize PBUF, there are three challenges need to be resolved:

- Estimating the available buffer of each switch in real time without bringing performance decreasing.
- A threshold is needed which is used to trigger PBUF and prevent buffer overflow.
- Efficiently distributing `table-miss` packets to destination, without causing new security issues and performance decreasing.

To address the first challenge, we sample the incoming packets to effectively estimate the available size of the switches. For the second challenge, we can aware before the switch buffer size is going to be full, since the network calculus is applied to find the lower bound [21]. To address the third challenge, regarding the `table-miss` packets distribution process, we propose the PBUF algorithm to efficiently forwarding packets to idle switches without bringing network congestion and the other networking security issues.

III. Related Work

Some previous works had presented some flooding attacks and defense methods in SDN. Mahout [22] uses traditional statistic based aggregation solutions to prevent flooding attacks in SDN. AVANT-GUARD [3] is presented to defense TCP SYN flooding attack in SDN, which can reduce the amount of data-to-control-plane interactions under DoS attacks, and solve the communication bottleneck between the data plane and the control plane. However, it only can defense against TCP SYN flooding attacks. Wang *et al.* [2] presented a data-to-control plane DoS attack in SDN and then introduced FloodGuard, an efficient framework for defensing against DoS attack by using migration agent and data plane cache. The migration agent flooding attack and aims to protect switches and controller when an attack occurs. An extra cache is designed that stores proactive flow rules and caches `table-miss` packets. FloodGuard is an updated version of AVANT-GUARD, which can prevent SDN from more kinds of DoS attack. Nevertheless, designing an extra cache for storing packets is expensive and cannot be generalized to all SDNs very well. Moreover, FloodGuard is attack driven approach which is triggered by a threshold value. However, Wang has not discussed the threshold value exhaustively in their paper. Instead of implementing an extra cache to store these `table-miss` packets, we propose a method that reduces the burden of a single switch by forwarding packets from the over loaded switches to idle switches. Moreover, PBUF use network calculus to estimate the switch buffer size and set a threshold.

IV. Design

In this section, we demonstrate a feasible approach called PBUF, which offloads some stateful flow processing and control tasks directly inside the network switches, so as to reduce the switch-to-controllers signaling overhead and the latency shortcomings [23]. At first, we will glean the statistics of incoming packets. Then, network calculus is used to estimate the switch buffer. Based on the approximation, we can forward the packets to a right switch to mitigate switch overloading. These packets were bounced several times in data plane before finally processed by controller. The design of PBUF will be represented in detail as follows.

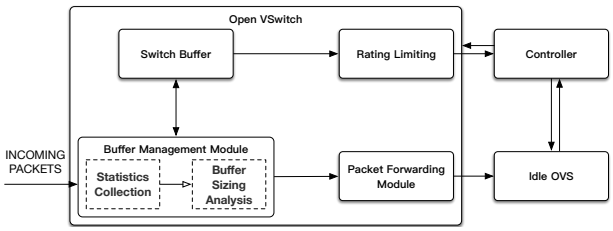


Fig. 2. Workflow of PBUF.

A. Overview

PBUF consists of two modules: (1) *buffer management* module and (2) *packet forwarding* module. The workflow of PBUF is shown in figure 2.

The *buffer management* module contains *statistics collection* and *buffer sizing analysis*. The *buffer management* module is used to gather incoming traffic and estimate the buffer size of switch. At first, the OVS gleans the statistics of incoming traffic that the used buffer size can be estimated by network calculus. Since PBUF is attack driven, we set a threshold value which equals to OVS default buffer size. Thus, PBUF will be triggered if the threshold value is not less than the OVS total buffer size. Since the network calculus will estimate a lower bound, the OVS will be alerted before the buffer overflows. In the *packet forwarding* module, the `table-miss` packets are forwarded to other switches by solving the optimization problem, which achieves low overhead and no congestion. Then these packets will be sent to the controller as `packet_in` messages by using a limited rate.

B. Statistics Collection

For the sake of collecting traffic information, we need to access switches for reading the traffic statistics. OVS supports three per-flow counters (packets, bytes and flow duration) for collecting the statistics in switch. In PBUF, packet counter is used to statistics gathering by default.

In this paper, OVS gathers these incoming packets statistics and then analyses these statistics. Each OVS monitors its own status instead of gathering these information by controller. As a collateral benefit, possible resource depletion (e.g., CPU usage), especially of the controller, is minimized during statistics collection. Furthermore, instead of monitoring each packet in the network, PBUF samples the incoming traffic for analyzing. Based on the collected statistics, the lower bound of the available buffer of each switch is detailed, which is detailed in next subsection. The available buffer will be used for designing packet forwarding algorithm.

C. Buffer Sizing analysis

*1) **Definitions**:* We suppose that there are K switches in SDN and with the same buffer size L. Furthermore, the

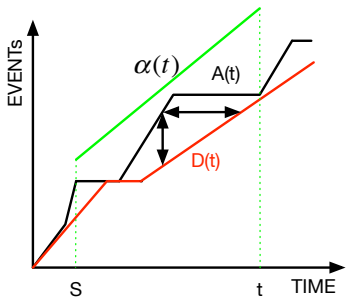


Fig. 3. Backlog and delay features based on network calculus.

switch i $(1 \leq i \leq K)$ serves n_i hosts. Then, we assume the used buffer size is B_i^{used} and the unused buffer size is B_i^{unused}. Furthermore, we assume that all switches have the same buffer size value. This assumption is relaxed. We make this assumption only because it would be tidier to describe the system. Consequently, we have:

$$B = B_i^{used} + B_i^{unused}. \tag{1}$$

In the SDN paradigm, when a packet arrives at an edge switch, according to the source address or source port or destination address or destination port, the packet will match a related existing rule in switch's TCAM and then forwarded it to the next switch. If there are no rules matched (`table-miss`), the `packet_in` message (including packet header and buffer ID which are only a little fraction of a packet) will be sent to controller when the switch's buffer is sufficient. However, if the buffer is running out, a `packet_in` message, containing an entire packet, is to be sent to controller for processing. The packets in switch buffer can be modeled as a queue system, therefore network calculus can be applied to estimate some related metrics. The packets' arrival can be denoted as an arrival process $A(t)$, which denotes the total number of packet arrivals in time slots $1, 2, ..., t$. The cumulative arrival process $A(t)$ is a non-decreasing, integer valued function on the non-negative integer Z_+ such that $A(0) = 0$. The number of packet arrivals at time t is denoted by $a(t)$. Hence, we have $a(t) = A(t) - A(t-1)$.

When $t \geq 0$, $\exists \alpha(\cdot)$, $\forall s \leq t$, $A(t) - A(s) \leq \alpha(t-s)$. Therefore, we consider that $\alpha(\cdot)$ is one of arrival curve of $A(t)$. In other words, the cumulative arrival process $A(t)$ is said to be (σ, ρ)-upper constrained, which is denoted as $A(t) \sim \alpha(t)$, where $\alpha(t) = \rho(t-s) + \sigma$. We hence call the function $\alpha(t)$ is an envelope of the arrivals process $A(t)$. As shown in figure 3, the $\alpha(t)$ can cover the process $A(t)$ and can be called as an envelope of $A(t)$. Furthermore, the burstiness and the average sustainable rate of arrivals are represented by σ and ρ, respectively. Moreover, we can see that the queue system will be backlogged at t if $A(t) > D(t)$.

*2) **SDN Switch Model**:* Our SDN switch model is shown in figure 4, which is similar to [24]. We assume that the model consists of cumulative `table-miss` packets arrival $A(t)$ and legitimate packets $M(t)$, service curve $S(t)$ and cumulative packets departure $D(t)$. The $S(t)$ is denoted as the rate of generating `packet_in` messages in switch. The $M(t)$ is

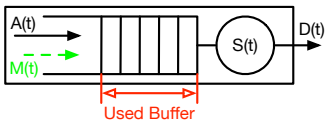


Fig. 4. An analytical model of SDN switch .

the incoming packets from benign hosts, which contains a little new packets and most matched packets. These matched packets will be forwarded to related switch directly. Hence, a little number of packets from $M(t)$ will be buffered in switch.

*3) **Analysis of the SDN Switch**:* The `table-miss` $A(t)$ should generate `packet_in` events and send to controller, which contain only a little fraction of the packet header and a buffer ID to be used by a controller if the buffer size is sufficient. The most part of packet still queue in buffer. The $A(t)$ will be buffered in a queue system waiting for process. However, the $M(t)$ is supposed to be a constant value M. Moreover, the most part of M will not be processed by server, which will be forwarded to destination directly. Therefore, only m packets should be processed. Given a system with service curve $S(t)$ and upper constrained arrivals with envelope $\alpha(t)$, the departure process $D(t)$ can be derived [21]. We have:

$$D(t) = \sup_{\tau \geq 0}\{\alpha(t+\tau) - (S(\tau) - m)\}. \tag{2}$$

Accordingly, the backlog (used buffer size) of switch i is defined as [21]:

$$B_i^{used} = A(t) - D(t). \tag{3}$$

Moreover, backlog is the vertical distance between the cumulative arrival and departure functions. By insertion of the definition of service curve and arrival envelope, a worst-case bound for the maximal backlog $B_{\max}$ can be derived [21]. We have:

$$B_{\max} \leq \sup_{\tau \geq 0}\{\alpha(\tau) - S((\tau) - m)\} = \alpha \oslash S(0) - m. \tag{4}$$

where $\oslash$ is referred to as the min-plus de-convolution.

We hence can estimate the worst-case backlog bound in a queue, which is caused by incoming packets. If the $B_{\max}$ reaches to the threshold B, we suppose that the buffer overflow will occur. Thereafter, the packets must be forwarded to other switches. The reason we set B as the threshold is that the $B_{\max}$ is an estimation of the worst-case which is larger than the real value.

D. Packet Forwarding

We propose a mitigate defense approach, under which, a switch treats each other switch as a *node*. When a *node* is going to run out of its own buffer, the other *nodes* will support their own idle buffer for buffer `table-miss` packets, thus to mitigate the flooding attacks. Moreover, these packets will be guided to other idle *nodes* by *packet forwarding* module. Consequently, these packets will be distributed to the entire network rather than being aggregated at a switch. In this way, the idle resources in network can be used to mitigate attacks.

Accordingly, a new problem is raised: how to distribute these packets to the other idle $nodes$ efficiently. PBUF follows the following principles: 1) the $node$'s buffer is not full, 2) the $node$ has a larger buffer size than the other, 3) the idle $node$ is nearest to the to-be-overloaded switch, 4) and the $node$ connects to more switches than the other. Furthermore, the controller has the information of the topology, such as the paths' bandwidth. Thus, the attack effects will be limited to minimal by PBUF, which aggregates the idle resource of the entire network for defending against flooding attack. At the first and second principles, we use network calculus to estimate the buffer size and then encapsulate the estimated buffer size and rate information in LLDP (Link Layer Discovery Protocol) packets, which is used to discover the interconnected links between the OVS in SDN. LLDP packets are sent regularly via each port of a switch and are addressed to a bridge-filtered multicast address, and are therefore not forwarded by switches, but only sent across a single hop. For the third principle, each $node$ has the knowledge of neighbor $nodes$. For the fourth principle, the controller has the knowledge of topology that we can pre-store this topology information in OVS on account of the topology is unchangeable.

The number of flooding packets will be large in a short time that the distributing packets method focuses on speed rather than effectively. Correspondingly, we present a PBUF algorithm to solve this problem is a promising candidate as it can favor large number packets, which rapidly chooses an idle $node$ to buffer these `table-miss` packets at each stage and brings little overhead. Although the ECMP or random method can also distribute these packets to entire network quickly, it raises some issues, such as bandwidth congestion and network loop. However, the calculated forwarding paths of PBUF update frequently and can avoid link congestion, which is updated according to the real time network state.

PBUF algorithm is "greedy": we leverage this algorithm to find a path efficiently that does not contain the to-be-overloaded $nodes$ and raise the other networking security issue. Furthermore, based on this algorithm, each packet selects approximate well $node$ according to the neighbor $nodes'$ states which update all the time. Therefore, all of the incoming packets will be distributed to idle $nodes$ to avoid bandwidth congestion. The main iterative steps of greedy algorithm are shown in algorithm 1. The algorithm 1 decides the input pkt destination according to $nodes$ and path resource level. If the incoming pkt has only one available $nodes$ to be forwarded, which happens to be the previous $nodes$, this pkt would be sent to controller. Otherwise, the incoming pkt will be sent to $dstNode$(Lines:2-7). Then, a $node$ will be chosen, which has a lower buffer size(Lines:9-24). To avoid network loop, the $node$ where the pkt coming from would be exclude (Lines:10-11).

Algorithm 1 PBUF

Require: Current node: $curtNode$, input packet: pkt
Ensure: Destination node: $dstNode$
1: $dstNode$ = null
2: **if** $curtNode$.nearByNodesNum == 1 **then**
3: **if** pkt received from $curtNode$.nearByNodes.first **then**
4: send to controller
5: **else**
6: $dstNode$ = $curtNode$.nearByNodes.first
7: **end if**
8: **else**
9: **for** $node$ in $curtNode$.nearByNodes **do**
10: **if** pkt received from $node$ **then**
11: **continue**
12: **else if** $dstNode$ == null **then**
13: $dstNode$ = $node$
14: **continue**
15: **end if**
16: **if** $node$.bufferNum $<$ $dstNode$.bufferNum **then**
17: $dstNode$ = $node$
18: **else if** $node$.bufferNum == $dstNode$.bufferNum **then**
19: **if** $node$.nearByNodesNum $>$ $dstNode$.nearByNodesNum **then**
20: $dstNode$ = $node$
21: **end if**
22: **end if**
23: **end for**
24: $dstNode$.bufferNum = $dstNode$.bufferNum + 1
25: **end if**

V. Evaluation

In this section, we evaluate the effectiveness and performance of PBUF. Firstly, we study the effectiveness of the buffer overflow attacks. Then, we evaluate the effectiveness of our proposed strategy with simulations. To investigate the overheads of the proposed strategy, we evaluate the performance on the controller.

A. Experiment Setup

Topology: We use *Mininet* to emulate the OpenFlow-enabled network data plane and implement the *buffer management* module and *packet forwarding* module in OVS. For sake of simulating PBUF, we generate a small topology (m Core switches, n Agg switches, k ToR switches, and each ToR switch has 2 hosts). For example, a topology is shown in figure 5 where $m = 2$, $n = 3$, $k = 6$. OpenDaylight is used as the controller. Moreover, an application running on OpenDaylight is used to guide the incoming packets to destination by matching source and destination address. Besides, the features of OVS and OpenFlow is 2.6.0 and 1.5.0, respectively.

Adversary model: For simulating flooding attack feature well, we implement the packet-level simulations. There are some legitimated packets from benign hosts in SDN, which contain `table-match` and `table-miss` packets. For modeling the process packets more intuitively, we make the assumption that the legitimated packets number is a constant value in each time slot. Furthermore, the number of `table-match` packets are much larger than the number of `table-miss` packets in incoming legitimated packets. We use *Hping3* to generate and simulate TCP-based flooding attacks that these flooding packets' source and destination are forged and random. Accordingly, each of incoming packet is a new packet for OVS that the OVS should generate a new `packet_in` message for each new packet. By default, the buffer size of OVS are 256 packets. Further, one benign host connects to the ingress OVS. The benign host will generate

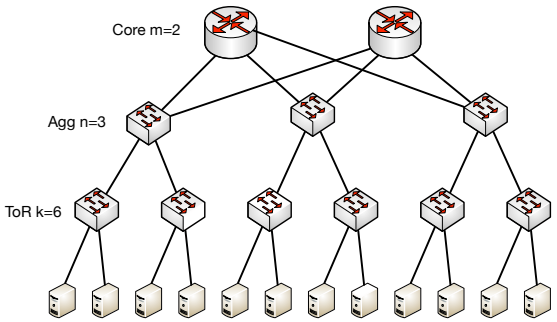


Fig. 5. Data plane topology.

legitimate flows with a low rate, such as 30pps (packets per second).

B. Flooding attack effectiveness

To present the effectiveness of the buffer overflow attacks in SDNs, we simulate an attack process with an average attack rate and record the ingress OVS buffer size under the flooding attack. The used buffer size $B_{ingress}^{used}$ of ingress OVS is used as the metric which is represented flooding attack effectiveness.

To represent the effectiveness of flooding attack, the first ingress OVS's buffer size is recorded based on OpenFlow scheme in a short period. Besides, we set the attack average rate as 500pps and generate a simple topology, $m = 10$, $n = 11$ and $k = 22$. Furthermore, we sample 10 packets in 100 packets.

The variation of OVS buffer size under the flooding attack based on OpenFlow scheme is shown in figure 6. We can observe that the buffer size reaches at 256 packets (OVS's default total buffer size) at about 0.8s without any defense approach. However, the duration of flooding attack maybe a longer time than 0.8s that results in the OVS overflow.

C. Defense effects

To study the defense effects of the buffer overflow attacks in SDNs, we also simulate an attacking process in SDNs with an average attack rate. Furthermore, we use the holding time as a metric to measure the flooding attack effectiveness based on PBUF and OpenFlow scheme, respectively. The holding time means that the SDN framework can hold when flooding attack happen. In other words, the flooding attack happens at t. The first OVS buffer overflow is at t' under flooding attack. the holding time hence can be represented as $(t' - t)$.

To measure the impact of the switch's number on defense effect, we vary m from 2 to 10, n from 3 to 11 and k from 6 to 22. The average attack rate is set as 500pps.

The defense effectiveness under 500pps is shown in figure 7. Comparing to the two holding times, we can observe that PBUF can help with mitigating flooding attack. In this experiment, we keep sending flooding packets until one of OVSs buffer is overflowed. As we can observe from figure 7, the holding time in each topology without any defense methods are almost same. Moreover, these holding times are all short. In other words, the SDN framework without any defense methods suffering from the flooding attack can be

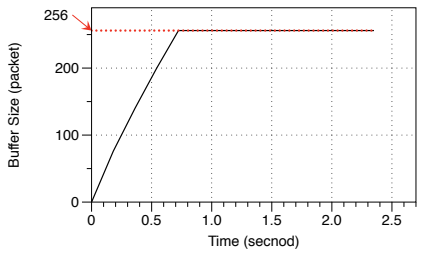


Fig. 6. The ingress OVS buffer size under the flooding attack based on OpenFlow scheme

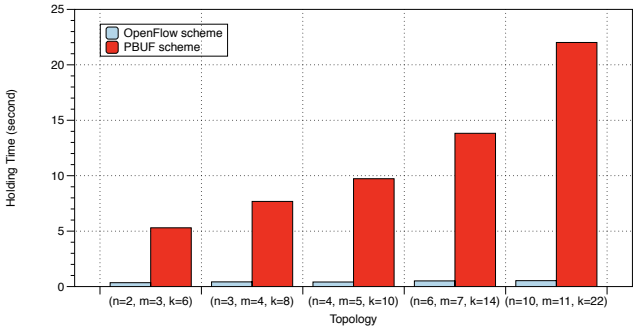


Fig. 7. Holding time based on OpenFlow and PBUF scheme.

compromised in short time regardless of the topology. When we implement PBUF in SDN framework, we can find that these holding times are increased and the red bars is increasing as the topology become larger. Because of the number of idle switches will increase as topology become more larger. Hence, the idle OVS's buffer resource will be larger. The simulation results show that PBUF can mitigate flooding attacks in SDN with the proper configuration.

D. Defense performance

In this subsection, we evaluate the performance of the OVS when PBUF is applied to defend the attack.

In *forwarding packet* module, we use PBUF to forwarding packets. However, some other forwarding methods can also be implemented in *forwarding packet* module. For example, ECMP and random method. These two forwarding methods are represented in algorithm 2 and algorithm 3, respectively. To measure the performance, ECMP and random method will be implemented in *forwarding packet* module as alternative methods. The bandwidth usage and buffer overflow are all used as metrics to measure the performance of packet forwarding. Furthermore, we suppose that the overflow probability means: $number_{overflow}/number_{all}$. $number_{all}$ means the number of all incoming packets; $number_{overflow}$ means the number of entire packet which overflow from the switch and the should be forwarded to controller for processing. Moreover, we use an open source tool *iperf* to measure the bandwidth.

We vary average attack rate from 0pps to 500pps. Furthermore, a topology ($m = 10$, $n = 11$ and $k = 22$) is generated. And the duration of flooding attack is 20s.

Defense performance under different attack rates are shown in figure 8. As shown in figure 8(a), we observe that OpenFlow

Algorithm 2 ECMP

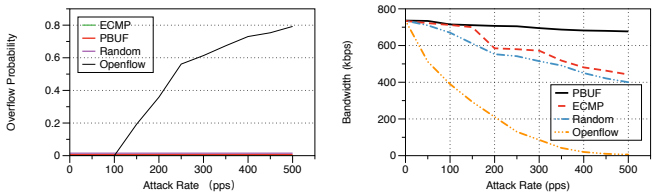
Require: input $packet$
Ensure: Destination $node$
1: dstNode = null
2: **if** currnetNode.bufferNum $<=$ $threshold$ **then**
3: send to controller
4: **else**
5: dstNode = Hash($packet$) {flow hashing}
6: **end if**

Algorithm 3 Random Method

Require: input $packet$
Ensure: Destination $node$
1: dstNode = null
2: **if** currnetNode.bufferNum $<=$ $threshold$ **then**
3: send to controller
4: **else**
5: dstNode = Random($packet$)
6: **end if**

scheme suffer from the overflow as the attacking flow rate increases. The switch's buffer overflow are all 0 when there is no attack. However, under attack, the OpenFlow scheme will result in high overflow probability. Particularly, the overflow probability is larger than 0.5 when the attack rate exceeding 220pps, and could be as high as 0.8 when the attack rate is 500pps. Meantime,we find that the other forwarding packet methods will not bring OVS buffer overflow. We can observe that the other three schemes are identical, which means there are no overflow happening in OVS. For the ECMP method, it statically stripes flows across available paths using flow hashing. Hence, the `table-miss` packets will be forwarded instead of asking for controller. Considering the random method, it forwards these `table-miss` packets randomly instead of asking for controller that there is no overflow in switch. For our heuristic algorithm implemented in *packet forwarding* module, it forward these `table-miss` packets to next idle switch quickly. Hence, our heuristic algorithm also can reach a forwarding efficiency level as the ECMP and random method. Furthermore, PBUF also do not brings switch buffer overflow. Therefore, the threshold works in time.

These forwarding packet methods all would affect the bandwidth, such as link congestion, which are presented in figure 8(b). We note that the OpenFlow bandwidth curve decreases quickly as the flooding attack rate increases and it will run out when the attack rate reach at 500pps. The random method also causes bandwidth congestion. Unlike OpenFlow method, ECMP and random methods consume about half of bandwidth. Although ECMP forwards `table-miss` packets quickly, it static maps of flows to paths does not account for either current network state, with bringing about bandwidth congestion issue. Also like ECMP, the random method does not consider the current network state and chooses the paths randomly. It hence brings about a bandwidth congestion issue. PBUF also brings a congestion issue, whereas it is less than the ECMP and random methods. A heuristic algorithm is implemented in *packet forwarding* module, which not only concerns path's bandwidth but also concern the other issues.



(a) Overflow. (b) Bandwidth.

Fig. 8. Defense Performance Under Different Attack Rates.

E. Overhead

This subsection mainly concerns on CPU usage of controller and processing time of all incoming packets. With respect to controller CPU usage, when under flooding attack, there will be much more random source address packets arriving at ingress switch. Therefore, the controller needs more CPU resource to process these flooding packets. In regard to the processing time, it represents flooding packets in from ingress OVS to destination is measured. Obviously, these `table-miss` packets should take some time and wait in the buffer for processing.

To measure the CPU usage, we generate a topology ($m = 10$, $n = 11$ and $k = 22$) and flooding attacks with average 80pps and 300pps, respectively. Besides, the duration of flooding attack are all set as 20s.

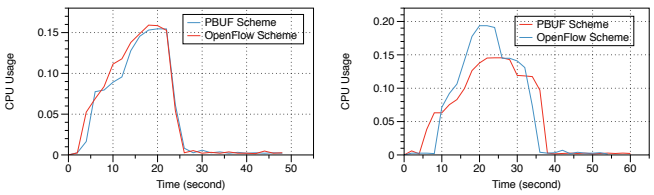
From figure 9(a), firstly, we can find that, under a low average rate 80pps, the flooding attack will all bring high usages of controllers' CPU at about 20s. A high CPU usage means that the application in controller should process the number of remaining packets is large. Since the controller should process these `packet_in` messages and instal new rules on OVSs by `packet_out` messages at the same time. As a result of which, the CPU usage will decrease since all new rules has been installed on OVSs. We can observe that the two curves are almost similar, which means PBUF is not triggered when the attack rate is 80pps.

We also observe the variation of CPU usage under an average 300pps flooding attack in figure 9(b). At first, the two CPU usages increase quickly and reaches to the peaks in seconds. Then the two curves decrease to 0 at different times. Moreover, we find that the peak of OpenFlow scheme is higher than PBUF scheme. Besides, as PBUF limits the rate of `packet_in` messages to controller, the peak values of CPU usage are different.

From Table I, we can find that the average processing times are almost identical under 80pps. Under the 300pps, the OpenFlow scheme and PBUF scheme takes about 36.83s and 38.95s to process these incoming packets, respectively. Comparing to OpenFlow scheme, PBUF takes more time (about 2s) to process these incoming packets. Rather than the OVS buffer overflow, the overhead is acceptable. Therefore, PBUF mitigates flooding attacks with an acceptable low overhead.

VI. Conclusion

In this paper, we point out that, because of the limitation of buffer resource in OpenFlow-enabled switches, SDNs are



(a) CPU Usage Under 80pps. (b) CPU Usage Under 300pps.

Fig. 9. CPU Usage Under 80pps and 300pps Rate

TABLE I
THE AVERAGE PROCESSING TIME UNDER 80PPS AND 300PPS

Average Processing Time	Under 80pps	Under 300pps
OpenFlow	26.39s	36.83s
PBUF	26.51s	38.95s

vulnerable to the flooding attack, which aims to SDN devices. Hence, we propose a defense approach which uses idle OVS resource in the whole SDN framework to mitigate flooding attack. We attempt to dynamically find idle resources when the local resources are used up, to improve the resistance of the network. Hence, we implement *buffer management* and *packet forwarding* modules in OVS, respectively. The experiments confirm that we can remarkably improve the SDNs capacity of defending against the flooding attack. Basing on these experiment results, we also provide theoretical guidance on how to estimate OVS buffer size against buffer overflow. In the future, we expect to find a method to differentiate the attacking flows from legitimate ones in real time, so that we can drop the attacking traffic before they enter further in the network, thus to solve the problem fundamentally.

ACKNOWLEDGMENT

This work is supported by the National Key Research and Development Program of China (2016YFB0800102, 2016YFB0800201, 2017YFB0803205), the National High Technology Research and Development Program of China (2015AA016103), the Key Research and Development Program of Zhejiang Province (2017C01064, 2017C01055), the Program for Key Science and Technology Innovation Team of Zhejiang Province (2013TD20) and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-defined internet architecture: decoupling architecture from infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 43–48.

[2] H. Wang, L. Xu, and G. Gu, "Floodguard: A dos attack prevention extension in software-defined networks," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 239–250.

[3] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 413–424.

[4] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 165–166.

[5] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for openflow networks," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 121–126.

[6] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 87–98, 2014.

[7] J. M. Dover, "A denial of service attack against the open floodlight sdn controller," Dover Networks LCC, Edgewater, MD, USA, 2013.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 254–265, 2011.

[9] C. Lin, C. Wu, M. Huang, Z. Wen, and Q. Cheng, "Adaptive ip mutation: A proactive approach for defending against worm propagation," in *Reliable Distributed Systems Workshops (SRDSW), 2016 IEEE 35th Symposium on*. IEEE, 2016, pp. 61–66.

[10] F. Chen, C. Wu, X. Hong, Z. Lu, Z. Wang, and C. Lin, "Engineering traffic uncertainty in the openflow data plane," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.

[11] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A robust, secure, and high-performance network operating system," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014, pp. 78–89.

[12] D. Kreutz, F. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 55–60.

[13] Y. Jiang, "A basic stochastic network calculus," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 123–134, 2006.

[14] F. Ciucu and J. Schmitt, "Perspectives on network calculus: no free lunch, but still good value," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, August, pp. 311–322.

[15] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen, "Scotch: Elastically scaling up sdn control-plane using vswitch based overlay," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 403–414.

[16] A. Canteaut, C. Lauradoux, and A. Seznec, "Understanding cache attacks," Ph.D. dissertation, INRIA, 2006.

[17] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based load balancing for fast datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 465–478, 2015.

[18] M. Calder, R. Miao, K. Zarifis, E. Katz-Bassett, M. Yu, and J. Padhye, "Don't drop, detour!" *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 503–504, 2013.

[19] C. Hopps, "Analysis of an equal-cost multi-path algorithm," RFC Editor, 2000.

[20] M. S. Kang and V. D. Gligor, "Routing bottlenecks in the internet: Causes, exploits, and countermeasures," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 321–333.

[21] Y. Jiang and Y. Liu, *Stochastic Network Calculus*. Heidelberg: Springer, 2008, vol. 1.

[22] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1629–1637.

[23] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4 - programming protocol-independent packet processors." *Computer Communication Review*, 2014.

[24] C. Lin, C. Wu, M. Huang, Z. Wen, and Q. Zheng, "Performance evaluation for sdn deployment: an approach based on stochastic network calculus," *China Communications*, vol. 13, no. Supplement, pp. 98–106, 2016.