Detecting Saturation Attacks Based on Self-Similarity of OpenFlow Traffic

Zhiyuan Li, *Member, IEEE,* Weijia Xing, Samer Khamaiseh, and Dianxiang Xu, *Senior Member, IEEE*

***Abstract*—As a new networking paradigm, Software-Defined Networking (SDN) separates data and control planes to facilitate programmable functions and improve the efficiency of packet delivery. Recent studies have shown that there exist various security threats in SDN. For example, a saturation attack may disturb the normal delivery of packets and even make the SDN system out of service by flooding the data plane, the control plane, or both. The existing research has focused on saturation attacks caused by SYN flooding. This paper presents an anomaly detection method, called SA-Detector, for dealing with a family of saturation attacks through IP spoofing, ICMP flooding, UDP flooding, and other types of TCP flooding, in addition to SYN flooding. SA-Detector builds upon the study of self-similarity characteristics of OpenFlow traffic between the control and data planes. Our work has shown that the normal and abnormal traffic flows through the OpenFlow communication channel have different statistical properties. Specifically, normal OpenFlow traffic has a low self-similarity degree whereas the occurrences of saturation attacks typically imply a higher degree of self-similarity. Therefore, SA-Detector exploits statistical results and self-similarity degrees of OpenFlow traffic, measured by Hurst exponents, for anomaly detection. We have evaluated our approach in both physical and simulation SDN environments with various time intervals, network topologies and applications, Internet protocols, and traffic generation tools. For the physical SDN environment, the average accuracy of detection is 97.68% and the average precision is 94.67%. For the simulation environment, the average accuracy is 96.54% and the average precision is 92.06%. In addition, we have compared SA-Detector with the existing saturation attack detection methods in terms of the aforementioned performance metrics and controller's CPU utilization. The experiment results indicate that SA-Detector is effective for the detection of saturation attacks in SDN.**

Index Terms*—*software-defined networking*, *security*, *saturation attack*, *anomaly detection*, *self-similar characteristic of OpenFlow traffic

I. Introduction

SOFTWARE-defined networking (SDN) is a new networking paradigm that has revolutionized network architectures [1]. Nowadays, there have been proposals to separate control and data planes with an open interface between them. OpenFlow [2] is the first widespread adoption of such an interface, allowing for centralized management of global network behavior and load balancing.

Z.Y. Li is with the Department of Cybersecurity, Jiangsu University, Zhenjiang, Jiangsu, 212013, China, e-mail: lizhiyuan@ujs.edu.cn.

W.J. Xing is with the Department of Computer Science, University of California Davis, Davis, CA 95618, USA, e-mail: wjxing@ucdavis.edu.

S. Khamaiseh is with the Department of Computer Science, Boise State University, Boise, ID 83725, USA, e-mail: samerkhamaiseh@boisestate.edu.

D.X. Xu is with the Department of Computer Science and Electrical Engineering, University of Missouri - Kansas City, Kansas City, MO 64110, USA, e-mail: dxu@umkc.edu.

Manuscript received Oct 1, 2018.

The security implications of SDN, however, can be a barrier to the wide applications of SDN [3], [4]. For example, Yoon et al. [4] have identified 12 attack vectors with respect to the data plane, the control plane, the control channel, and the application layer due to the design and implementation weaknesses of the existing SDN controller platforms. One of the notable attacks studied by multiple research groups is the control plane saturation attack through SYN flooding [4]–[7]. Different from traditional computer networks, packet forwarding decisions in SDN are implemented by frequent communications between the data and control planes via the OpenFlow communication channel. Because the forwarding of unknown packets is handled by the control plane, malicious hackers may exploit SYN flooding to counterfeit a large number of unknown packets such that the forwarding functionality leads to packet-in flooding of the control plane. When the packet-in flooding exhausts the system resources of the control plane, the normal delivery of packets is interrupted. Currently, the main techniques for dealing with SYN flooding based saturation attacks include recycling of the oldest half-open TCP, SYN cookies, and SYN proxies [4]–[6]. These countermeasures, however, are unable to handle other types of saturation attacks beyond SYN flooding. In fact, our research has shown that saturation attack may also result from the following five types of outside attacks: IP spoofing, ICMP flooding, UDP flooding, other types of TCP flooding (RST and FIN), and port scan. These attacks, either alone or combined, can lead to packet-in flooding of the control plane, flow-rule flooding of the data plane [8], or both. They can interrupt the normal delivery of packets in the data plane and even disable the OpenFlow channel between the data and control planes. As we have demonstrated these attacks in both physical and simulation SDN environments, they impose serious security risks.

To detect the aforementioned saturation attacks in SDN, this paper presents a lightweight method, called SA-Detector, that monitors the OpenFlow traffic between the control and data planes, but does not inspect the contents of the traffic. SA-Detector builds upon the results of investigation into the self-similarity degree of OpenFlow traffic, which is a unique feature of SDN. Self-similarity, pioneered by Benoit B. Mandelbrot [9], describes the phenomenon where a certain property of an object, such as an image and a time series of network traffic, is preserved with respect to scaling in space and/or time. Self-similarity of network flows is a statistical property in time scale that can be measured by the notion of Hurst exponent [10]. Our research has shown that the normal OpenFlow traffic has a low degree of self-similarity due to the

unique SDN/OpenFlow architecture. However, the OpenFlow traffic during the occurrences of saturation attacks has a higher degree of self-similarity. As such, SA-Detector exploits the Hurst exponents of normal and abnormal OpenFlow traffic to determine anomaly of OpenFlow traffic. It is worth pointing out that, the existing research has applied self-similarity to intrusion detection for traditional networks [11]–[14] by using the Hurst exponent of normal traffic as the only indicator. This paper is the first effort to investigate the self-similarity degree of OpenFlow traffic, and use the Hurst exponents of both normal and abnormal traffic to detect saturation attacks.

To evaluate our approach, we have conducted extensive experiments by using both physical and simulation SDN environments with various time scales, network scales, Internet protocols, network applications, and traffic generation tools. For the physical SDN environment, we have collected 222 samples with a total duration of more than 260 hours of normal OpenFlow traffic. For the simulation environment with different numbers of nodes and various network topologies, we have collected 284 samples totaling more than 280 hours of normal OpenFlow traffic. In each environment, we have performed 63 saturation attacks that cover all combinations of the six methods: IP spoofing, ICMP flooding (Ping of Death), UDP flooding, SYN flooding or LAND attack, SARFU (TCP) flooding, and port scan. Each attack has successfully made the control plane and/or the data plane saturated as demonstrated by the fact that the OpenFlow vSwitch is no longer responsive and/or the controller has thrown an exception. The results have shown that, the average accuracy of detection is 97.68% for the physical environment and 96.54% for the simulation environment; the average precision is 94.67% for the physical environment, and 92.06% for the simulation environment. Moreover, we have compared SA-Detector with the existing methods in terms of aforementioned performance metrics and controller's CPU utilization. The experiment results indicate that SA-Detector is highly effective for detecting saturation attacks.

The remainder of this paper is organized as follows. To make the paper self-contained, Section II gives a brief introduction to SDN and OpenFlow before reviewing the related work. Section III describes the saturation attacks that can be realized with different methods. Section IV discusses the proposed detection method based on the different self-similarity characteristics of normal and abnormal OpenFlow traffic. Section V presents our empirical studies. Section VI concludes this paper.

II. Background and Related Work

A. SDN/OpenFlow Overview

SDN breaks the vertical integration by decoupling the network control logic (control plane) from the underlying routers and switches that forward the network packets (data plane). As a result, network switches have become packet forwarding devices and the control logic component is implemented by the centralized controller. The network functions, such as firewall and network management, are deployed in the controller system as its applications. The network applications interact with the control plane by using the northbound API, such as RESTful API. The interactions between the data and control planes depend on the standard southbound API, such as the OpenFlow protocol.

In an Openflow-based SDN environment, each network device, called OpenFlow vSwitch, needs to maintain a set of flow tables. Each flow table contains a series of flow rules installed by the control plane, which instructs the OpenFlow vSwitch on how to handle the incoming packets. The flow rules can be either preinstalled (proactively), or installed on-demand (reactively) by the connected controller. In SDN, the reactive and proactive flow rules installment methods usually coexist. The reactive method can automatically process packets between the data and control planes according to the global network topology. A network administrator can use the proactive method to add a static flow rule into an OpenFlow vSwitch. In the former case, when the OpenFlow vSwitch receives a packet from a host, it will first check the packet header to determine whether or not it matches any flow entry in the pipeline of flow tables. If there is no match, it will encapsulate the first arrived packet of this flow or the packet header as an OFPT_PACKET_IN message and send it to the controller. Once the controller receives the OFPT_PACKET_IN message, it will reply an OFPT_FLOW_MOD message to the OpenFlow vSwitch. The OpenFlow vSwitch installs a new rule for this network flow and then forwards the remaining packets of this flow according to the new flow rule. In this paper, we use the hybrid method of proactive and reactive flow rules installment to detect and mitigate saturation attacks as a static network configuration is inadequate for dealing with different types of saturation attacks.

B. Detection and Countermeasure of Saturation Attack

Saturation attack is a kind of resource exhaustion attack on the control plane, data plane, or both. It is caused by packet-in flooding against the controller. In the existing research [6], [7], [15]–[20], saturation attack is mainly implemented by launch SYN flooding on benign hosts in SDN so as to produce packet-in flooding. Several methods [15]–[18] have been proposed to mitigate SYN flooding attack. Implemented on the data plane, AVANT-GUARD [15] and LineSwitch [16] change the time sequences of OpenFlow protocol in a way similar to the man-in-the-middle attack. The modified SDN system can be unstable, and degrade the forwarding capability of an OpenFlow vSwitch. They monitor the ongoing SYN connections and block the malicious requests. Implemented on the control plane, OPERETTA [17] and SLICOTS [18] can mitigate the SYN flooding attack against web servers. They monitor the ongoing SYN connections and block the malicious requests. A main limitation is that they cannot differentiate the normal and abnormal SYN packets. They block the SYN requests when the total number of the packets is greater than or equal to the predefined threshold. Moreover, the above methods are only suitable for SYN flooding attack. They may not be able to deal with other types of saturation attacks, such as UDP flooding. It is also inaccurate to use a predefined number of packets as a criterion for distinguishing attack traffic from the normal traffic.

Both FloodGuard [6] and FloodDefender [7] are an extension to the prevention of SYN flooding attack. They are protocol-independent prevention methods against different types of attack traffic (e.g. TCP-based flooding attacks, UDP-based flooding attacks or other flooding attacks). They designed a middleware component between a controller platform and its applications, respectively. In FloodGuard, the middleware component consists of three new functional modules: attack detection module, flow rule analyzer module and packet migration module. In FloodDefender, the middleware component includes attack detection module, table-miss engineering, packet filter, and flow rule management. They use the same attack detection module. The attack detection module is used to monitor the rate of OFPT_PACKET_IN messages. Packet-in flooding attack is identified by using a certain threshold. When the rate of OFPT_PACKET_IN messages is greater than a preset threshold, both FloodGuard and FloodDefender will activate the rest modules. For example, FloodGuard writes a wildcard flow rule into the OpenFlow switch and redirects all OFPT_PACKET_IN packets to a data plane cache. FloodDefender detours table-miss packets to neighbor switches using a wildcard flow rule. When these neighbor switches receive the detoured table-miss packets, they will send OFPT_PACKET_IN packets to the controller. When the controller receives OFPT_PACKET_IN messages, the packet filter module will extract four features, such as packet count, byte count, asymmetric packet count and asymmetric byte count from one flow entry. After that, they use support vector machine (SVM) to detect normal or attack packets. The common deficiency of FloodGuard and FloodDefender lie in the fact that using OFPT_PACKET_IN message rate is ineffective for the detection of these attacks because normal bursty traffic and abnormal flooding traffic may have similar OFPT_PACKET_IN packets arrival rates [5]. In other words, the normal bursty traffic will wrongly trigger the traffic filter and migration modules. References [19], [20] also use SVM to detect the anomaly. However, the SVM classifier will spend two weeks or more to identity the normal or attack traffic. The learning time is too long for packet_in flooding attacks detection and defense. And the definitions of recall rate and false-positive rate in [7] are inconsistent with standard definitions in machine learning. Hence, its so-called detection results are not widely accepted.

C. Network Traffic Self-similarity for Intrusion Detection

Anomaly detection using self-similarity of network traffic is a lightweight intrusion detection method [11], [12]. Different from signature-based intrusion detection [21], it does not inspect the content of packets. Thus, it is suitable for network environments with large volumes of traffic or encrypted traffic. The existing work has focused on the detection for denial of service (DoS) and distributed denial of service (DDoS) in external network environments, such as Internet, clouds, and wireless networks [13], [14], [22]–[24]. The intrusion detection evaluation datasets produced by MIT Lincoln Lab [25] were commonly used for the empirical evaluation. The datasets include DoS attacks, user to root attacks, remote to local attacks, network scan attacks identified by their signatures. [13], [14], [22] used self-similarity analysis to study novel anomaly detection methods. [13], [14] compared the Hurst exponent of a give traffic sample with the Hurst exponent of normal (non-attack) traffic dataset to determine if it is an attack. The difference in [13] and [14] is that the collected traffic samples are extracted by the byte sizes and the number of packets, respectively. This simple comparison tends to produce high false rates. [22] assumes that all network traffic follows the characteristic of self-similarity and attack traffic is from IP spoofing. Its anomaly detection is based on the Lyapunov Exponent. The above assumptions are not suitable for OpenFlow traffic. [23] compares the Hurst exponents of normal traffic and DDoS attack traffic for intrusion detection in simulated WiMAX network environment. In our previous research, we collected OpenFlow traffic in simulation environments and exploited the Hurst exponent differences between normal and abnormal OpenFlow traffic to detect saturation attacks in SDN [26]. This paper has significantly improved the previous work in the following aspects: (1) analysis of packet arrival events of OpenFlow traffic; (2) extensive experiments with both physical and simulation SDN environments; (3) complexity analysis of the detection algorithms, (4) offline and online implementations of the detection method, and (5) experimental comparisons with the existing approaches.

III. Saturation Attack Model

In this section, we first introduce how the control plane can be saturated by packet-in flooding and how the data plane can be saturated by flow entry flooding, then we describe the model of various saturation attacks that lead to packet-in flooding, flow entry flooding, or both.

A. Packet-in Flooding

When an OpenFlow vSwitch receives a packet, it will first check the packet header to determine whether or not it matches any entry of the flow tables. In the case of no match, it will send an OFPT_PACKET_IN message to the controller. Once receiving the OFPT_PACKET_IN message, the controller replies an OFPT_FLOW_MOD message. The OpenFlow vSwitch then installs a new rule for this network flow. When a large number of OFPT_PACKET_IN packets are continuously forwarded to the controller, the controller will be busy dealing with these packets. Such packet-in flooding may result in exhaustion of the controller's resources such as computing, memory and transmission capabilities. When the resource consumptions are greater than or equal to the controller's maximum capacities, the system cannot provide the expected forwarding service.

The resource exhaustion threat is denoted as (1).

$$\begin{cases} q_c \times N_{pkt\_in} \geq C_{\max}, \\ q_{bm} \times N_{pkt\_in} \geq M_{\max}, \\ q_{bw} \times N_{pkt\_in} \geq B_{\max}. \end{cases} \tag{1}$$

where q_c is the computational resource for one packet processing, q_{bm} denotes the queue buffer memory taken by one packet, q_{bw} is the bandwidth utilization for one packet transmission, N_{pkt_in} is the number of OFPT_PACKET_IN packets,

$C_{\max}$, $M_{\max}$, and $B_{\max}$ are the maximum computational capacity, buffer memory capacity, and channel bandwidth between a controller and an OpenFlow vSwitch, respectively.

Note that there are two types of time constraints on the existence of a flow rule in an OpenFlow vSwitch: soft time vs hard time. For the soft time, if a flow rule has not been used for several seconds (soft time) in an OpenFlow vSwitch, the flow rule will be automatically deleted from the OpenFlow vSwitch. When a flow rule exceeds the default hard time, the flow rule will also be automatically deleted from the OpenFlow vSwitch. As shown in Fig. 1, consider a flow rule between h-1 and h-2 in an OpenFlow vSwitch in the former case. When a saturation attack happens, a large number of fake packets result in congestion between the data and control planes. The flow rule between h-1 and h-2 has not been used in the default soft time, and hence the flow rule will be deleted from the OpenFlow vSwitch. The communication link between h-1 and h-2 is disturbed. For the latter case, suppose that the flow rule between h-1 and h-2 doesn't exist in an OpenFlow vSwitch. When a saturation attack happens, the congestion between the data and control planes makes the normal communication packets keep staying in the waiting queue for a long time. As a result, the flow rule between h-1 and h-2 cannot be generated in the OpenFlow vSwitch, and the hosts h-1 and h-2 cannot communicate with each other.

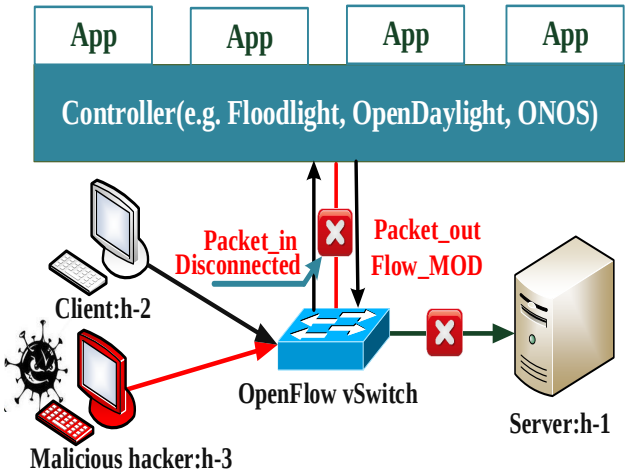


Figure 1. Sample saturation attacks

B. Flow Entry Flooding

When packet-in flooding happens, the controller forwards a large number of OFPT_FLOW_MOD and OFPT_PACKET_OUT packets to the OpenFlow vSwitch. The large number of OFPT_PACKET_OUT packets would result in packet-out flooding. The OFPT_FLOW_MOD packets make the invalid flow rules installed in the OpenFlow vSwitch. This indicates potential resource exhaustion in the data plane, depending on the processing and memory capabilities of the OpenFlow vSwitch. When the invalid flow entries have used up the limited memory space, any other legitimate flow rule will not be installed in the OpenFlow vSwitch. In this case, all the legitimate network packets are refused by the OpenFlow vSwitch.

When the current processing capability reaches up to the maximum capacity of the OpenFlow vSwitch, the read timeout exception between the controller and the OpenFlow vSwitch may happen. As a result, the controller will disconnect OpenFlow vSwitch $\{switchid\}$. $\{switchid\}$ refers to the identification number of the OpenFlow vSwitch in a running controller system. The connected switch will not be able to send any messages or respond to echo requests from the controller. Thus, the OpenFlow communication channel is closed, as shown in Fig. 1.

C. Attack Model

Saturation attack aims to make the control (or data) plane saturated through packet-in flooding (or flow entry flooding). The main approach is to send massive packets that do not match any entry of the flow tables. This in turn leads to large volumes of OFPT_PACKET_IN and OFPT_FLOW_MOD packets. Note that control plane saturation does not necessarily imply data plane saturation, and vice versa. They depend on the respective capacities of the controller and the OpenFlow vSwitch. There are several ways to launch saturation attacks.

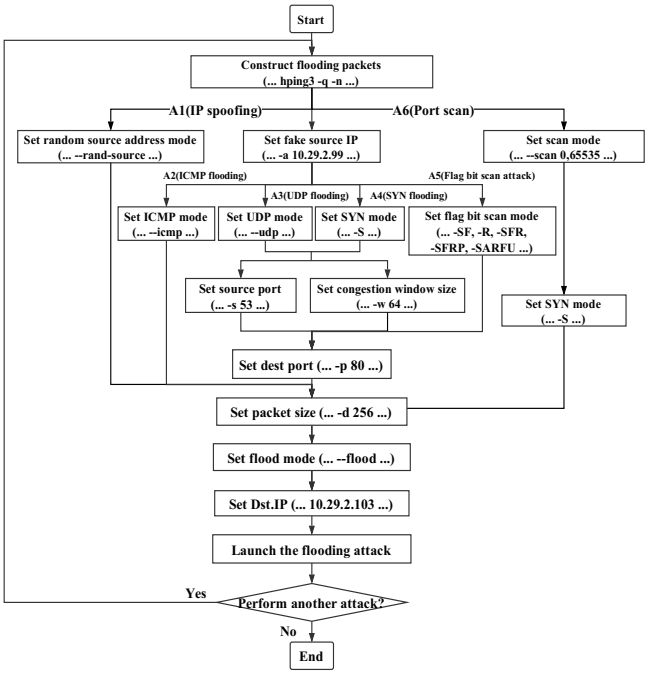


Figure 2. Saturation Attack Graph

Fig. 2 shows the model of all saturation attacks that have been evaluated in this paper. They are implemented by using "hping3" as the packet generation tool. The parameters of hping3 represent packet size, packet generation speed, fake IP addresses, etc [27]. There are six attack methods: IP spoofing, ICMP flooding, UDP flooding, SYN flooding or LAND attack, RST and FIN (TCP) flooding, and port scan. They are denoted as A_1,A_2,A_3,A_4,A_5 and A_6, respectively. Each path from "start" to "end" represent a complete attack with one of the six methods. The loop structure indicates that

a compound attack can be created by combining any number of the attack methods. Such an attack can be executed using multiple hosts or multiple processes on the same host. As will be described in Section V, our experiments have evaluated 63 attacks that represent all possible combinations of the six attack methods. These attacks are programmed as shell scripts and executed automatically. All the attacks are successful in that each has made the control plane or the data plane saturated as evidenced by the consequence that the OpenFlow vSwitch is no longer responsive to requests and/or the controller has thrown an exception, i.e., any two or more equations in (1) hold.

IV. Detection of Saturation Attacks

In this section, we first introduce the OpenFlow traffic arrival model and the concept of self-similarity of OpenFlow traffic. Next, we use Hurst exponent to represent the self-similarity degree of OpenFlow traffic. The method of rescaled range (R/S) analysis is used to estimate the normal Hurst exponent H_n and the abnormal Hurst exponent H_a for the normal and abnormal OpenFlow traffic samples, respectively. Let H_x denote the Hurst exponent of an unlabeled OpenFlow traffic sample. Finally, we propose an anomaly detection method based on Hurst exponent of OpenFlow traffic to detect the occurrence of a saturation attack.

A. Packet Arrival Events Analysis for OpenFlow Traffic

Fig.3 shows the OpenFlow traffic arrival model. A yellow square represents matched packets, and a green square represents mismatched packets. The OpenFlow vSwitch plays "traffic shape" role in the SDN/OpenFlow system. It only forwards a few of mismatch packets to the controller. The most of matched packets are directly forwarded to the destination host by the flow rules in the OpenFlow vSwitch. When a saturation attack happens, abnormal OpenFlow packet arrival rate is significantly different from normal OpenFlow packet arrival rate, as shown in Fig. 4. Fig. 4(a) and 4(b) depict the plot sequences for normal packet counts (number of packets per time unit) in different time units. Each subsequent plot is obtained from the previous one by increasing the time resolution by a factor of 10. Their average packet arrival rates per second are 2.72. Fig. 4(c) and 4(d) show the abnormal OpenFlow traffic on different time units. Both of them are intuitively very "similar". This indicates the self-similarity degree of abnormal OpenFlow traffic is higher than that of normal OpenFlow traffic. Next, we assume that OpenFlow packet arrivals follow a covariance stationary stochastic process. Thus we can measure the self-similarity degree of OpenFlow traffic to identify difference between normal and abnormal traffic.

Suppose $X_t = (x_t, x_{t+1}, x_{t+2}, \cdots)$ is a time series in an OpenFlow traffic sample, where x_t denotes the number or byte size of OpenFlow packets within each time interval (e.g., 10 milliseconds, 1 second or 10 seconds). The autocorrelation function $\rho(k)$ that measures the similarity between X_t and its shifted time series X_{t+k} is as follows:

$$\rho(k) = \frac{\mathrm{cov}(X_t, X_{t+k})}{\sigma^2},\ k \geq 0 \tag{2}$$

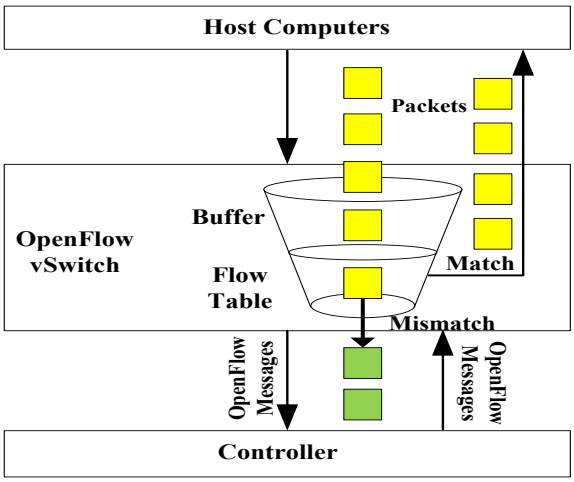


Figure 3. OpenFlow Traffic Arrival Model

where $\mathrm{cov}(X_t, X_{t+k})$ and σ^2 denote the covariance and variance, respectively.

$$\mathrm{cov}(X_t, X_{t+k}) = E[(X_t - \mu)(X_{t+k} - \mu)],\ k \geq 0 \tag{3}$$

$$\sigma^2 = E[(X_t - \mu)^2] \tag{4}$$

$\mu = E[X_t]$ is the mean of X_t. The covariance $\mathrm{cov}(X_t, X_{t+k})$ depends on k, rather than t. According to the self-similarity of network traffic [10], the autocorrelation function $\rho(k)$ is of the following form:

$$k^{-\beta} L(t),\quad k \in \mathbb{N} \tag{5}$$

where $0 < \beta < 1$ and L is slowly varying at infinity. For simplicity, we assume that L is asymptotically constant.

We divide X_t into m non-overlapping sub-time series $X_i (i = 1, 2, 3, \cdots, m)$. For each X_i, if its variance is equal to $m^{-\beta}$ times the variance of X_t and the autocorrelation function $\rho^{(m)}(k)$ of X_i is equal to $\rho(k)$, it is called approximately second-order self-similar. This is formalized by (6):

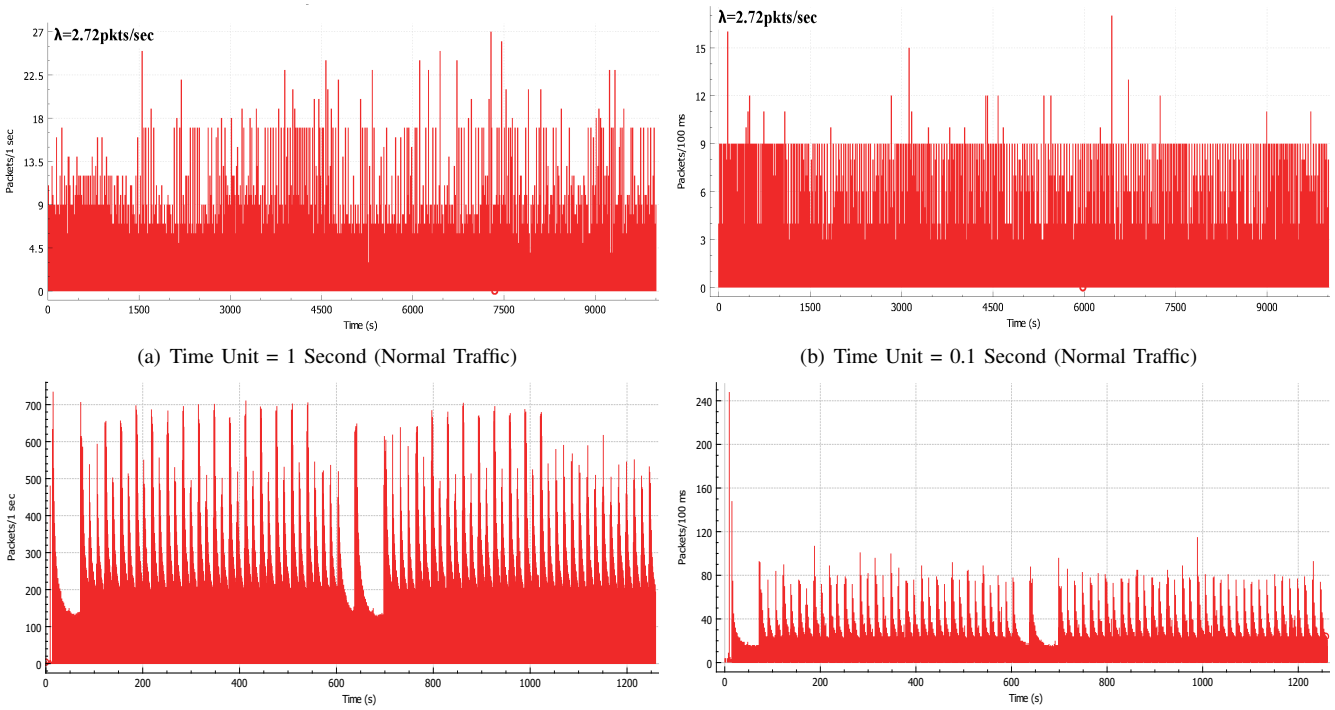
$$\begin{cases} \mathrm{var}(X_i) \sim \sigma^2 m^{-\beta} \\ \rho^{(m)}(k) \sim \rho(k) \end{cases} \tag{6}$$

where '$\sim$' means 'approximately equal'. The autocorrelation function $\rho(k)$ is revised as follows:

$$\rho(k) \sim k^{-(2-2H)} L(t) \tag{7}$$

Self-similarity parameter H in equation (7) is called the Hurst exponent and it is used to measure the self-similarity degree of the stochastic process. Here, $H = 1 - \beta/2$ $(0.5 < H < 1)$. An H value in the range 0.5$\sim$1 indicates a time series with long-term positive autocorrelation, meaning both that a high value in the series will probably be followed by another high value and that the values a long time into the future will also tend to be high. An H value in the range 0$\sim$0.5 indicates a time series with long-term switching between high and low values in adjacent pairs, meaning that a single high value will probably be followed by a low value and that the value after

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2019.2959268, IEEE Transactions on Network and Service Management



(a) Time Unit = 1 Second (Normal Traffic)

(b) Time Unit = 0.1 Second (Normal Traffic)

(c) Time Unit = 1 Second (Abnormal Traffic)

(d) Time Unit = 0.1 Second (Abnormal Traffic)

Figure 4. Packet Arrival Rates for Normal ((a), (b)) and Abnormal ((c), (d)) OpenFlow Traffic

that will tend to be high, with this tendency to switch between high and low values lasting a long time into the future.

B. Anomaly Detection Based on The Hurst Exponent

SA-Detector is based on the self-similarity characteristics of OpenFlow traffic measured by Hurst exponent. There are several models for estimating Hurst exponent, such as R/S analysis, detrended fluctuation analysis (DFA), Whittle's estimator, and wavelet analysis. In this paper, we use the R/S analysis [28] as shown in Algorithm 1.

To estimate the Hurst exponent, we divide a time series with its full size N into different sub-time series. In this paper, we use the function "*logarithmic_n*" to get different sub-time series. The return value of function "*logarithmic_n*" is a data array including $\min_n$, $\min_n \times f$, $\min_n \times f^2$, $\cdots$, $\max_n$, where f is a factor, $\min_n$ represents the minimal value of size n, and $\max_n$ denotes the maximal value of size n. Another choice is the function "binary_n". The return value of function "binary_n" is a data array including $N/2$, $N/4$, $\cdots$. For a sub-time series $X^{(i)}$, the rescaled range R and the standard deviation S are calculated by lines 6∼16 in Algorithm 1. Next, we can repeatedly do the calculation process for the rest of sub-time series, and get a data array $rs(rs[i] = \frac{R(i)}{S(i)})$. Finally, we plot $\log(rs)$ as a function of $\log \kappa$ (κ is also a data array), and use the polynomial curve fitting, called the function of polyfit to fit the straight line. The first parameter of function "polyfit" is to take the log of the data array κ on x-axis, the second parameter of function "polyfit" is to take the log of the data array *rs* on y-axis, and the third parameter of function "polyfit" represents the polynomial-fitting function. In Algorithm 1, we choose the random sample consensus (RANSAC) regressor [29] as the polynomial-fitting function. RANSAC regressor is an iterative method to estimate parameters of a mathematical model from an observed dataset that contains outliers. Unlike the linear regressor, RANSAC regressor makes the outliers no influence on the values of the estimates. For finding the best linear model, the number of iterations is set as 100. Afterwards, the model may be improved by reestimating it using all members of the consensus set. However, if an observed dataset contains more inliers than outliers, RANSAC regressor doesn't reach consensus. Then, we use the least squares polynomial fit instead of the RANSAC regressor to generate a straight line. The slope of the straight line is Hurst exponent H.

In Algorithm 1, the function of time series division is shown in lines 2∼4. Its time complexity is $O(1)$. The function of a data array *rs* calculation for all divided sub-time series is shown in lines 5∼18. The complexity is $O(len(n) \times n[i]^2)$. The function of polynomial curve fitting is shown in lines 19∼20, and its complexity is $O(1)$. Hence, Algorithm 1's complexity is $O(len(n) \times n[i]^2 + 2)$. As $len(n)$ is a constant, it reduces to $O(n[i]^2)$, where $n[i]$ is the size of a sub-time series.

Based on Algorithm 1, we now present the anomaly detection method. We need to calculate the Hurst exponents of normal OpenFlow traffic (H_n^b and H_n^p) and the Hurst exponents of abnormal OpenFlow traffic (H_a^b and H_a^p). Here, H_n^b and H_n^p represent the Hurst exponents of normal OpenFlow traffic in unit of byte size and the number of packets, respectively. Additionally, H_a^b and H_a^p represent the Hurst exponents of abnormal OpenFlow traffic in unit of byte size and the number of packets, respectively.

Algorithm 2 shows the procedure of anomaly detection. The

Algorithm 1: Hurst(X_t)

Input : OpenFlow traffic series X_t
Output: Hurst Exponent H

Declare n, X_i, Y_i, Z_i, rs, κ: data array, $n[i]$: size of a sub-time series;
$N = len(X_t)$;
$n = logarithmic\_n(min_n = 4,\ max_n = 0.1 \times N,\ factor = 1.2)$;
$X_i \leftarrow i$–th sub time series$(x_0, \cdots, x_{n[i]-1})$;
for $(i = 0;\ i < len(n);\ i++)$ **do**
  $\mu = mean(X_i) = \frac{x_0 + x_1 + \cdots + x_{n[i]-1}}{n[i]}$;
  for $(j = 0;\ i < n[i];\ j++)$ **do**
    $Y_i[j] = X_i[j] - \mu$;
  end
  for $(j = 0;\ j < n[i];\ j++)$ **do**
    for $(k = 0;\ k \le j;\ k++)$ **do**
      $Z_i[j] = Z_i[j] + Y_i[k]$
    end
  end
  $R_i = max(Z_i) - min(Z_i)$;
  $S_i = \sqrt{\frac{1}{n[i]} \times \sum_{j=0}^{(n_i-1)} (Y_i[j])^2}$;
  $rs[i] = R_i / S_i$;
end
$\kappa \leftarrow (1, 2 \cdots, len(n))$;
$H = polyfit((log(\kappa),\ log(rs),\ linear))$;
return H

input of Algorithm 2 includes an OpenFlow traffic sample, the normal and abnormal Hurst exponents H_n and H_a, the threshold value δ. The output of Algorithm 2 is the OpenFlow traffic sample is normal or attack. We still use Algorithm 1 to get the average Hurst exponent H_x for this OpenFlow traffic sample. When the Hurst exponent H_x (H_x^b and H_x^p) is greater than or equal to H_a, the OpenFlow traffic sample is identified as a saturation attack. If the Hurst exponent H_x (H_x^b and H_x^p) is less than or equal to H_n, the OpenFlow traffic sample is identified as a normal state. Otherwise, when the Hurst exponent H_x (H_x^b and H_x^p) is less than or equal to $\frac{H_n + \delta \cdot H_a}{1+\delta}$, the OpenFlow traffic sample is identified as a normal state; when the Hurst exponent H_x (H_x^b and H_x^p) is greater than $\frac{H_n + \delta \cdot H_a}{1+\delta}$, the OpenFlow traffic sample is identified as a saturation attack. $H_x \le \frac{H_n + \delta \cdot H_a}{1+\delta}$ originates from $\frac{H_x - H_n}{H_a - H_x} \le \delta$, which means that the distance between H_x and H_n is δ times less than or equal to the distance between H_x and H_a. $\frac{H_x - H_n}{H_a - H_x} > \delta$ means that the distance between H_x and H_n is δ times greater than the distance between H_x and H_a. In other words, if the Hurst exponent H_x of an OpenFlow traffic sample is closer to the normal Hurst exponent H_n, the OpenFlow traffic sample is identified as normal, otherwise abnormal (i.e., occurrence of a saturation attack). The threshold δ is a decision boundary between a saturation attack and a normal state. It depends on the proportion between the distance $|H_x - H_a|$ and the distance $|H_x - H_n|$. The adjustment of δ results in the changes of the false positive, the true positive, the false negative, and the true negative. To adjust δ, we can increase it by 1 when it is greater than 1, or reduce it by 10% when it is in the range of 0 to 1. We can obtain an optimal value through repeated adjustments. In Algorithm 2, the function of anomaly detection based on Hurst exponent is shown in lines 1∼15. The time complexity is $O(1)$.

Algorithm 2: AnomalyDetection [OpenFlow(OF) traffic samples]

Input : An OF traffic sample X_t, H_n^b, H_n^p, H_a^b, H_a^p, threshold value δ
Output: Normal or Attack

$H_x^b = Hurst(X_{t-10ms}^b)$ or $Hurst(X_{t-1s}^b)$ or $Hurst(X_{t-10s}^b)$;
$H_x^p = Hurst(X_{t-10ms}^p)$ or $Hurst(X_{t-1s}^p)$ or $Hurst(X_{t-10s}^p)$;
if $H_x^b \le H_n^b$ *or* $H_x^p \le H_n^p$ **then**
  return Normal;
end
if $H_x^b \ge H_a^b$ *or* $H_x^p \ge H_a^p$ **then**
  return Attack;
end
if $H_n^b < H_x^b < H_a^b$ *or* $H_n^p < H_x^p < H_a^p$ **then**
  if $H_x^b \le \frac{H_n^b + \delta \cdot H_a^b}{1+\delta}$ *or* $H_x^p \le \frac{H_n^p + \delta \cdot H_a^p}{1+\delta}$ **then**
    return Normal;
  else
    return Attack;
  end
end

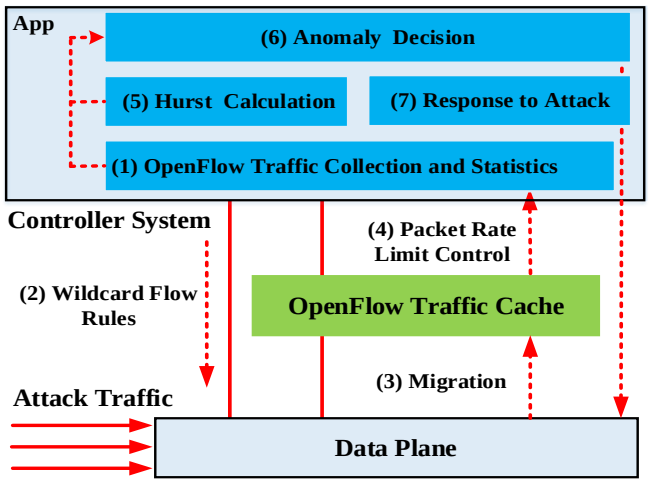
Burst traffic is often classified as normal network communication. It may also exhibit a self-similarity behaviour. In this work, burst traffic originates from two cases: (1) elephant size flow, such as high quality video, and (2) flash crowds, where a large number of users sending requests to a server node simultaneously. The former case may not exhibit a self-similarity behaviour due to one table-miss packet for one flow, as shown in Fig.3. The later case may exhibit self-similarity behaviour that is similar to saturation attacks. The calculated Hurst values are a little bit lower than that of saturation attack. The traffic samples near the anomaly decision boundary tend to have an important impact on the analysis result. We use the feature of proportion between the number of packet_in packets and packet_out packets to obtain a lower false positive rate. That is because attackers usually launch saturation attacks by fake IP addresses. As a result, the number of packet_in packets is much more than the number of packet_out packets.

C. Implementation of SA-Detector

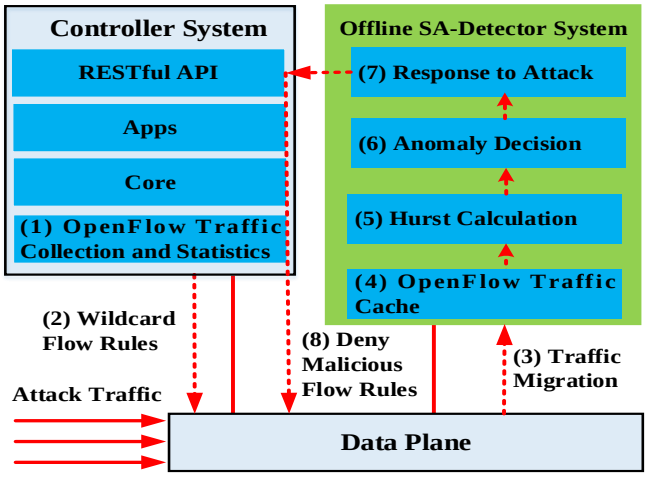
The online and offline implementations of SA-Detector are shown in Fig. 5. The solid line represents the physical connection between two devices, and the dotted line denotes the logical sequences between two functional modules. The online implementation resides in the controller system, whereas the offline implementation is outside the controller system.

1) The online implementation of SA-Detector

Similar to FloodGuard [6], when saturation attacks happen, a wildcard flow rule is used to redirect all OpenFlow traffic



(a) The Online Implementation of SA-Detector



(b) The Offline Implementation of SA-Detector

Figure 5. The Implementation of SA-Detector

to an extra OpenFlow traffic cache between an OpenFlow vSwitch and a controller system to avoid the overload of system resources. The OpenFlow traffic cache is an extra storage device that temporarily caches table-miss packets during the saturation attack. The online implementation of SA-Detector is an application of the controller, as shown in Fig. 5(a).

Steps 1∼4: OpenFlow traffic redirection

When a large number of OFPT_PACKET_IN packets are flowing into the controller system, the controller system has installed a wildcard flow rule into the OpenFlow vSwitch to redirect OpenFlow traffic to the OpenFlow traffic cache. The cache will temporarily store all the table-miss packets. As a result, the saturation attack will not overload the OpenFlow vSwitch and the controller system. The data collection and statistics component in SA-detector dynamically adjusts the rate limit from OpenFlow traffic cache based on the utilization of system resource and the rate of incoming packet_in messages.

Step 5: Hurst exponent calculation

The Hurst calculation component includes methods of the Hurst exponential calculation, such as the rescaled range (R/S) analysis, the aggregated variance method, the Whittle estimator, the variance of residuals, the absolute value method and the wavelet-based method. It receives attack traffic from the OpenFlow traffic cache. The packet and byte counts are calculated in a specified time interval, such as 10ms, 1s and 10s. We need to collect at least 50 statistical results for the detection of an attack in the specified time interval. They are used as input for Hurst exponent calculation. The output of Hurst calculation is the Hurst exponent for a captured OpenFlow traffic sample during a time period.

Step 6: Anomaly decision

We use Algorithms 1 and 2 to determine whether or not saturation attacks are happening. We use the ratio of the number of OFPT_PACKET_OUT to OFPT_PACKET_IN packets to further differentiate the attack traffic and the burst traffic.

Step 7: Response to attack

Once a saturation attack has been determined, the network administrator may enable the function "flow manager" to create and install flow rules into the corresponding OpenFlow vSwitch to block malicious traffic from malicious hosts.

2) The offline implementation of SA-Detector

The offline implementation of SA-Detector is independent of the controller system, as shown in Fig. 5(b). A wildcard flow rule is used to redirect all OpenFlow traffic to a server outside the controller system, as shown in steps 1∼3. The server stores the table-miss packets and analyze the traffic. In the offline version, the steps 4∼7 are implemented to detect and block the malicious traffic. The descriptions for the steps 4∼7 are consistent with step 1 and steps 5∼7 of the online version.

V. Experiments

In this section, we introduce the experiment setup and data collection, describe the measurements and comparisons of self-similarity characteristics of normal and attack OpenFlow traffic samples, present the evaluation results of SA-detector in terms of accuracy, false positive rate, precision, recall, and F_1 score, and compare SA-Detector with the state-of-the-art works in terms of five performance metrics mentioned above and CPU utilization of a controller.

A. Experiment Setup and Data Collection

We have conducted experiments using both physical and simulation SDN environments. The advantage of using a physical SDN environment is the authentic OpenFlow traffic from real-world applications. The network scale and topology are limited, though. Using a simulation environment, we can easily create large scale networks, change the network topology as needed, and generate bursty OpenFlow traffic. Tools such as Mininet can generate OpenFlow traffic that is similar to real OpenFlow traffic. A limitation, however, is that it is not easy to mimic the behaviors of real-world applications.

In our experiment, the physical SDN environment consists of a Pica8 P-3290 OpenFlow vSwitch, a Floodlight v1.2 controller, and six Ubuntu 16.04 hosts. As shown in Fig. 6, we set host h-1 as a web server, h-2, h-5 and h-6 as the clients, and h-3 and h-4 as the malicious nodes. The configurations of all hosts

are shown in table I. The simulation environment is based on Mininet v2.2.1 [30] with a remote controller Floodlight v1.2 [31]. The main parameters are listed in table II. The values of these parameters are obtained from the examples in Mininet v2.2.1. The network structures include star topology, linear topology, and tree topology. Each OpenFlow vSwitch connects with the same number of hosts. The simulation environment allows us to experiment with various network topologies and a varying number of hosts.

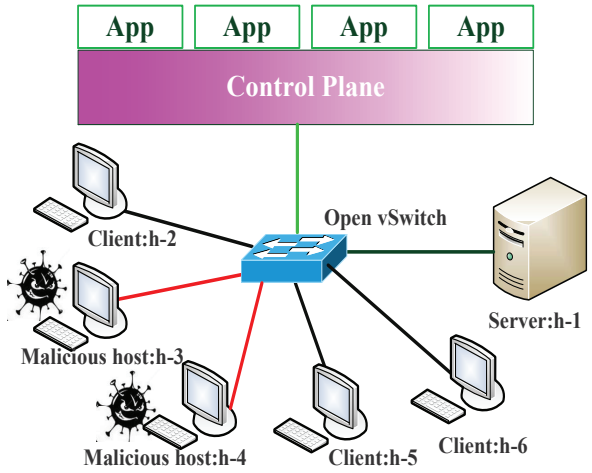


Figure 6. The Physical SDN Topology

Table I
HOSTS IN THE PHYSICAL SDN ENVIRONMENT

Host Name	CPU Info	Memory Info
Controller	Xeon(R) W3503@2.4GHz	8GB
$h-1$	Core(TM) i5-4590@3.3GHz	8GB
$h-2$	Core(TM) i5-3470@3.2GHz	8GB
$h-3$	Xeon E5-1620v4@3.5GHz	8GB
$h-4$	Xeon E5-1620v4@3.5GHz	8GB
$h-5$	Core(TM)2 E8500@3.16GHz	4GB
$h-6$	Core(TM) i7-6700HQ@2.6GHz	8GB

Table II
PARAMETERS OF THE SIMULATION SDN NETWORK

Parameters	Description	Default Values
n_c	Number of controllers	1
n_s	Number of OpenFlow vSwitch	10
n	Number of hosts	200∼400
p_e	Allocated CPU of a host	$\frac{System\ CPU \times 50\%}{n}$
b_w	Link bandwidth	1,000 Mbps
τ	Link delay	$0.5ms$
ε	Packet loss rate of a link	2%
q	Max queue size of OpenFlow vSwitch	10,000

We collected the normal OpenFlow traffic by using various traffic generation parameters, such as different traffic workloads, different packets sizes and different protocols. In the physical environment, we have considered three types of network traffic workloads: (a) low workload where the number of generated packets ranges from 100∼1500 PPS (packets per second), (b) medium workload where the number of generated packets ranges from 1500∼3500 PPS (packets per second), and (c) high workload where the number of packets ranges from 3500∼5000 PPS (packets per second). In the simulation environment, we have only considered one type of network traffic workload where the number of generated packets ranges from 50∼500 PPS (packets per second) due to the limited resources. Packet sizes range from 64 to 512 KB with a variety of network protocols (TCP, UDP, ICMP, DCCP, SCTP, Telnet, VoIP, DNS, ARP, MLD, RTSP, FTP, SFTP, RTP). Among the total traffic data, TCP, UDP, ARP, ICMP, FTP/SFTP, VoIP, Telnet, RTSP/RTP/SCTP, DNS/DCCP, and MLD account for 36%, 27%, 14%, 7%, 4%, 3%, 3%, 3%, 2%, and 1%, respectively.

Table III
SUMMARY OF OPENFLOW TRAFFIC DATASETS

Dataset	Environment	Number of Normal Samples	Number of Abnormal Samples
Dataset1	Physical Environment	222 (Size: 63 GB)	63 (Size: 38 GB)
Dataset2	Simulation Environment	284 (Size: 72 GB)	63 (Size: 37 GB)

For the physical environment, we have collected 222 samples of normal OpenFlow traffic at different time scales (1 hour∼4 hours) by using three tools (D-ITG, TRex, and OSTINATO) that complement each other. There are 200 burst OpenFlow traffic samples in the 222 normal OpenFlow traffic samples. The total size of the captured traffic files is about 63GB and the total duration of the sample traffic is about 260 hours, as shown in the dataset1 of table III. We have used D-ITG (Distributed Internet Traffic Generator) to generate real Internet traffic. D-ITG is an open source platform that is capable of generating IPv4 and IPv6 traffic and replicate different workloads of Internet applications. The core features of D-ITG are provided by ITG-Send and ITG-Receive. ITGSend can generate parallel traffic flows and send them to different ITGRecv instances. ITGRecv is responsible for receiving traffic flows from ITGSend. D-ITG provides the ability to generate multiple unidirectional traffic flows for different protocols such as IPv4, IPv6, TCP, UDP, ICMP, SCTP (Stream Control Transmission Protocol), DCCP (Datagram Congestion Control Protocol), DNS, Telnet, and VoIP. Secondly, we exploit TRex to generate Internet traffic among layers 4∼7. TRex can generate stateful traffic based on traffic templates for simulating multi-application traffic scenarios. Its traffic generation speed is up to 200Gb/sec. We use OSTINATO, a packet crafter and traffic generator, to send packets with different protocols, such as ARP, IPv4, IPv6, ICMP, UDP, TCP, HTTP, SIP (Session Initiation Protocol), RTSP (Real Time Streaming Protocol), MLD (Multicast Listener Discovery Protocol) and NNTP (Network News Transfer Protocol). For the simulation environment, we have collected 284 samples of normal OpenFlow traffic (bursty or non-bursty traffic) at different network scales (200∼400 hosts) with various network protocols and applications, such as ping, Iperf, FTP, SFTP, HTTP, RTP, etc. There are 250 burst OpenFlow traffic samples in the 284 normal OpenFlow traffic samples. The total size of the captured traffic files is about 72GB and the total duration of the sample traffic is about 280 hours, as

shown in the dataset2 of table III. For each of the physical and simulation environments, the time interval of captured packets ranges from 10ms to 10sec.

For both SDN environments, we use Hping3 and LOIC (Low orbit Ion Cannon) to perform 63 saturation attacks according to the model in Fig. 2. The 63 attacks cover all possible combinations of the six attack methods: IP spoofing, ICMP flooding (Ping of Death), UDP flooding, SYN flooding or LAND attack, SARFU (TCP) flooding, and port scan. For instance, there are 6 attacks that use only one of the methods, and 15 attacks that combine two of the methods. The most complex attack combines all of the six methods. Each of the attacks made the control or data plane saturated. The sufficient and necessary condition of each saturation attack is any two equations or more in (1). Only when the two equations or more meet, the abnormal sample is collected. The capture time ranges from 10 minutes to half an hour. The time interval of captured packets ranges from 10ms to 10sec.

For each dataset collected from the physical and simulation environments, we first ramdonly sort the normal samples and the abnormal (attack) samples, respectively. Then we apply K-fold (K=10) cross-validation to the analysis of the dataset. Specifically, 90% of the normal (or abnormal) samples are used to calculate the normal (or abnormal) Hurst exponent, and the remaining 10% of the normal (or abnormal) samples are used to test the anomaly detection method. This process is repeated 10 times to cover all 10-fold configurations of the dataset.

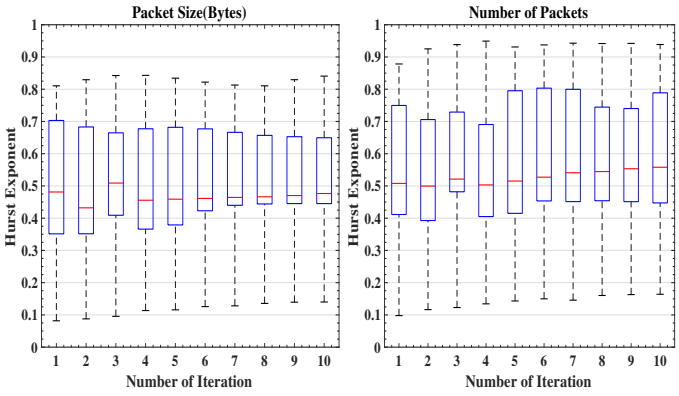


Figure 7. Hurst Exponent H_n Based on 10 times 10-Fold Cross Validation for Normal OpenFlow Traffic in the Physical SDN Environment

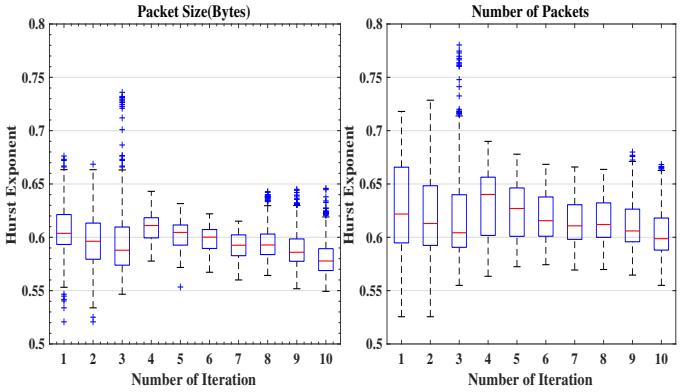


Figure 8. Hurst Exponent H_n Based on 10 times 10-Fold Cross Validation for Normal OpenFlow Traffic in the Simulation SDN Environment

B. Evaluation of Self-Similarity Degrees

The byte and packet counts were extracted for evaluating the self-similarity degrees of OpenFlow traffic. The extracted time scales are 10ms, 1 sec and 10sec. Here, we only show the Hurst exponent for its time scale equal to 10ms for all normal and abnormal SDN/OpenFlow traffic samples due to the page limit.

1) Hurst exponent estimation for normal OpenFlow traffic

Fig. 7 shows the average Hurst exponent estimation results from all normal SDN/OpenFlow traffic samples in the physical SDN environment. The experimental results in Fig. 7 show that the median value of Hurst exponent H_n^b varies from 0.4 to 0.5 in the unit of byte count (packet size) when the number of iteration ranges from 1 to 10. And Fig. 7 shows that the median value of Hurst exponent H_n^p varies from 0.5 to 0.6 in the unit of packet count (number of packets). Next, we use the same method to calculate the average Hurst exponent H_n in the simulation SDN/OpenFlow environment, as shown in Fig. 8. The experimental results in Fig. 8 show that the median value of Hurst exponent H_n^b is around 0.6 in the unit of byte count (packet size). And Fig. 8 shows that the median value of Hurst exponent H_n^p ranges from 0.6 to 0.65 in the unit of packet count (number of packets).

2) Hurst exponent estimation for abnormal OpenFlow traffic

Fig. 9 shows the average Hurst exponent estimation results from all abnormal SDN/OpenFlow traffic samples in the physical SDN environment. The experimental results in Fig. 9 show that the median value of Hurst exponent H_a^b varies from 0.7 to 0.8 in the unit of byte count (packet size). And Fig. 9 shows that the median value of Hurst exponent H_a^p is around 0.8 in the unit of packet count (number of packets). Next, we use the same method to calculate the average Hurst exponent H_a in the simulation SDN/OpenFlow environment, as shown in Fig. 10. The experimental results in Fig. 10 show that the median value of Hurst exponent H_a^b varies from 0.65 to 0.7 in the unit of byte count (packet size). And Fig. 10 shows that the median value of Hurst exponent H_a^p ranges from 0.75 to 0.8 in the unit of packet count (number of packets).

3) Hurst exponent comparison and explanation

The Hurst measurement results for normal OpenFlow signaling traffic between the control and data planes are different from normal Internet traffic. The Hurst measurement results for normal Internet applications traffic is around the value of 0.75 as shown in [10]–[14]. However, the Hurst measurement results for abnormal OpenFlow signaling traffic between control and data planes are consistent with normal Internet traffic. As shown in Fig.3, the Poisson distribution can be used for the normal OpenFlow signaling traffic arrival modeling, but the Poisson distribution cannot accurately describe the process of the abnormal OpenFlow signaling traffic arrival. Hence, the self-similarity characteristics of traffic can be used for the abnormal OpenFlow signaling traffic arrival modeling. Next,

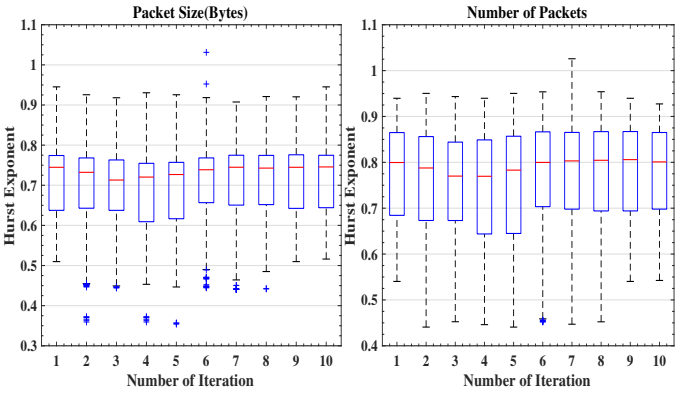


Figure 9. Hurst Exponent H_a Based on 10 times 10-Fold Cross Validation for Abnormal OpenFlow Traffic in the Physical SDN Environment

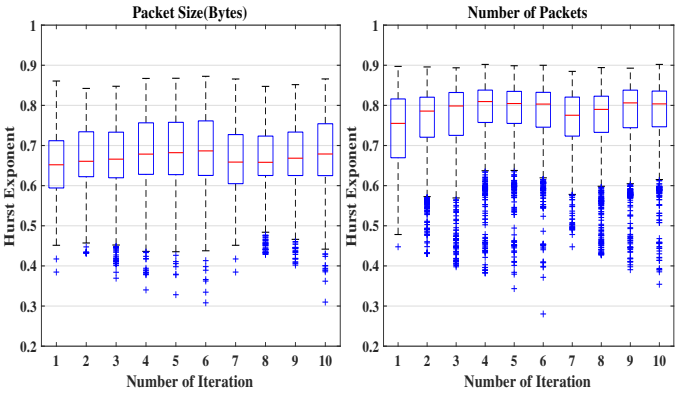


Figure 10. Hurst Exponent H_a Based on 10 times 10-Fold Cross Validation for Abnormal OpenFlow Traffic in the Simulation SDN Environment

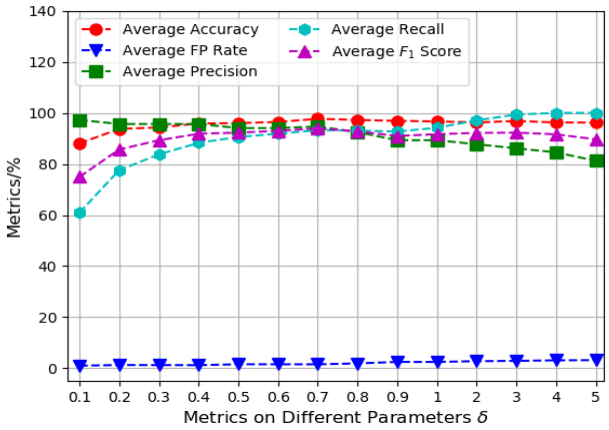
we use the Hurst exponent difference between normal and abnormal OpenFlow signaling traffic to detect the saturation attacks.

C. Evaluation of Anomaly Detection

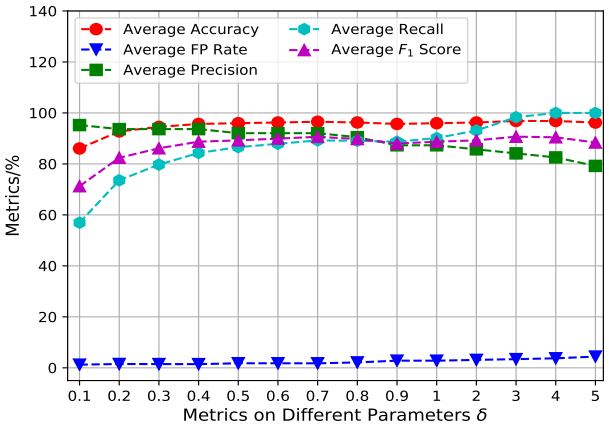
In this section, the offline version of SA-Detector is enabled on the experiments in subsection 1). Additionally, the online version of SA-Detector is enabled on the experiments in subsection 2).

1) Effect of parameter δ and time scales on performance metrics

We use 10-fold cross validation to evaluate the five performance metrics of SA-Detector, including accuracy, false positive rate, precision recall and F_1 score. Fig.11 presents the performance metrics of SA-detector when the time interval is 10ms. (a) and (b) show the five performance metrics trends when δ ranges from 0.1 to 5 in both simulation and physical SDN environments, respectively. As shown in Fig.11(a), we can get the highest F_1 score 93.94% when δ is 0.7 for the physical SDN environment. The corresponding average accuracy is 97.68%, the average false positive rate is 1.57%, the average precision is 94.67%, the average recall is 93.23%. As shown in Fig.11(b), the average accuracy is 96.54%, the average false positive rate is 1.77%, the average precision is 92.06%, the average recall is 89.25% and the F_1 score is 90.63% when δ is 0.7 for the simulation SDN environment. Moreover, when δ is 3, we can get the higher average accuracy 96.86%, the higher average false positive rate 3.41%, the lower average precision 84.13%, the higher average recall 98.33%, and the higher average F_1 score 90.68%. In this case, the accuracy, recall and F_1 score are the highest, but the precision is low and false positive rate is high. Hence, in the simulation SDN environment, the SA-Detector model with δ = 0.7 is considered a better choice from the perspective of tradeoff among the five performance metrics.



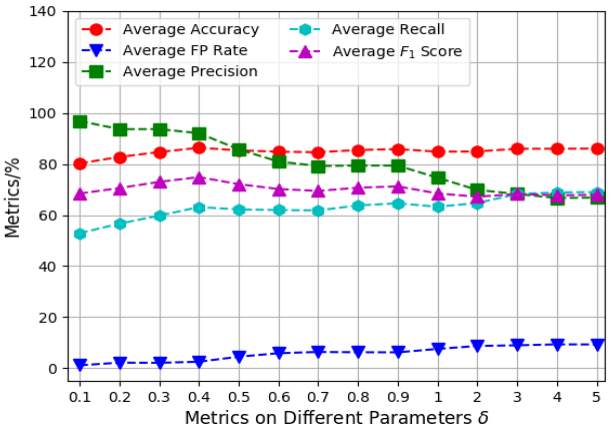
(a) Physical SDN Environment



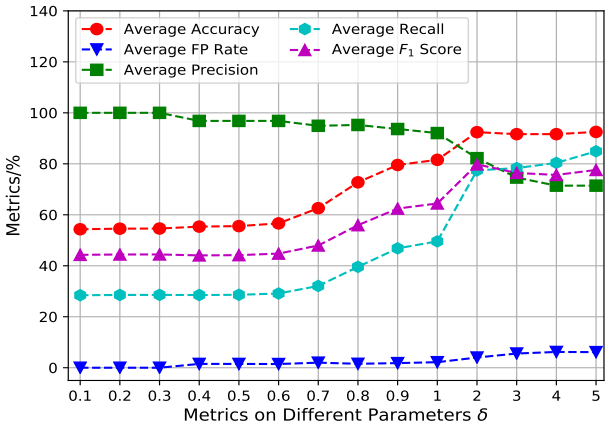
(b) Simulation SDN Environment

Figure 11. Metrics on Different Parameters δ (Time interval: 10ms)

Fig.12 shows the performance metrics of SA-detector when the time interval is set to 1s. Fig.12(a) and 12(b) show the five performance metrics when δ ranges from 0.1 to 5 in both simulation and physical SDN environments, respectively. In the physical SDN environment, we find that the five metrics are optimal when δ is 0.4. The average accuracy is 86.35%, the average false positive rate is 2.59%, the average precision is 92.06%, the average recall is 63.11% and the F_1 score is 74.88%. Although the average recall is higher than δ = 5, the other performance metrics become worse. In the simulation SDN environment, we can get the highest F_1 score 79.753 when δ is equal to 2. The corresponding average accuracy is

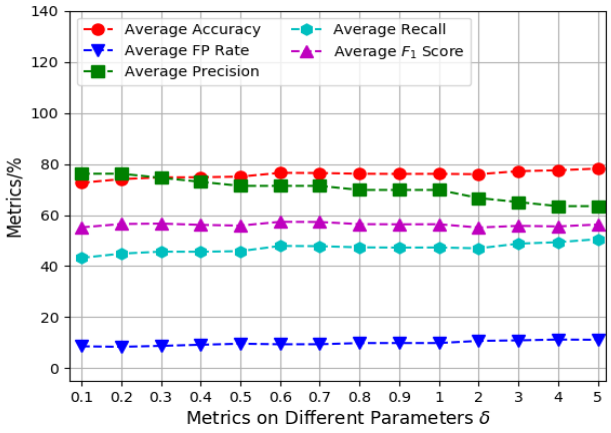


(a) Physical SDN Environment

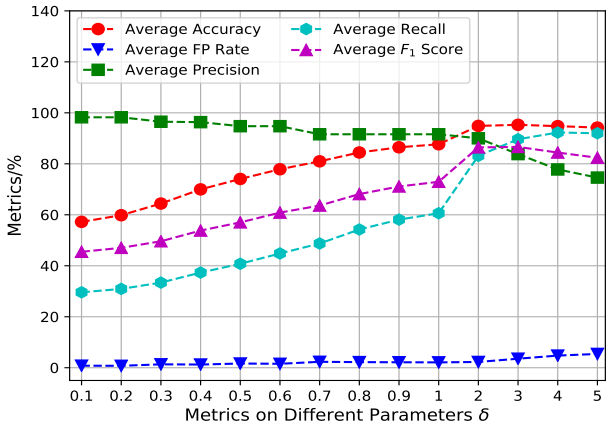


(b) Simulation SDN Environment

Figure 12. Metrics on Different Parameters δ (Time interval: 1s)



(a) Physical SDN Environment



(b) Simulation SDN Environment

Figure 13. Metrics on Different Parameters δ (Time interval: 10s)

92.42%, the average false positive rate is 4.0%, the average precision is 82.22% and the average recall is 77.43%. When the time scale is 1s, most of the five performance metrics are worse than that of the time scale equal to 10ms. That is because the Hurst exponent calculation needs more traffic data to do the polynomial curve fitting. More importantly, we cannot successfully find an optimal δ value to achieve the tradeoff in the five performance metrics when the δ value varies from 0.1 to 5.

Fig.13 shows the performance metrics of SA-detector when the time interval is set to 10s. Fig.13(a) and 13(b) show the five performance metrics when δ ranges from 0.1 to 5 in both simulation and physical SDN environments, respectively. In the physical SDN environment, the F_1 score reaches the highest (57.36) when δ is 0.6. The corresponding average accuracy is 76.53%, the average false positive rate is 9.42%, the average precision is 71.43% and the average recall is 47.92%. In the simulation SDN environment, the F_1 score reaches the highest (86.63%) when δ is 3. The corresponding average accuracy is 95.3%, the average false positive rate is 3.54%, the average precision is 83.81% and the average recall is 89.64%. When the time scale is 10s, most of the calculated five performance metrics are consistent with that of the time scale 1s. Both of them are worse than that of the time scale 10ms. We cannot find an optimal δ value to make the five performance metrics best when the δ value varies from 0.1 to 5.

In summary, the experimental results show that SA-detector has achieved the optimal performance in both physical and simulation SDN environments when the capture time interval is set to 10ms and δ is 0.7. Fig. 11(a) shows the best performance metrics for the physical SDN environment, such as the highest accuracy (97.68%), precision (94.67%), recall (93.23%), F_1 score (93.94%) and the lowest false positive rate (1.57%). Fig. 11(b) also shows the best performance metrics for the simulation environment, such as the highest accuracy (96.54%), precision (92.06%), recall (89.25%), F_1 score (90.63%) and the lowest false positive rate (1.77%). Note that the performance in the physical SDN environment is a little bit higher than that of the simulation environments. That is because the simulation environment accounts for more complex network structures than the physical environment.

There are two advantages for the time scale set to 10ms. One is that we can use more traffic data to achieve polynomial curve fitting and get the accurate Hurst exponent. The other advantage is that the time performance of the Hurst exponent calculation is still acceptable for attack detection and defense when the time scale is equal to 10ms.

2) Comparison with existing approaches

To compare SA-Detector with the existing methods, we performed a family of saturation attacks in three scenarios: (1) an SDN/OpenFlow network with FloodGuard [6], (2) an SDN/OpenFlow network with FloodDefender [7], and (3) an SDN/OpenFlow network with our SA-Detector.

Fig. 14 and Fig. 15 show the comparisons of SA-Detector with FloodGuard and FloodDefender in both simulation and physical SDN environments, respectively. In the physical SDN environment, the attack rate varies from 100 PPS to 5000 PPS. In the Mininet simulation environment, a software OpenFlow vSwitch is dysfunctional by about 500 PPS of table-miss traffic. Hence, we made the attack rate vary from 50 $\sim$ 500 PPS. We can find that the average accuracy, precision, recall and F_1 score go down with attack rate, but the average false positive rate goes up with attack rate. The five metrics of SA-Detector and FloodDefender are better than FloodGuard, and the performance differences of SA-Detector and FloodDefender are just a little bit small. The experimental differences result from the different detection methods for them. FloodGuard is based on certain anomaly threshold (packet_in message rate) and CPU utilization of a controller. The disadvantage of the predefined anomaly threshold is not suitable for the dynamic networks applications. More importantly, we tried to generate the burst traffic in the physical SDN environment. The burst traffic can reach up to 5000 PPS which is close to the effect of saturation attack. Such burst traffic samples made FloodGuard weaker than others. SA-Detector and FloodDefender use the similar detection features, such as packet count and byte count. The difference is that FloodDefender uses the support vector machine (SVM) to classify the normal and abnormal traffic, whereas SA-Detector exploits self-similarity of OpenFlow traffic. In our experiments, We found that the SVM method needs two weeks or more to get the classification results and FloodDefender appears to be more time consuming. FloodDefender in [7] also uses the same method with FloodGuard to improve the attack detection time, but its detection results are similar as FloodGuard. In Fig. 14, the attack detection results of FloodDefender are from the SVM classifier.

Fig. 16 compares the CPU utilization of SA-Detector with FloodGuard and FloodDefender in the physical SDN environment. The saturation attack starts at 0s. We can observe that the CPU utilization of the compared methods increases quickly and reaches a peak at about 1.5s. Then they begin to go down slowly due to corresponding migration rules, such as the installation of the migration flow rules in our SA-Detector and FloodGuard, and the table-miss engineering and packet_in traffic detour in FloodDefender. For SA-Detector, the CPU utilization cannot immediately restore the numerical level before the saturation attack due to the enabled OpenFlow traffic cache. Instead, the utilization maintains at a medium level for some time. At about 6s, the CPU utilization of a

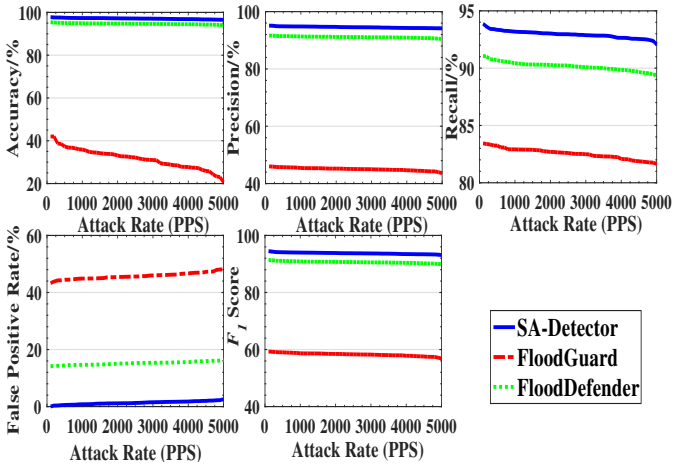


Figure 14. Metrics on Different Attack Rate (PPS) in the Physical SDN Environment

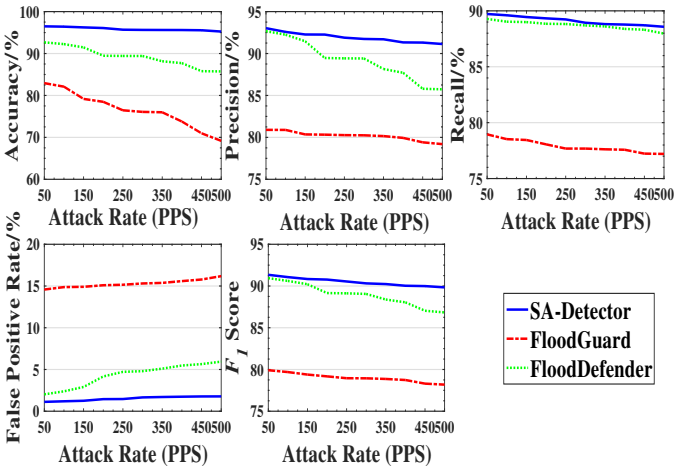


Figure 15. Metrics on Different Attack Rate (PPS) in the Simulation SDN Environment

controller goes back to the initial level due to the response attack. FloodDefender and FloodGuard have shown the similar trend, which is caused by the two-phase filtering in FloodDefender and dataplane cache in FloodGuard, respectively. Note that SA-Detector focuses on detection of saturation attacks. Therefore, the comparison with prevention methods is beyond the scope of this paper.

VI. CONCLUSIONS

We have presented SA-Detector for detecting saturation attacks implemented by various flooding methods in SDN. It is based on the self-similarity differences between normal and abnormal OpenFlow traffic flows and the ratio of the number of OFPT_PACKET_IN packets to OFPT_PACKET_OUT packets. Our extensive experiments with both physical and simulation environments have shown that SA-Detector is effective for detecting these saturation attacks, which pose real security threats to SDN systems. Compared with other detection methods, SA-Detector is better in terms of detection performance metrics and CPU utilization. As a lightweight method that does not inspect the content of packets, SA-

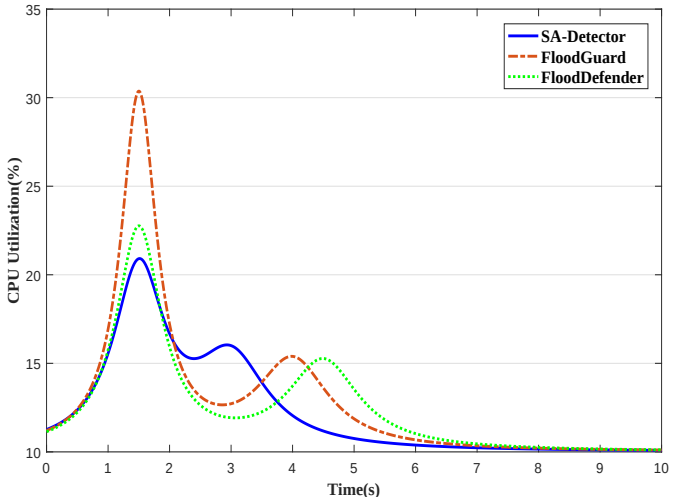


Figure 16. CPU Utilization under Saturation Attack

Detector is suitable for network environments with large volumes of traffic and encrypted traffic. Although this paper has focused on six attack methods for the saturation of control and data planes, SA-Detector serves as a general approach to anomaly detection of OpenFlow traffic. Our future work will focus on the self-similarity optimization method of OpenFlow traffic and malicious node tracking. We also plan to apply machine learning algorithms to intrusion detection by integrating self-similarity degrees with other features of OpenFlow traffic.

REFERENCES

[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] I. Alsmadi and D. Xu, "Security of software defined networks: A survey," *computers & security*, vol. 53, pp. 79–108, 2015.

[4] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow wars: Systemizing the attack surface and defenses in software-defined networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3514–3530, 2017.

[5] D. Kotani and Y. Okabe, "A packet-in message filtering mechanism for protection of control plane in openflow networks," in *Proceedings of the 10th ACM/IEEE symposium on Architectures for networking and communications systems*. ACM, 2014, pp. 29–40.

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

[7] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "Flooddefender: Protecting data and control plane resources under sdn-aimed dos attacks," in *Proceedings of the IEEE International Conference on Computer Communications(INFOCOM)*. IEEE, 2017, pp. 1–9.

[8] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proceedings of the 2nd ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 165–166.

[9] B. B. Mandelbrot, *The fractal geometry of nature*. WH freeman New York, 1983, vol. 173.

[10] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on networking*, vol. 2, no. 1, pp. 1–15, 1994.

[11] Y. Xiang, Y. Lin, W. Lei, and S. Huang, "Detecting ddos attack based on network self-similarity," *IEE Proceedings-Communications*, vol. 151, no. 3, pp. 292–295, 2004.

[12] H. Kwon, T. Kim, S. J. Yu, and H. K. Kim, "Self-similarity based lightweight intrusion detection method for cloud computing," in *Proceedings of the Asian Conference on Intelligent Information and Database Systems*. Springer, 2011, pp. 353–362.

[13] M. Li, "Change trend of averaged hurst parameter of traffic under ddos flood attacks," *Computers & security*, vol. 25, no. 3, pp. 213–220, 2006.

[14] W. H. Allen and G. A. Marin, "On the self-similarity of synthetic traffic for the evaluation of intrusion detection systems," in *Proceedings of the IEEE Symposium on Applications and the Internet*. IEEE, 2003, pp. 242–248.

[15] 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.

[16] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "Lineswitch: tackling control plane saturation attacks in software-defined networking," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1206–1219, 2017.

[17] S. Fichera, L. Galluccio, S. C. Grancagnolo, G. Morabito, and S. Palazzo, "Operetta: An openflow-based remedy to mitigate tcp synflood attacks against web servers," *Computer Networks*, vol. 92, pp. 89–100, 2015.

[18] R. Mohammadi, R. Javidan, and M. Conti, "Slicots: an sdn-based lightweight countermeasure for tcp syn flooding attacks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 487–497, 2017.

[19] A. S. da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn," in *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2016, pp. 27–35.

[20] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran, "Athena: A framework for scalable anomaly detection in software-defined networks," in *Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 249–260.

[21] J. Koziol, *Intrusion detection with Snort*. Sams Publishing, 2003.

[22] A. Chonka, J. Singh, and W. Zhou, "Chaos theory based detection against network mimicking ddos attacks," *IEEE Communications Letters*, vol. 13, no. 9, 2009.

[23] M. Shojaee, N. Movahhedinia, and B. T. Ladani, "Traffic analysis for wimax network under ddos attack," in *Proceedings of the 2nd Pacific-Asia Conference On Circuits, Communications and System (PACCS)*, vol. 1. IEEE, 2010, pp. 279–283.

[24] H. J. Jeong, H. Kim, W. Ahn, J. Oh, D. Lee, S. K. Ye, and J. R. Lee, "Analysis and detection of anomalous network traffic," in *Proceedings of the 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*. IEEE, 2016, pp. 403–408.

[25] K. R. Kendall, "A database of computer attacks for the evaluation of intrusion detection systems," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[26] Z. Li, W. Xing, and D. Xu, "Detecting saturation attacks in software-defined networks," in *Proceedings of the IEEE International Conference on Intelligence and Security Informatics(ISI)*. IEEE, 2018, pp. 1–6 (Accepted).

[27] S. Sanfilippo, "hping3 (8) linux main page," 2005.

[28] J. Beran, R. Sherman, M. S. Taqqu, and W. Willinger, "Long-range dependence in variable-bit-rate video traffic," *IEEE Transactions on communications*, vol. 43, no. 234, pp. 1566–1579, 1995.

[29] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," in *Readings in computer vision*. Elsevier, 1987, pp. 726–740.

[30] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[31] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 7–12.

Zhiyuan Li received his Ph.D. degree in Information and Communication Engineering at Nanjing University of Posts and Telecommunications, China. After the PhD study, he continued his postdoctoral research in Jiangsu High Technology Research Key Laboratory for Wireless Sensor Networks. From September 2016 to August 2018, he was an assistant research professor in the Department of Computer Science at Boise State University, USA. He is currently an associate professor in the School of Computer Science and Telecommunication Engineering at Jiangsu University, China. His research interests include software-defined networks and intrusion detection and prevention. He is a member of the IEEE and ACM and a senior member of China Computer Federation.

Weijiang Xing received his B.S. degree in Computer Science from Huazhong University of Science and Technology University, China. He is currently a graduate student at University of California Davis. He was a visiting research assistant at Boise State University from Fall 2017 to Spring 2018. His research interests are millimeter wave propagation, Agtech and SDN networks.

Samer Khamaiseh received his M.S. degree in Computer Science and Ph.D. degree in Computing with an emphasis on Cybersecurity from Boise State University, USA. He is an assistant professor in the Department of Computer Science at Midwestern State University. Prior to joining BSU in 2015, he was a software engineer at EtQ company. His research interests are software-defined networking, software security, network security, and machine learning. He is a member of the IEEE.

Diangxiang Xu (SM'01) received the B.S., M.S., and Ph.D. degrees in Computer Science from Nanjing University, China. He is a professor in the Department of Computer Science Electrical Engineering at University of Missouri - Kansas City. Prior to joining UMKC in 2019, he was a professor of Computer Science at Boise State University, USA. His research interests include software security, access control, software-defined networking, and software engineering. He is a senior member of the IEEE.