2015 IEEE Trustcom/BigDataSE/ISPA

An Entropy-Based Distributed DDoS Detection Mechanism in Software-Defined Networking

Rui Wang, Zhiping Jia*, Lei Ju
School of Computer Science and Technology
Shandong University
Jinan, China
Email: jzp@sdu.edu.cn

***Abstract*—Software-Defined Networking (SDN) and OpenFlow (OF) protocol have brought a promising architecture for the future networks. However, the centralized control and programmable characteristics also bring a lot of security challenges. Distributed denial-of-service (DDoS) attack is still a security threat to SDN. To detect the DDoS attack in SDN, many researches collect the flow tables from the switch and do the anomaly detection in the controller. But in the large scale network, the collecting process burdens the communication overload between the switches and the controller. Sampling technology may relieve this overload, but it brings a new tradeoff between sampling rate and detection accuracy. In this paper, we first extend a copy of the packet number counter of the flow entry in the OpenFlow table. Based on the flow-based nature of SDN, we design a flow statistics process in the switch. Then, we propose an entropy-based lightweight DDoS flooding attack detection model running in the OF edge switch. This achieves a distributed anomaly detection in SDN and reduces the flow collection overload to the controller. We also give the detailed algorithm which has a small calculation overload and can be easily implemented in SDN software or programmable switch, such as Open vSwitch and NetFPGA. The experimental results show that our detection mechanism can detect the attack quickly and achieve a high detection accuracy with a low false positive rate.**

Keywords - SDN; OpenFlow; DDoS; Entropy

I. INTRODUCTION

With the explosion of server virtualization and cloud computing, the traditional network architectures are ill-suited to the dynamic computing and storage needs of the current data centers. The Open Networking Foundation (ONF) which is a non-profit consortium has promoted Software-Defined Networking (SDN) [1] as a new norm for networks. SDN has a centralized controller and many switches which are simply forwarding the packets. This architecture decouples the network control plane and data plane enabling the network control to be programmed and the underlying infrastructure to be abstracted for applications and services. By using SDN, enterprises and carriers can build highly scalable, flexible network that adapts to the ever-changing businesses needs.

The OpenFlow protocol [2] is the first standard communications interface defined between the control and forwarding layers of the SDN architecture. Each OpenFlow switch has at least one flow table which contains a set of flow entries; each flow entry consists of match fields, counters and a set of rules applied to forward packets. Based on these predefined match rules, OpenFlow switch uses the concept of flow to identify the network traffic and record its information by the counters. OpenFlow-based SDN technologies enable IT to address dynamic nature of the current application, reduce operations and management complexity.

However, as a promising architecture, SDN is still not completely designed and many aspects need to be improved [22]. The attributes of centralized control and programmability associated of SDN platform also bring many network challenges, as discussed in [12] and [14]. Specially, DDoS attack [15] which is one of the main security challenges to the traditional Internet is still a huge threat to SDN. As a kind of DDoS, DDoS flooding attack usually originates from widely distributed zombies and targets to exhaust victim's bandwidth or resources. SDN has some new security attack points, such as SDN controller, virtual infrastructure and OpenFlow Network. If the attacker gains access to the controller or the SDN switch, he may be capable of aggregating enough power to launch DDoS attacks by using a large number of switches under the controller and malevolently changing the flow table in switch [13]. Nowadays, a lot of people have done researches on DDoS detection and migration in SDN e.g., [16] and [20]. Many of them started to take consideration of the flow recorded feature of the OpenFlow table, such as [17]. By periodically sending the flow tables to the controller, the controller can see the global flow information. Then the anomaly detection algorithm running in the controller can do analysis on these collected data and detect whether DDoS attacks had happend. This may works well when the network is small. However, when the network scale becomes large, the flow collection and statistics processes may overload the control plane [16]. And also the attack response time is depended on the polling time [24]. If we need a quick detection and simply shorten that time, it will increase the number of collected data and aggravate the overhead. Flow sampling technologies, such as sFlow [3], NetFlow [4], have been utilized to reduce this overhead [16]. But it bring a new tradeoff between sampling rate and detection accuracy.

As studied in [24], [26] and [27], more and more researchers try to bring some intelligence to the OF switch. In this work, we propose an entropy-based distributed DDoS flooding attack detection mechanism in OpenFlow-based SDN. We aim to lighten the overhead caused by flow collection. Meanwhile, we let the switch be smart enough to detect the DDoS attack proactively, not need to wait for the controller polls the flow tables and be detected the attack reactively. Then

*Corresponding author

978-1-4673-7952-6/15 $31.00 © 2015 IEEE
DOI 10.1109/Trustcom.2015.389

the attack can be found and reported more quickly. The main contributions of this paper can be summarized as follows:

1) Unlike the existing works, our DDoS detection algorithm is running in the OpenFlow edge switch. By doing statistics and analysis on the network traffic coming to the OpenFlow network, it achieves detecting the attack locally. So the heavy communication between the controller and the switches is reduced.

2) Taking advantage of the OpenFlow table in switches, we extend it by adding a copy of the packet number counter of each flow entry. So it's easy to get the flow information during a monitoring period.

3) In view of the computing ability of the OF switch, we use an effective entropy-based method to determine the randomness of the flow data and point out potential DDoS attacks. The experimental results show that our algorithm can detect the attack at the early monitoring intervals as the attack begins and achieve a high detection accuracy with a low false positive rate.

The remainder of this paper is organized as follows. Section II describes the related work. Section III explains our entropy-based distributed DDoS detection mechanism in detail. Section IV presents our experimental setups and results. Finally, Section V offers the concluding remarks.

II. Related Work

There has many researches on SDN and OpenFlow so far, e.g., [5] and [6]. Currently, they start to be a hot topic, not only limited in the research areas but also in the industry. In [7], the authors presented the experience of deploying Google's WAN, B4, using SDN principles and OpenFlow to manage individual switches. Various SDN controller softwares have been proposed by open source organizations or commercial corporations, such as NOX [8] and Floodlight [9]. Many traditional equipment manufacturers, like HP and Huawei, have gradually rolled out the SDN hardware switches that support OpenFlow. This means SDN has began the commercial road. And we can also use SDN software or programmable switches like Open vSwitch [10], NetFPGA [11] instead.

In the last few years, the security issues of SDN has gotten more and more attention. These studies can be classified as two aspects: utilize the SDN's centralized control to provide novel security service and secure network itself [19]. As an example of the first one, in [18] the authors utilized the SDN technology to defend against DDoS attacks in cloud computing. In this paper, we care about the second one. In [13], seven threat vectors are listed with three of them are specific to SDN. The centralized control of SDN makes the controller become more vulnerably to be attacked. To mitigate the bottleneck of the control path of SDN under the surge of control traffic, a solution that can elastically improve the control plane capability is presented in [19] by using an Openflow vSwitch overlay network.

As mentioned in the introduction, the attack's targets aren't restricted to the controller. To secure the OpenFlow network, many anomaly detection mechanisms have been designed in the SDN environment. In [20], to detect network security problems in home and office networks, the authors implemented four prominent traffic anomaly detection algorithms in an SDN context using OF compliant switches and NOX as a controller. However, the authors only used the low rate network traffic to do the experiments as they focused on the home environment and didn't discussed the overhead to the control plane when use these approaches in high rate traffic situation. In [17], to detect DDoS flooding attack at the controller, the authors defined a flow collector module which is responsible for periodically requesting flow entries of all flow tables from OF switches and did flow analysis by Self Organizing Maps. As an early flow-based intrusion detection mechanism in SDN, that work made use of the flow statistics character of OF switch. But that work also didn't concern about the controller's overhead caused by flow entries collecting process. In [16], with the use of the sFlow protocol, the authors reduced flow data gathering by sampling and reduced the required communication between OF switches and controller, thus easing the control plane's overload in the large network traffic condition. Moreover, the authors designed a work-wide anomaly mitigation using OpenFlow. But the authors didn't study that the flow sampling may affect the accuracy of anomaly detection. In [21], the authors proposed an adaptive flow collection method for anomaly detection in SDN. Using the programmable nature of SDN, an adaptive algorithm controls the temporal and spatial aggregation of the counting function in the switch. The experiments showed that the adaptive based counting can detect anomalies more accurately with less overhead. In [23], unlike the above flow sampling technology, the authors proposed a software defined traffic measurement architecture OpenSketch which implements hash-based data structures in the OF switch to count traffic. OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting) that enables a simple and efficient way to collect measurement data. But that kind of solution still relies on sending all the counters to the controller for analysis. In [24], the authors figured out that flow-based methods periodically collect data from the switch, the controller can only achieve accurate analysis for measurement at a large time-scale. If the attack traffic changes at a period smaller than the time-scale, the control rules couldn't work. The hash-based solutions are the same.

At the same time, more and more researchers try to bring some intelligence to the switch, e.g., [27]. In [26], the authors extended the existing OpenFlow data plane with two new modules. The first one is a connection migration module which enables the data plane to shield the control plane from DDoS SYN attacks. The second is an actuating trigger module which is a method for quickly responding to the network threat. In [28], an embedded layer 2-4 DDoS mitigation solution has been made. Rate-limited flow rules are programmed in the switch for the automatically detected layer 2-4 flow, like the trigger module. The shortcoming of that kind of approach is the threshold need to be calculated based on the traffic character of the network and different hosts may have to set different values. The latest Ethernet switch chip CTC8096 designed by Centec has supported the Elephant Flow Detection (EFD) and flow tracing on-chip [25]. Further more, in [24], the authors pointed an idea that we can leverage the local CPU to run simple measurement programs to collect and analyze more data at switches. If we can perform local analysis at the switch locally, we can easily support small time-scale measurement and highly variable traffic at line speed.

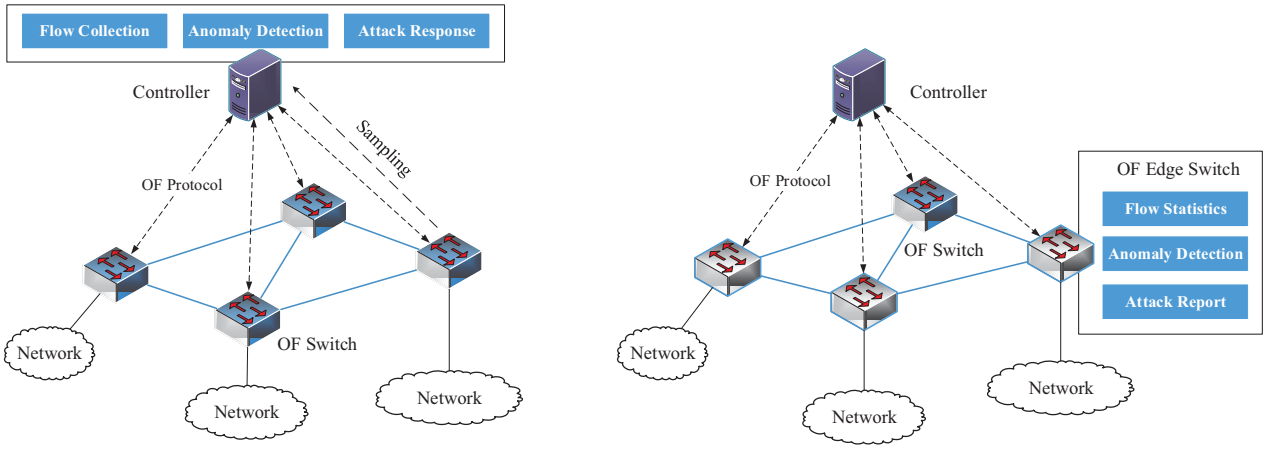

Fig. 1: SDN anomaly detection architecture

The DDoS attack detection mechanisms in the traditional networks have been widely studied, e.g., [29] and [30]. Those methods can be adopted in SDN as in [20] has done. As a member of anomaly detection methods, the entropy-based approach is real-time capable and can handle large amounts of traffic data [31]. More important, it has a low computing overhead. Many researchers have used the entropy-based approach to solve the security problems, e.g., [32] and [33]. Therefore, taking advantage of flow-based nature of SDN and considering the capacity of the OpenFlow switch, we propose an entropy-based DDoS flooding attack detection mechanism running in the edge switch locally. Therefore the attack on the OpenFlow network can be detected at a small time-scale and the communication overhead can be reduced.

III. DDoS Detection Model and Algorithms

In this section, we describe our DDoS detection mechanism in detail. We first list the motivation, assumptions and notations used in this paper, followed by the description of the flow statistics process. An entropy based DDoS anomaly detection model and the related algorithm will be presented in section III-C and III-D, respectively.

A. Motivation and Assumptions

The DDoS attacks are targeted to exhaust the victim's resources, like computing power and bandwidth. In this work, we focus on the DDoS flooding attack on the OF network of SDN. The attacks control the zombie hosts to send a large number of packets to the victim. Flooding attacks can be more easily detected near the victim, where all attack packets can be observed [15]. Therefore, as shown in Fig. 1, we put our attack detection mechanism at the OpenFlow edge switch where closes to the victim. Every edge switch is in charge of its local OpenFlow network's input traffic. So different from the flow sampling or collecting works, our DDoS detection is distributed which reduces the heavy communication overhead without the frequently flow collection.

As discussed in the related work, it's unrealistic and unwise to let the switch run complex anomaly detection algorithms, such as neural networks. Information entropy proposed by Shannon is a concept of information theory [34]. The entropy-based detection mechanism has a relative low calculation overload. The entropy is used to measure the randomness change of the incoming flows during a given time period. As a flow-based anomaly detection mechanism, we combine it with the flow-based feature of SDN to calculate the entropy value more conveniently.

TABLE I: Notations in this paper.

Notation	Definition
ΔT	The monitoring interval
$IPsrc$	The IP source address of a packet
$IPdst$	The IP destination address of a packet
$Srcport$	The source port of a packet
$Dstport$	The destination port of a packet
$IPproto$	The protocol type of a packet (TCP/UDP)
S_j	An OpenFlow switch
IF_i	An inputflow
N_{IF_i}	The variation of the number of packets of a given flow
$\|IF_i(t)\|$	The number of packets of the inputflow IF_i at time t

In order to make our analysis clear and simple, we present our assumptions as follows:

- One OpenFlow network connects to one OpenFlow edge switch at a time. One OpenFlow edge switch can connect more than one OpenFlow network. They can be treated as one network.
- Except the victim hosts, the OpenFlow network has other hosts. However, not every host in this network is the attack target. The victims are minority.
- During the attack, the number of DDoS flooding attack packets is much higher than the legitimate flows'.
- At any time, there will have traffic whether legitimate or malicious coming to more than one host of the OpenFlow network.

Notations used in this paper are listed in Table I, where the column "Notation" contains the names of variables, and the definition of each variable is presented in the column "Definition".

B. Flow Statistics Process

No matter what kind of anomaly detection mechanism, the flow collection is an important part to get the statistic data. In

Match Fields							Priority	Counters			Instructions	Timeouts	Cookies
Ingress port	IPsrc	IPdes	Srcport	Dstport	IPproto	···	···	···	Per Flow Entry	···	···		
									Received Packets	***RP_Local***			

Fig. 2: Flow Entry.

OpenFlow-based SDN, the OpenFlow switch has at least one flow table. The flow table can have many flow entries. Now, we give out the definition of flow in this paper.

Definition 1: **(Flow).** Flow refers to a set of sequential packets which have the same properties traveling through the same network during a period of time.

The "same properties" are not fixed and depend on the application scenarios. We define a flow by a five-tuple: $\langle IPsrc, IPdst, Srcport, Dstport, IPproto\rangle$. As shown in Fig. 2, the flow entry of OpenFlow switch records the information of a flow naturally. To every matched flow, the switch will add $Received\_Packets$ with one. Due to the decouple of the control and data plane, the traditional OpenFlow edge switch doesn't know its local topology and whether a IP address belongs to its local network or not. Secondly it also doesn't know the packet number of a specific flow handled by it during a interval. So without changing the decoupling characteristic, we extend the flow entry in the OF table by adding a copy counter (RP_Local) of $Received\_Packets$. The RP_Local is used to solve the problems mentioned above at the same time.

Firstly, for example, a simple SDN topology: $h1 - S1 - S2 - S3 - h2$. Host $h1$ wants to communicate with $h2$, like simple $h1$ $ping$ $h2$. ARP would be used to help $h1$ get the $default\ gateway\ mac$. At the first time, the OpenFlow switch $S1$ couldn't handle this ARP request packet and would send it to the controller by packet-in form. Then the controller can get the $h2$ mac response by simply flooding ARP request. Then the controller will know the relationship among IP, mac, $switch$ and $port$. When the controller issue the flow rules to the switches, the $Received\_Packets$ is initialized as 0. The RP_Local of the flow entry whose $IPdst$ is $h2$ will be set 0 initially when $h2$ belongs to its local network, like $S3$ in this example. Otherwise it will be set an unavailable value -1 to mark the $IPdst$ isn't in its network, like $S1, S2$. So, RP_Local is set as below:

$$RP\_Local = \begin{cases} -1, & IPdst \notin S_j, unavailable\ value, \\ 0, & otherwise, initial\ value. \end{cases}$$

Definition 2: **(Inputflow).** For an edge switch, an inputflow refers to the incoming flow whose IP destination address is in its local network.

To the second problem, we give the above definition of inputflow. As shown in Fig. 3, a local flow whose IP destination address is in the same network is still treated as an inputflow. We try to use inputflow to describe the incoming traffic to the edge switch. The inputflow can be identified by the $RP\_Local \neq -1$ of the flow entry. To record the number of packets of a inputflow at a time, at the end of every ΔT the value of the counter $Received\_Packets$ is copy to RP_Local. Therefore, an inputflow to an edge switch S_j can be defined as follows.

$$IF_i = \{\langle IPsrc, IPdst, Srcport, Dstport, IPproto\rangle | IPdst \in S_j\} \quad (1)$$

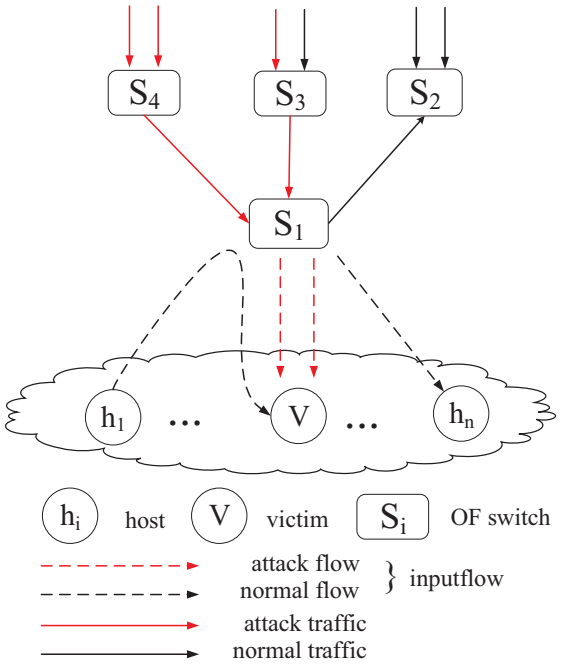

Fig. 3: A simple DDoS attack through an edge switch S_1.

This will not lose the detail information when we want to do fine-grained analysis. We can get the variation of the number of packets of an inputflow during the interval ΔT by formula (2). Then, the RP_Local of inputflow is updated as formula (3). Now, the edge switch can get the count number of the incoming traffic during the monitoring interval.

$$\begin{aligned} N_{IF_i}(t + \Delta T) &= |IF_i(t + \Delta T)| - |IF_i(t)| \\ &= Received\_Packets - RP\_Local \end{aligned} \quad (2)$$

$$RP\_Local = Received\_Packets \quad (3)$$

C. Entropy Based Anomaly Detection Model

TABLE II: Notations used in Anomaly Detection Model.

Notation	Definition
X_i	The number of packets sending to an active local host whose IP address=IP_i during ΔT interval
p_i	The probability of local IP address IP_i during ΔT
N	The total number of active local IP address during ΔT
$H(S_j)$	The entropy value of the edge switch S_j
$H_n(S_j)$	The entropy value of the edge switch S_j at normal status
$H_a(S_j)$	The entropy value of the edge switch S_j under DDoS attack
$E(S_j)$	The mean entropy value of the edge switch S_j at normal status
δ	Detection threshold
M	The minimum times that satisfied the formulae (8)
W	Window size
K	The number of nearest normal entropy values
σ	The standard deviation of normal entropy values
λ	Threshold multiplicative factor

In this section, we present an entropy based anomaly detection model for DDoS flooding attack in SDN. A list of

notations used in the model are defined in Table II. As the attacker can easily forge the IP source address, port number, so there can have many kinds of inputflow to the OpenFlow switch under attack. Some inputflows even appear only once. So we do the flow aggregation at first. This operation will let our model support the wildcard flow rules, not limit to the detailed flow rules.

Flow Aggregation. As the attackers target to one or more victims in a local network at one time. In this paper, we set $IPdst$ as an attribute to aggregate the inputflows that have the same $IPdst$. The controller can change the aggregate attribute on demand, like $\{IPdst, Dstport\}$. Therefore, the count number for an active local IP address during the ΔT can be calculated as follows:

$$X_i = \sum_{m=1}^{l} N_{IF_m}[IPdst = IP_i] \tag{4}$$

Then, we can get $X = \{X_1, X_2, X_3, ..., X_N\}$ as the count number of the incoming packets sending to every active local IP address through the edge switch.

Entropy Calculation. We use the frequency of each IP address to approximately estimate its probability.

$$p_i = \frac{X_i}{\sum_{i=1}^{N} X_i} \tag{5}$$

Then we can get the probability distribution of local IP address, $P = \{p_1, p_2, p_3, ..., p_N\}$. So, to an edge switch S_j, we calculate its information entropy value by:

$$H(S_j) = -\sum_{i=1}^{N} p_i \log p_i \tag{6}$$

As assumed, there will not happen no traffic situation which obviously has no attack, and also $N > 1$. So, based on the characteristics of the entropy function, we let the entropy value between (0,1] by normalized as below. We still use the symbol $H(S_j)$ later.

$$H'(S_j) = \frac{H(S_j)}{\log N} \tag{7}$$

Anomaly Determination. The timeline of the OF network can be divided into two segments: before DDoS flooding attack and under DDoS flooding attack. The entropy value of the edge switch S_j is corresponding denoted by $H_n(S_j)$ and $H_a(S_j)$. The network traffic is relative stable without the attack. Therefore, $H_n(S_j)$ is relative stable at the same. Let $E(S_j)$ be the mean entropy value at the normal status. When the victim hosts in the local network of the edge switch are attacked, the number of the packets that have the same $IPdst$ will increase quickly. This will cause the $H_a(S_j)$ to decrease sharply. The following formula will be met.

$$E(S_j) - H_a(S_j) > \delta \tag{8}$$

However, flash crowd will cause the same result. Considering the continuity of the DDoS attacks, to avoid this kind of misjudgment, the edge switch believes that some local hosts are under DDoS flooding attack if at least M times in the last W times ΔT interval the $H(S_j)$ satisfying the formula (8). Then, the switch will trigger a DDoS attack alert to the controller by sending an asynchronous information.

In order to make the anomaly detection be adaptive, we let the mean value $E(S_j)$ and the threshold δ adaptive to the change of the network traffic and won't use the fixed values. When the previous entropy value doesn't meet the formula (8), the previous entropy value is regard as normal. In the nearest K previous normal ΔT interval, we calculate the weighted mean value of $H_n(S_j)$ by:

$$E(S_j) = \sum_{i=1}^{K} \alpha_i \cdot H_n(S_j)[i], \ \sum_{i=1}^{K} \alpha_i = 1 \tag{9}$$

In order to emphasize the nearest entropy value, we make $\alpha_i < \alpha_j$,for $i < j$. These weighting coefficients are fixed but can also be changed by the controller. Then the standard deviation σ of these $H_n(S_j)$ can be calculated as follows:

$$\sigma = \sqrt{\frac{1}{K}\sum_{i=1}^{K}(H_n(S_j)[i] - E(S_j))^2} \tag{10}$$

Then we set the threshold δ as follows:

$$\delta = \lambda\sigma \tag{11}$$

λ is multiplicative factor. The $E(S_j)$ and σ can be calculated with the normal network traffic initially.

D. Algorithm for the Anomaly Detection Model

In this section, we design the algorithm for the anomaly detection model according to the above modeling and analysis. The anomaly detection algorithm is running in every OpenFlow edge switch. This algorithm monitors the inputflow based on the flow-based nature of SDN and does analysis on the collected data. Once a DDoS flooding attack is confirmed, this algorithm will send an alert information to the controller. The detail is shown as Algorithm 1.

The proposed algorithm can be easily implemented in the SDN software switch or programmable switch. It brings some intelligence to the switch. It can work as an independent module and the switch can do the forwarding as usual. So it does not change the nature of OpenFlow-based SDN.

The time complexity of the proposed algorithm is bounded by $O(n)$, n is the size of flow table. The additional storage contains the copies of the counters and the previous entropy values. Then, the space complexity is also $O(n)$.

E. Discussion on the Anomaly Mitigation

After the DDoS attack has been detected, another important thing to do is the anomaly mitigation. At first, the controller need to figure out whether this switch does a misjudgement or not. Because some hosts in the network may have more flash crowd traffic than others, e.g., FTP server. This can be solved by the White List table as in [16] discussed. Then the controller or the switch need to find who is the victim of the attack. The simplest way to response the attack is inserting a "drop action" rule to the switch about all the packets to this victim. This can work very efficiently if the attack traffic

Algorithm 1 The Anomaly Detection Algorithm.

initialize the local threshold parameters: $E(S_j), \delta$, detection parameters: M, W, K, λ and the interval ΔT;
for all $Flow \in S_j$ **do**
  if $RP\_Local \neq -1$ **then**
    $identify\ as\ IF_1, IF_2, ..., IF_s$;
  end if
end for
$when\ \Delta\ T\ is\ over$
for all $IF_i \in S_j$ **do**
  $N_{IF_i}(t + \Delta T) = Received\_Packets - RP\_Local$;
  $RP\_Local = Received\_Packets$;
  if $IPdst = IP_j$ **then**
    $X_j += N_{IF_i}$;
  end if
end for
for $i \leftarrow 1$ to N **do**
  $p_i = \frac{X_i}{\sum_{i=1}^{N} X_i}$;
  $H(S_j) += -p_i \log p_i$;
end for
do $H(S_j) = \frac{H(S_j)}{\log N}$;
if $E(S_j) - H(S_j) > \delta$ **then**
  if $M\ times\ in\ W$ **then**
    $DDoS\ flooding\ attack\ confirm\ and\ report$;
  end if
else
  $E(S_j) = \sum_{i=1}^{K} \alpha_i \cdot H_n(S_j)[i]$;
  $\sigma = \sqrt{\frac{1}{K} \sum_{i=1}^{K} (H_n(S_j)[i] - E(S_j))^2}$;
  $\delta = \lambda\sigma$;
end if
$go\ to\ line\ 2$

is very huge. However, the legitimate traffic is affected at the same time. In consideration of the centralized control feature of the network, we can easily trace back to the source OpenFlow networks which contain the attackers by marking every packets with the switch ID. And then do the source filtering at the source switch.

IV. Experiments

In this section, we first describe the experimental setups. And then we investigate the feasibility of our DDoS attack detection mechanism by several DDoS attack experiments. At last, we evaluate the performance and efficiency of our anomaly detection algorithm.

A. Experimental Setups

To evaluate our work, we use Mininet [35] as the network simulator. Mininet can create a realistic virtual network, running real kernel, switch and application code. It is also a great way to develop and experiment with OpenFlow and SDN systems. Our algorithm is implemented in Open vSwitch, a famous SDN software switch which can handle the required traffic loads. The algorithm is running as a periodic task in the switch. We replace the original Open vSwitch in Mininet with our modified one. And also, we use a Java-based open source controller, Floodlight [9]. The experiments are done at Ubuntu 14.04, i5 CPU and 8G RAM.

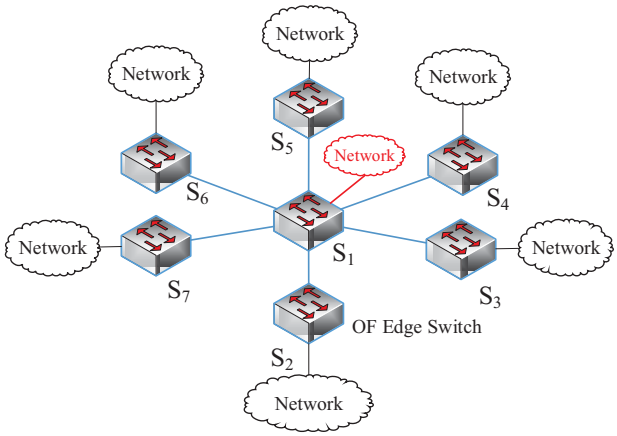

Fig. 4: Network topology used.

As shown in Fig. 4, we use a tree topology which is created by the Python Api of Mininet. The bandwidth of each link is set 800 Mbps. Each OpenFlow network contains at leat 20 hosts. The victim host is located in the network of S_1 and DDoS attackers launch the attack from other networks. D-ITG (Distributed Internet Traffic Generator) [36] is used to generate the legitimate traffic. The rates are changed from 50 to 500 Mbps which represent slow and high traffic network environments, as shown in Table III. 80% of the legitimate traffic is TCP traffic and 20% is UDP traffic. The Center for Applied Internet Data Analysis (CAIDA) has released "DDoS Attack 2007" Dataset which contains the attack traffic to the victim [37]. We leverage this dataset to do our experiments and use the Python tool Scapy [38] to generate DDoS flooding attack traffic from zombie hosts to the victim.

TABLE III: Parameter values of the traffic.

	Average traffic rate (Mbps)	Attack rate (pkts/s)
Exp.1	50	50-200
Exp.2	100	300-500
Exp.3	500	1000-2000

B. Anomaly Detection

To examine our proposed mechanism can work correctly in OF switch, we first simulate one DDoS attack on the host h_1 in the network of S_1 using the parameter of $Exp.1$. The simulation time is 200 s. And during the time 100-150 s, we launch a DDoS flooding attack at rate of 150 pkts/s from the network of $S_2 - S_7$. Through the log file, the normalized entropy value tendency on the IP destination address of the switch S_1 is shown in Fig. 5. The monitoring interval ΔT in this scenario is 7 s. The red line is the normal entropy value without attack. We can see the entropy value changed within a range. From the blue line, we can figure out that the the entropy value declined rapidly at 100s and kept a low value until the 150 s. If we choose the $M = 2, W = 3, \lambda = 3$, then the DDoS attack can be detected at time 112 s.

To evaluate the influence of different monitoring interval ΔT, we change it from 3 to 7 s. As shown in Fig. 6, we can see that they all have the same tendency. When the ΔT is set 3 s, the attack will be detected at time 105 s, and 110 s when set 5 s. Thus, the smaller interval can have more quickly detection than the longer one. However, we can observe the curve of the small

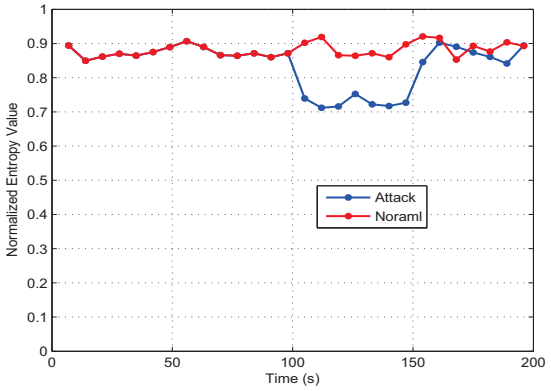

Fig. 5: The normalized entropy value of IPdst Flow.

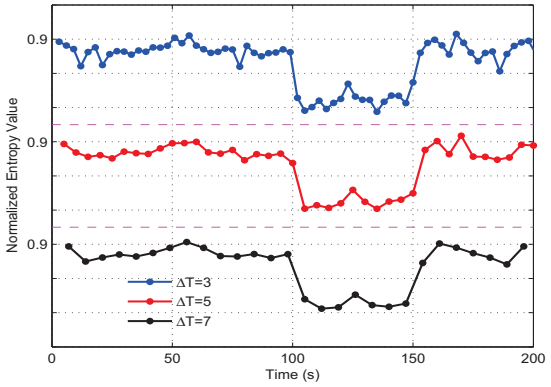

Fig. 6: Entropy values for different ΔT.

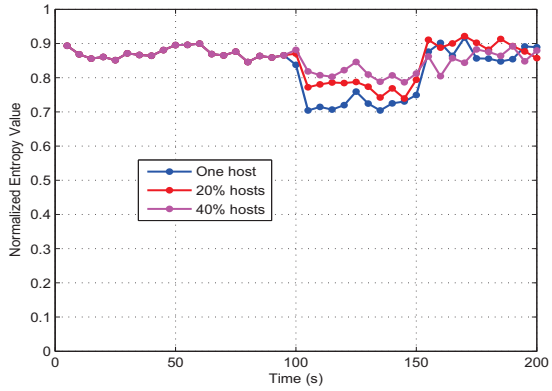

Fig. 7: Entropy values for different rates of victim hosts.

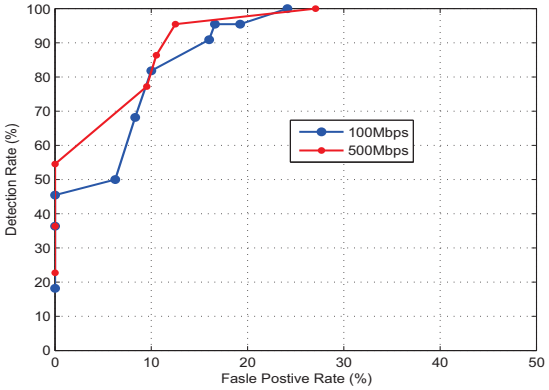

Fig. 8: ROC curves for the 100 and 500 Mbps cases.

interval is less smooth than long one's as the network traffic during the small interval is small correspondingly. Besides, the switch has to do more calculation in the same duration.

To evaluate the effect of different numbers of attack targets to our algorithm, we change the rate of victim hosts. ΔT is set 5 s and we still use the parameter of $Exp.1$. As shown in Fig. 7, we can see that when the 40% hosts are under attack at the same time, the entropy value changed less than the other two and it is more difficult to distinguish the attack traffic from the normal unless uses smaller λ. But this brings more misjudgements. So the less attack victims at the same time, the better detected out algorithm will do.

C. Performance Evaluation

The performance of our detection mechanism is evaluated with $Detection\ Rate$, $False\ Positive\ Rate$. The $Detection\ Rate(R_d)$ of the DDoS is defined as follows:

$$R_d = \frac{TP}{TP + FN} \tag{12}$$

True Positives (TP) are attack traffic detected as attack traffic and False Negatives (FN) are attack traffic detected as legitimate traffic. In the experiments, R_d reflects how many attacks of all the generated attacks can be detected.

$False\ Positive\ Rate$ is defined as follows:

$$R_{fp} = \frac{FP}{TN + FP} \tag{13}$$

False Positives (FP) are legitimate traffic detected as attack traffic. True Negatives (TN) are legitimate traffic detected as legitimate traffic. R_{fp} reflects how many mistaken attacks are detected out of the normal traffic when the network is without DDoS attacks.

We use the $Receiver\ Operating\ Characteristic\ (ROC)$ curves to show the trade-off between the detection rate and false positive rate by varying the factor λ. These metrics are measured under DDoS UDP flooding attacks to one victim host. The traffic parameters are used of Exp.2 and Exp.3 in the Table III. The monitoring interval ΔT is 5 s, M is set 2 and W is 3. As shown in Fig.8, we can see that the algorithm can achieve 100% detection rate while has approximately 25% false positive ratio for both 100 and 500 Mbps background traffic. The algorithm works better in high traffic scenario, because this scenario has a higher rate of background network traffic which is beneficial to randomize the traffic.

D. Efficiency Analysis

As said in [32], the network security community lacks suitable real large-scale DDoS attacks datasets. As an emerging network, it is also hard for SDN to find suitable and acknowledged datasets to do the compare with different algorithms. Moreover, existing good works about the anomaly detection in SDN, such as [16] and [20], even didn't list the detail of the algorithms they used. However, all these centralized works need to collect flow information from the switch regularly and

the attack detection time depends on the polling interval. A small time-scale will detect the attack quickly, but it collects much more flow data than the long one. In this aspect, our local algorithm can reduce this communication overhead and do the detection at a small time-scale. The centralized detection at the controller can detect the attacks happened in the whole network. In our algorithm, one edge switch can only detect the DDoS attacks on its local network. But when distributed running in every edge switch, the algorithm can also detect the attacks in the whole network.

V. Concluding and Future Works

In this paper, we extended a counter copy of the flow entry in the OpenFlow table. Using this value, we can identify inputflow easily and then achieved the flow statistics process in the switch. To monitor the incoming traffic, we proposed an entropy-based DDoS flooding attack detection model and the corresponding algorithm. By running this algorithm in the edge switch of SDN, we achieved the distributed anomaly detection mechanism. The calculation of entropy value has a low overload and this algorithm can be easily implemented in the SDN software or programmable switch. Our mechanism can reduce the frequently data gathering process and relieve the communication overload between the OF switches and the controller. Due to the flow aggregation process, our detection mechanism can also work well when uses the wildcard flow rules during the DDoS attack. The experimental results showed that our algorithm can detect the attack at the early monitoring intervals when the attack begins and achieve a high detection accuracy with a low false positive rate.

In the future, we are planning to design the victim identification mechanism in the edge switch, which can realize the quickly anomaly response. And also, we want to leverage the centralized control characteristic to realize the trace back of DDoS attack and the source filtering.

Acknowledgment

This research is sponsored by the Natural Science Foundation of China (NSFC) under Grant No. 61202015, Research Fund for the Doctoral Program of Higher Education of China (RFDP) grant 20120131120033, Shandong Provincial Natural Science Foundation under Grant No. ZR2013FM028, the Fundamental Research Funds of Shandong University under No. 2015JC030, and Guangdong Provincial open fund of Key Laboratory 2012A0614024.

References

[1] Software-Defined Networking. https://www.opennetworking.org/images/stories /downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[2] OpenFlow. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf.

[3] sFlow. http://www.sflow.org/

[4] Cisco NetFlow. http://www.cisco.com/go/netflow

[5] Dixit A, Hao F, Mukherjee S, et al. Towards an elastic distributed SDN controller. In *ACM SIGCOMM Computer Communication Review*, ACM, 2013, 43(4): 7-12.

[6] Hand R, Keller E. ClosedFlow: openflow-like control over proprietary devices. In *Proceedings of the third workshop on Hot topics in software defined networking*, ACM, 2014: 7-12.

[7] Jain S, Kumar A, Mandal S, et al. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM Computer Communication Review*, ACM, 2013, 43(4): 3-14.

[8] NOX. http://www.noxrepo.org/

[9] Floodlight. http://www.projectfloodlight.org/

[10] Open vSwitch. http://openvswitch.org/

[11] Lockwood J W, McKeown N, Watson G, et al. NetFPGA–an open platform for gigabit-rate network switching and routing. In *IEEE International Conference on Microelectronic Systems Education*, IEEE, 2007: 160-161.

[12] Scott-Hayward S, O'Callaghan G, Sezer S. Sdn security: A survey. In *Future Networks and Services*, 2013 IEEE SDN for (SDN4FNS), IEEE, 2013: 1-7.

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

[14] Sezer S, Scott-Hayward S, Chouhan P K, et al. Are we ready for SDN? Implementation challenges for software-defined networks. In *Communications Magazine*, IEEE, 2013, 51(7): 36-43.

[15] Mirkovic J, Reiher P. A taxonomy of DDoS attack and DDoS defense mechanisms. In *ACM SIGCOMM Computer Communication Review*, 2004, 34(2): 39-53.

[16] Giotis K, Argyropoulos C, Androulidakis G, et al. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. In *Computer Networks*, 2014, 62: 122-136.

[17] Braga R, Mota E, Passito A. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In *2010 IEEE 35th Conference on Local Computer Networks (LCN)*, IEEE, 2010: 408-415.

[18] Wang B, Zheng Y, Lou W, et al. DDoS Attack Protection in the Era of Cloud Computing and Software-Defined Networking. In *2014 IEEE 22nd International Conference on Network Protocols (ICNP)*,IEEE, 2014: 624-629.

[19] Wang A, Guo Y, Hao F, et al. Scotch: Elastically Scaling up SDN Control-Plane using vSwitch based Overlay. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, ACM, 2014: 403-414.

[20] Mehdi S A, Khalid J, Khayam S A. Revisiting traffic anomaly detection using software defined networking. In *Recent Advances in Intrusion Detection*, Springer Berlin Heidelberg, 2011: 161-180.

[21] Zhang Y. An adaptive flow counting method for anomaly detection in sdn. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, ACM, 2013: 25-30.

[22] Nunes B, Mendonca M, Nguyen X, et al. A survey of software-defined networking: Past, present, and future of programmable networks[J]. 2014.

[23] Yu M, Jose L, Miao R. Software Defined Traffic Measurement with OpenSketch. In *NSDI*. 2013, 13: 29-42.

[24] Moshref M, Yu M, Govindan R. Resource/accuracy tradeoffs in software-defined measurement. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, ACM, 2013: 73-78.

[25] Centec CTC8096. http://www.centecnetworks.com/en/ProductList.asp?ID=287

[26] Shin S, Yegneswaran V, Porras P, et al. 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: 413-424.

[27] Vizvry M, Vykopal J. Future of DDoS Attacks Mitigation in Software Defined Networks. In *Monitoring and Securing Virtualized Networks and Services*, Springer Berlin Heidelberg, 2014: 123-127.

[28] Krishnan R, Krishnaswamy D, Mcdysan D. Behavioral Security Threat Detection Strategies for Data Center Switches and Routers. In *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, 2014: 82-87.

[29] Feinstein L, Schnackenberg D, Balupari R, et al. Statistical approaches to DDoS attack detection and response. In *DARPA Information Survivability Conference and Exposition, 2003, Proceedings*, IEEE, 2003, 1: 303-314.

[30] Mirkovic J, Reiher P. A taxonomy of DDoS attack and DDoS defense mechanisms. In *ACM SIGCOMM Computer Communication Review*, 2004, 34(2): 39-53.

[31] Wagner A, Plattner B. Entropy based worm and anomaly detection in fast IP networks. In *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, IEEE, 2005: 172-177.

[32] Yu S, Zhou W, Doss R, et al. Traceback of DDoS attacks using entropy variations. In *IEEE Transactions on Parallel and Distributed Systems*, 2011, 22(3): 412-425.

[33] Kumar K, Joshi R C, Singh K. A distributed approach using entropy to detect DDoS attacks in ISP domain. In *International Conference on Signal Processing, Communications and Networking*, IEEE, 2007: 331-337.

[34] Shannon C E. Prediction and entropy of printed English. In *Bell system technical journal*, 1951, 30(1): 50-64.

[35] Mininet. http://mininet.org/

[36] D-ITG. http://traffic.comics.unina.it/software/ITG/

[37] The CAIDA UCSD "DDoS Attack 2007" Dataset. http://www.caida.org/data/passive/ddos20070804_dataset.xml

[38] Scapy. http://www.secdev.org/projects/scapy/