A Detection Method for a Novel DDoS Attack against SDN Controllers by Vast New Low-Traffic Flows

Ping Dong[1], *Member, IEEE*, Xiaojiang Du[2], *Senior Member, IEEE*, Hongke Zhang[1], Tong Xu[1]
[1]School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China
[2]Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
pdong@bjtu.edu.cn, xjdu@temple.edu

***Abstract*—**A Distributed Denial of Service (DDoS) attack against controllers is one of the key security threats of Software-Defined Networking (SDN). The breakdown of a controller may disrupt a whole SDN network. Nowadays, a novel DDoS means is that the attackers may generate vast new low-traffic flows to trigger malicious flooding requests to overload the controllers. It is difficult to prevent this attack, as the attackers may connect to any interface of any switch in an SDN network. In this paper, we propose an effective detection method, which is designed to detect the DDoS attack and to further locate the compromised interfaces the malicious attackers have connected. We first classify the flow events associated with an interface, then make a decision using Sequential Probability Ratio Test (SPRT), which has bounded false negative and false positive error rates. In addition, we evaluate the performance of the proposed method using DARPA Intrusion Detection Data Sets. We also discuss and compare our method to three other detection methods, which are based on the percentage, count, and entropy of the flows, respectively, and demonstrate the superiority of our method in terms of promptness, versatility and accuracy.

Keywords*—DDoS; detection; SDN; controller;*

I. INTRODUCTION

Software-defined networking (SDN) separates a network's control logic from its underlying switches, simplifies network management, and facilitates network evolution. These specific capabilities make SDN deployable in many network environments, from home and enterprise networks to data centers in cloud networks.

The wide variety of use cases make SDN security a serious concern [1]. Out of all known SDN exploits, Distributed Denial of Service (DDoS), which overloads the centralized controllers, is an SDN-specific attack [2] and one of the most severe threats to SDN because its ability to break down a controller can disrupt a whole network.

OpenFlow [3] is a protocol that standardizes how SDN switches communicate with an SDN controller. In OpenFlow, the switches will typically ask a controller, using packet-in messages, to obtain new flow rules for any data flows they do not know how to handle. However, a novel DDoS attack is such that, with a large volume of new data flows generated by the malicious attackers, many packet-in messages may be generated by the switches and aggressively sent to the controller, which may exhaust the resources or even lead to a failure of the controller [4].

This novel DDoS attack against SDN controllers has some specific features and can hardly be detected by the traditional DDoS defense methods:

First, it is difficult for each of the switches to detect the malicious flows. Initially, The attackers may locate in different subnets that connect to different switches, so that the traffic passing each switch may be low. Additionally, no matter how heavy the traffic of a new flow is, only the first few packets of the flow will be encapsulated in the packet-in messages and sent to the controller. Thus, the attackers will prefer to use low-traffic flows to trigger the attack.

Second, it is difficult for a controller to judge whether it is under a DDOS attack based only on the number of incoming queries, since a flooding of queries may result from burst nonmalicious flows.

Third, this attack is a kind of reflection-based flooding in which the attackers do not send the malicious flows directly to a controller. Instead, the attackers may generate the flows based on any protocol (TCP, UDP, et al.) to trigger the attack against a controller. The special method leads to some specific features that differ from those of the known DDoS attacks, such as ICMP flood, TCP SYN flood, and HTTP flood [5]. Correspondingly, existing detection and defense mechanisms [6] are not fully applicable to this attack.

Several recent papers [7] [8] pointed out that the SDN controller is a vulnerable target of DDoS attacks and suggested policing packets destined to the controller. D. Kotani [9] introduced a packet-in filtering mechanism to protect the SDN control plane. The mechanism records the values of packet header fields before sending packet-in messages, then filters out packets that have the same values as the recorded ones. However, if the malicious attackers generate new flows, in which the packets have different values from the recorded ones, the proposed mechanism will become ineffective. S. M. Mousavi [10] proposed an early detection method for the DDoS attacks against the SDN controller. The method is based on the entropy variation of the data flows' destination IP addresses. It assumes that the destination IP addresses are almost evenly distributed in the normal flows, while the malicious flows are destined to a small amount of hosts. However, it is not difficult for the malicious attackers to generate lots of new low-traffic flows, with their destination IP addresses evenly distributed, to overload the controllers.

In this paper, we propose an effective detection method for the novel DDoS attack against SDN controllers by vast new low-traffic flows. Our method has the following advantages:

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

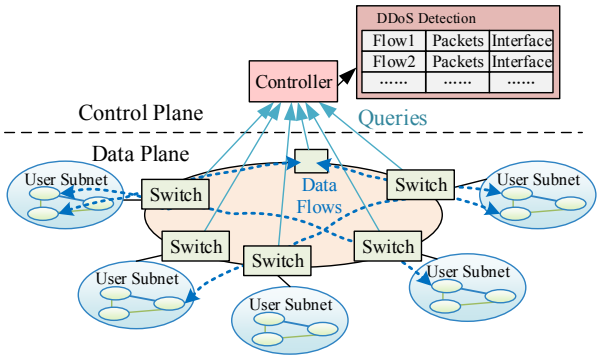

Fig. 1. Network model.

1) Promptness. Benefitting from the Sequential Probability Ratio Test (SPRT), which is a powerful statistical tool, the proposed method can quickly detect the compromised interface after only a small number of successive observations of low-traffic flows.

2) Versatility. The proposed method can detect the DDoS attacks against the SDN controllers, no matter how the malicious flows are generated. That is, our method can detect the malicious attacks triggered by TCP flood, ICMP flood, or even some unknown means leading to the flooding of requests to an SDN controller.

3) Accuracy. The proposed method can distinguish normal burst traffic from malicious flows, and make more accurate decisions than the detection methods based on percentage, count and entropy of the data flows.

The remainder of this paper is organized as follows. The network model for our proposed method and some network assumptions are stated in Section II. The details of our method in terms of flow classification and attack detection are described in Section III. Experimental evaluations and discussions are provided in Section IV, followed by the conclusion.

II. Problem Formulation

In this section, we first state the DDoS detection problem in the SDN network. In particular, we formulate the network model for our proposed method. Then, we describe the network assumptions and the attacker models that we use to evaluate our method.

Fig. 1 illustrates the logical view of the network model. Since a DDoS attack could come from any of the interfaces of any switch, our goal is to detect the attack and locate the potential interfaces that are compromised.

We define an interface of an SDN switch as *a compromised interface* if it is connected by the malicious DDoS attackers or hijacked zombies. Many new low-traffic flows are likely to be injected into the compromised interfaces by the malicious attackers and zombies, with the purpose of triggering a high volume of switch-to-controller messages to overload the SDN controller.

We define F_b^i for $b=1,2,\cdots$ as the successive observations of a random variable F, which is a flow event corresponding to the sequence of flows x_b^i injected into an interface i of an SDN switch. We let F_b^i=1 if flow x_b^i is low-traffic, and F_b^i=0 otherwise. A compromised interface differs from a normal interface in terms of the quantity of incoming low-traffic flows. Specifically, a compromised interface has a higher probability of being injected into low-traffic flows than a normal interface. Formally,

$$Pr\left(F_b^i=1\,|\,H_1\right) > Pr\left(F_b^i=1\,|\,H_0\right)$$

where H_1 is the hypothesis denoting that the interface i is compromised, and H_0 is the hypothesis denoting the interface is normal.

We aim to detect whether an interface i has been compromised.

In the detection, we assume that each switch is capable of obtaining the statistics information of the incoming flows and reporting it to the controller. This can be achieved using several methods. First, OpenFlow itself can monitor per-port and per-rule byte and packet counters. Second, NetFlow [11] provides a way to collect per-flow statistics in switches and sends the flow records to a collector. NetFlow checks the header fields of each packet to see if the packet matches an existing flow. Third, sFlow [12] can estimate the byte and packet counts of flows by fetching real-time packet samples from switches. sFlow captures one packet out of every N packets, and sends the header of the sampled packet to a central collector for flow statistics. We assumed that the flow statistics will pass our DDoS detection modules running on the northbound interface of the controller.

In addition, it is reasonable that attackers tend to generate vast new low-traffic flows to launch an attack in an SDN network because: (1) Only vast packet-in messages may congest a controller. (2) Only new flows can trigger packet-in messages towards a controller. (3) Only low-traffic flows are of high-efficiency for such an attack. A flow with heavy traffic cannot enhance an attack. On the contrary, it will consume more of the attacker's resources than a flow with low traffic, because no malicious packet-in messages will be generated after the delivery of a corresponding flow rule.

III. Detection Based on SPRT

Our method is comprised of a flow classification function and an attack detection function. In this section, we describe these functions in detail.

A. Flow Classification

The flow classification function classifies the data flows in the network. A data flow is considered to be either a *normal flow* or a *low-traffic flow*. Upon receiving a statistic report about a data flow, the flow classification function makes the distinction.

Let c_b^i be the packet counts of flow x_b^i. Let $C_{\max}$ be the threshold to classify a flow. Then, a flow event F_b^i is defined

as a Bernoulli random variable:

$$F_b^i = \begin{cases} 1, & \text{if } c_b^i \le C_{\max} \\ 0, & \text{if } c_b^i > C_{\max} \end{cases} \quad (1)$$

The value of $C_{\max}$ can be obtained and recalibrated by training processes. For example, an offline process can recalibrate $C_{\max}$ if the traffic characteristics change.

In an SDN network, the match fields [3] of each flow table entry consist of the packet headers, the ingress port, and, optionally, other fields. Due to the global perspective and centralized control of SDN, the flow classification function can adaptively classify data flows according to the match fields in the flow tables.

After the classification of a flow, the flow classification function reports the result to the attack detection function.

B. Attack Detection Based on SPRT

The attack detection function analyzes the list of observed events (*normal flows* and *low-traffic flows)* to decide whether an interface is compromised.

In reality, a detection function can make two types of errors: false positive and false negative. As described in Section II, we consider H_1 as a detection of a compromised interface, and H_0 as a normality. False positive means the acceptance of H_1 when H_0 is true, while false negative means the acceptance of H_0 when H_1 is true. To avoid these two errors, we define α and β as the user-specified probabilities of false positive and false negative, respectively. The error rates of the designed detection function should not exceed α and β for false positive and false negative, respectively.

To design an effective detection method that meets the requirements, we adopt SPRT, a powerful statistical tool, to evaluate interface i from the n observed flow events F_b^i, $b = 1,2,\cdots,n$. We define D_n^i as an evaluation of interface i's behavior by the detection function. Let D_n^i be the log-probability ratio considering all n *normal flow* and *low-traffic flow* events noted for interface i. Upon receiving an event F_b^i from the flow classification function, the detection function evaluates:

$$D_n^i = \ln \frac{\Pr\left(F_1^i, \cdots, F_n^i \middle| H_1\right)}{\Pr\left(F_1^i, \cdots, F_n^i \middle| H_0\right)} \quad (2)$$

Assume that each flow event F_b^i is independent and identically distributed, then we have

$$D_n^i = \ln \frac{\Pi_{b=1}^n \Pr\left(F_b^i \middle| H_1\right)}{\Pi_{b=1}^n \Pr\left(F_b^i \middle| H_0\right)} = \sum_{b=1}^n \ln \frac{\Pr\left(F_b^i \middle| H_1\right)}{\Pr\left(F_b^i \middle| H_0\right)} \quad (3)$$

Since F_b^i is a Bernoulli random variable, let

$$\Pr\left(F_b^i = 1 \middle| H_0\right) = 1 - \Pr\left(F_b^i = 0 \middle| H_0\right) = \lambda_0 \quad (4)$$

$$\Pr\left(F_b^i = 1 \middle| H_1\right) = 1 - \Pr\left(F_b^i = 0 \middle| H_1\right) = \lambda_1 \quad (5)$$

where $\lambda_1 > \lambda_0$ because a comprised interface is more likely to be injected into the low-traffic flows to overload an SDN controller.

Intuitively, our SPRT-based detection method can be considered as a one-dimensional random walk. When a low-traffic flow ($c_b^i \le C_{\max}$) is observed, the flow event $F_b^i = 1$ occurs and the walk moves upward one step. When a normal flow ($c_b^i > C_{\max}$) is observed, the flow event $F_b^i = 0$ occurs and the walk moves downward one step. According to Eq. (3), (4), and (5), we get:

$$\begin{aligned} D_n^i &= \begin{cases} D_{n-1}^i + \dfrac{\Pr\left(F_b^i = 1 \middle| H_1\right)}{\Pr\left(F_b^i = 1 \middle| H_0\right)}, & \text{if } c_b^i \le C_{\max} \\ D_{n-1}^i + \dfrac{\Pr\left(F_b^i = 0 \middle| H_1\right)}{\Pr\left(F_b^i = 0 \middle| H_0\right)}, & \text{if } c_b^i > C_{\max} \end{cases} \\ &= \begin{cases} D_{n-1}^i + \ln \dfrac{\lambda_1}{\lambda_0}, & \text{if } c_b^i \le C_{\max} \\ D_{n-1}^i + \ln \dfrac{1-\lambda_1}{1-\lambda_0}, & \text{if } c_b^i > C_{\max} \end{cases} \end{aligned} \quad (6)$$

where $D_0^i = 1$.

We then define the detection function for testing a compromised interface against a normal interface as follows: given two boundaries B and A where $B < A$, on the basis of the log-probability ratio D_n^t, the SPRT for H_0 against H_1 is given as follows:

$D_n^i \le B$: accept H_0 and terminate the test.

$D_n^i \ge A$: accept H_1 and terminate the test.

$B < D_n^i < A$: continue the test process with an additional observation.

It is suggested in [13] that the values of A and B are reasonable to be set to

$$\begin{cases} A = \beta / (1-\alpha) \\ B = (1-\beta) / \alpha \end{cases} \quad (7)$$

From the above equations, we can conclude that four parameters α, β, λ_0, and λ_1 are required by the SPRT-based detection method. Among them, α and β limit the false positive error rate and false negative error rate, respectively, and give two boundaries (A and B) of the one-dimensional random walk. λ_0 and λ_1 are the probability distribution parameters for the flow events, and affect the number of observations required for the detection function to reach a decision (either H_0 or H_1).

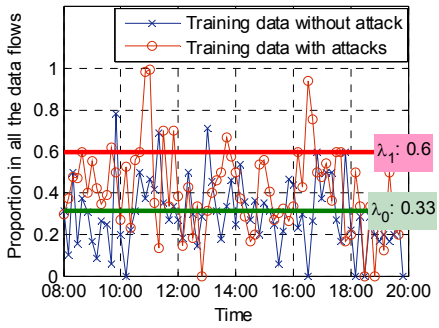

Fig. 2. Proportion of the low-traffic flows in all the flows for the training data without attack (March 4th) and with attack (March 11th), respectively. λ_0 and λ_1 are estimated.

TABLE I DESCRIPTIONS OF SOME OF THE ATTACKS THAT MAY OVERLOAD THE SDN CONTROLLERS

Attacks	Descriptions
neptune	A SYN flood on one or more TCP ports.
smurf	ICMP requests with the victim's spoofed source IP are broadcast to a network, to create a reply flood to the victim.
ipsweep	A surveillance sweep sending ICMP requests to numerous IP addresses to detect which nodes are listening on a network.
portsweep	A surveillance sweep that scans numerous ports to detect which services are open on a single or multiple nodes.

IV. EVALUATION AND DISCUSSION

We chose to use DARPA Intrusion Detection Data Sets to evaluate our proposed method, as they contain both training data without/with attacks and testing data with abundant types of attacks [14], i.e., 201 attack instances of about 56 types of attacks.

A. *Parameters of the Method*

As mentioned in Section II, our SPRT-based method requires four user-defined parameters α, β, λ_0, and λ_1. Among them, α and β are normally small values limiting the desired false positive and false negative rates, which are usually between 0.01 and 0.05 and can be specified by the users of the detection approach. We set $\alpha = 0.01$ and $\beta = 0.02$ in the evaluation.

λ_0 and λ_1 should indicate the probabilities that a low-traffic flow passes through a normal interface and a compromised interface, respectively. Thus, it is reasonable for us to estimate λ_0 and λ_1 using the training data without and with attacks, respectively. The "outside tcpdump data", which provides extensive examples of attacks and background traffic, was used in our evaluation. The data provides all data packets transmitted between the nodes inside and outside of a network.

We classified the flows by counting the number of packets in each flow. If the packet count of a flow was below a designated threshold, the flow in question was classified as a low-traffic flow. Otherwise, it was classified as a normal flow. Afterwards, we calculated the proportion of the low-traffic flows in all the flows to estimate λ_0 and λ_1.

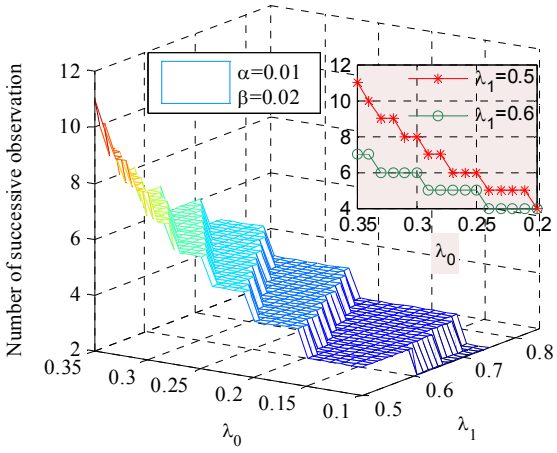

Fig. 3. The minimum number of successive observations of low-traffic flows for detecting a compromised interface.

Without loss of generality, the threshold was set to 3 packets, the packet count of a normal TCP 3-way handshake. The packets with the same four-tuple, i.e., (source IP, source port, destination IP, destination port), were considered to belong to the same flow. The communications between the adjacent interfaces were filtered because they would not trigger packet-in messages towards the controllers.

Fig. 2 illustrates the estimation of λ_0 and λ_1. As Fig. 2 shows, the characters of the flows were similar when there were no DDoS attacks on both days (March 4th and March 11th). However, on March 11th, two attacks named " neptune" and "ipsweep" (listed in TABLE I) occurred at 11:04:16 and 16:36:10, respectively. These attacks lead to vast new low-traffic flows that could overload the controller. Accordingly, we assigned the mean value and the upper boundary value of the normal proportion of the low-traffic flows to λ_0 and λ_1, respectively. Thus, we got $\lambda_0 = 0.33$ and $\lambda_1 = 0.6$.

In the next section, we will discuss how the values of λ_0 and λ_1 affect the performance of our SPRT-based method.

B. *Promptness*

Promptness is an important property of our SPRT-based DDoS detection method. As we learn from Fig. 2, a large volume of low-traffic flows will be injected into a compromised interface when a DDoS attack occurs. This section evaluates the minimum number of successive observations of low-traffic flows to detect a compromised interface when H_1 is true.

To illustrate the minimum number of successive observations required to detect a compromised interface, Fig. 3 shows the number of observations as a function of λ_0 and λ_1. As mentioned in the previous sub-section, we set the false positive rate $\alpha = 0.01$ and the false negative rate $\beta = 0.02$. As shown in Fig. 3, it only takes several observations for SPRT to make a decision. Basically, the greater the difference between λ_1 and λ_0, a smaller successive number of observations is required for our method to reach a detection.

As the DARPA Intrusion Detection Data Sets gave us $\lambda_0 = 0.33$ and $\lambda_1 = 0.6$ in our evaluation, our method can quickly detect a compromised interface after only six successive observations of low-traffic flows.

We only show the minimum number of required successive observations in the above discussion. There is a similar trend for the average number of required observations when H_1 is true.

C. *Versatility*

We evaluated the versatility of our method by submitting it to the data traces with malicious traffic in the DARPA Intrusion Detection Data Sets.

Since we assume that a flow rule corresponds to a four-tuple, the attacks listed in TABLE I may overload an SDN controller. Notably, these attacks can work at different layers and have different principles. For example, "neptune" woks at the transport layer and is a typical denial-of-service attack, while "ipsweep" woks at the network layer and is not usually considered a denial-of-service attack. However, the key characteristic they share is the ability to generate many new flows, which may lead to the overloading of an SDN controller.

Note that the extent of damage an attack can cause depends on the match fields of flow rules. For example, "neptune" can overload a controller when a flow rule corresponds to a four-tuple, but can hardly overload the controller when a flow rule only corresponds to the destination IP address of a flow. Our flow classification function can adaptively classify the flows according to the match fields of flow rules.

Fig. 4 (a) shows the number of total flows on April 5th. Fig. 4 (b) shows all the denial-of-service attacks and a "portsweep" attack used in the datasets in chronological order. Although traditional attack classification methods do not regard "portsweep" as a DDoS attack, it may be used to generate vast low-traffic flows and further trigger the DDoS attacks to overload the SDN controllers. In addition to "portsweep", "smurf" and "neptune" launched DDoS attacks against the SDN controllers on April 5th. As Fig. 4 (c) shows, our method successullfy detected all three of these attacks, even though they work at different protocol layers and have different principles. Additionally the reports return neither false positive errors nor false negative errors for the attacks without a large volume of low-traffic flows.

D. *Accuracy Comparison and Discussion*

For comparison, we discuss three different methods in detecting the DDoS attacks, which are based on the percentage of low-traffic flows, the number of low-traffic flows and the entropy variation of destination IP addresses, respectively. For simplicity, we refer to them as the percentage-based detection (PD), the count-based detection (CD) and the entropy-based detection (ED). They are discussed in the following:

- PD. An interface is considered to be compromised if the percentage of low-traffic flows is higher than a threshold. Unlike in our SPRT-base method, which has bounded false negative and false positive error rates, it is challenging to select a threshold value for the single parameter of PD. A small threshold value may result in the false positive errors, while a large threshold value may result in the false negative errors.

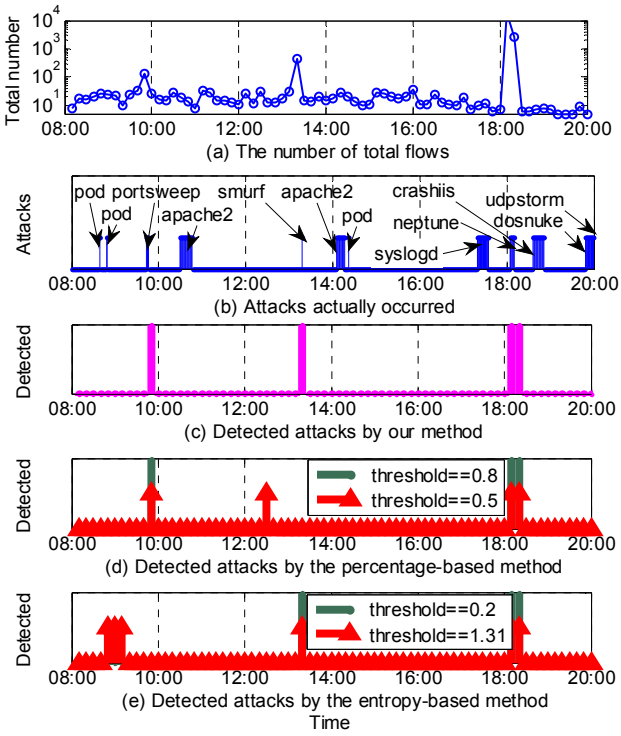

Fig. 4. a) The number of total flows; b) Attacks actually occurred; c) Detected attacks by our method; d) Detected attacks by the percentage-based method; e) Detected attacks by the entropy-based method.

- CD. CD faces the similar problems to PD; that is, it is also challenging to select an appropriate threshold value for CD.

- ED. An interface is considered to be compromised if the entropy of the flows' destination addresses is lower than a threshold. As we state in Section I, false negative errors may occur if the attackers generate vast new flows with the destination IP addresses evenly distributed. At the same time, false positive errors may occur if the destination IP addresses of the normal flows are not evenly distributed.

We evaluated the performance of PD and ED using the same data trace to compare with our method. For comparison, we first divided the datasets into mini-timeslots. The mini-timeslot can be regarded as the hard timeout timer for the flow entries as well as the detection period of our method. A hard timeout timer is the duration after which a flow entry is removed from the flow table in an SDN switch.

Note that it is not necessary for our SPRT-based method to divide the datasets into mini-timeslots, because the method can observe continuously until it reaches a decision. In contrast, an essential component for PD, CD and ED is a window size that is based on a timeslot or number of packets. This is also a reason why the SPRT-based method can make a quicker and more accurate decision than other methods.

Fig. 4 (d) shows the performances of PD with thresholds

0.5 and 0.8, respectively. Note that if the total number of flows is less than 30 per timeslot in our evaluation, the detection is not performed due to the low risk and high randomness of the flows. We can observe that:

First, PD made a false positive error at 12:30 with threshold 0.5. As shown in Fig. 4 (b), no attack really occurred at that time.

Second, PD made a false negative error with both thresholds 0.5 and 0.8. Hence, "smurf" was underreported. PD might have detected "smurf" if we chose a threshold lower than 0.5. However, this would result in more false positive errors than were made under a threshold of 0.5.

Thus, we can conclude that it is challenging to choose a threshold value for the single parameter of PD.

Fig. 4 (e) shows the performances of ED with both thresholds 0.2 and 1.31, respectively. We can observe that:

First, ED made several false positive errors with threshold 1.31, which is the threshold used in [10]. From Fig. 4 (b), we can know that ED reported the "pod" attacks. We say these reports were false positive errors because these "pod" attacks did not launch attacks against the controller. We can find lots of ICMP messages that were sent from a single malicious node to a single host when the "pod" attacks occurred in the datasets. Thus, ED detected it because the entropy was lower than the threshold 1.31 during this timeslot. However, our goal is to detect the DDoS attack against the controller but not the attack against a host. Because the "pod" attack only requires a single flow rule and would not generate many requests to the controller, these reports were false positive reports.

Second, ED made a false negative error with both thresholds 0.2 and 1.31. Hence, "portsweep" was underreported. We learn from the datasets that the destination IP addresses of this attack was almost evenly distributed because the attackers tried to probe many hosts in the subnet. Thus, the entropy value was quite high during this timeslot, although many new flows were generated and vast requests were sent to the controller.

The results prove what we discuss at the beginning of this section, i.e., the ED will be ineffective if the normal flows are not evenly distributed or the malicious flows are evenly distributed. Note that the "normal flows" mentioned here are the flows do not trigger attacks against the controller. ED may be used to detect the denial-of-service attack against a host, such as "pod", but it is another issue and out of the scope of this paper.

V. Conclusions

In this paper, we present an efficient detection method for a novel DDoS attack against SDN controllers by vast new low-traffic flows. The method is based on a powerful statistical tool, SPRT, which has bounded false negative and false positive error rates. The method can detect DDoS attacks and further locate the compromised interfaces the malicious attackers have connected. Evaluations are performed by submitting the method to the data traces with abundant types of attacks in the DARPA Intrusion Detection Data Sets. Compared to the detection methods based on the percentage, count and entropy of the flows, the proposed method has some outstanding properties in terms of promptness, versatility and accuracy.

Acknowledgment

This work was supported in part by the Fundamental Research Funds for the Central Universities (2014JBM004, 2015JBM001), in part by Beijing Higher Education Young Elite Teacher Project (YETP0534), in part by National 973 Program of China (2013CB329100).

References

[1] A. Akhunzada, E. Ahmed, A. Gani, M. Khan, M. Imran, and S. Guizani, "Securing software defined networks: taxonomy, requirements, and open issues," IEEE Communications Magazine, vol. 53, pp. 4, pp. 36-44, 2015.

[2] D. Kreutz, F. M. 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, 2013, pp. 55–60.

[3] Open Networking Foundation, "OpenFlow Switch Specification Version 1.5.1," https://www.opennetworking.org, 2015.

[4] S. Shin, 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, 2013, pp. 165-166.

[5] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 2046-2069, 2013.

[6] W. Eddy, "TCP SYN Flooding Attacks and Common Mitigations," IETF RFC 4987, 2007.

[7] Q. Yan and F. Yu, "Distributed denial of service attacks in software-defined networking with cloud computing," IEEE Communications Magazine, 2015, vol. 53, no. 4, pp. 52-59.

[8] K. Benton, L. J. Camp, and C. Small, "Openflow vulnerability assessment," In Proceedings of the second ACM SIGCOMM workshop on hot topics in software defined networking, 2013, pp. 151-152.

[9] D. Kotani and Y. Okabe, "A packet-in message filtering mechanism for protection of control plane in openflow networks," In Proceedings of the tenth ACM/IEEE symposium on architectures for networking and communications systems, 2014, USA, pp. 29-40.

[10] S. M. Mousavi, "Early Detection of DDoS Attacks in Software Defined Networks Controller," 2014. (Master dissertation, Carleton University Ottawa)

[11] B. Claise, "Cisco Systems NetFlow Services Export Version 9," IETF RFC 3954, 2004.

[12] P. Phaal and M. Lavine, "sFlow version 5," http://www.sflow.org/sflow_version_5.txt, July 2004

[13] A. Wald, Sequential Analysis. J. Wiley & Sons, 1947.

[14] MIT Lincoln Laboratory, "Intrusion detection attacks database," http://www.ll.mit.edu/ideval/docs/attackDB.html.