Hindawi
Security and Communication Networks
Volume 2021, Article ID 5097267, 22 pages
https://doi.org/10.1155/2021/5097267

Research Article

SDNDefender: A Comprehensive DDoS Defense Mechanism Using Hybrid Approaches over Software Defined Networking

Tianfang Yu, Lanlan Rui, and Xuesong Qiu

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Correspondence should be addressed to Xuesong Qiu; xsqiu@bupt.edu.cn

Received 13 May 2021; Revised 13 September 2021; Accepted 27 September 2021; Published 18 October 2021

Academic Editor: Weizhi Meng

In traditional networks, DDoS attacks are often launched in the network layer or the transport layer. Researchers had explored this problem in depth and put forward plenty of solutions. However, these solutions are only suitable for scenarios such as a single link or victim side network and could not analyse traffic distribution from the angle of the global network. Also, the TCP/IP network architecture lacks abilities to quickly conduct resource deployment and traffic scheduling. When DDoS attacks occur, victims usually could not respond in time. With the superiorities of centralized control mode and global topological view, Software-Defined Networking (SDN) provides a new way to get over the above issues. In this paper, we adopt a combination of diverse technologies to design SDNDefender, a SDN-based DDoS detection and defense mechanism, which is composed of two core components aiming to counter the most popular DDoS attacks including IP spoofing attack and TCP SYN flood attack. We carry out quantitative simulation experiments for evaluating SDNDefender from many metrics. The experimental results show that in contrast to other DDoS defense algorithms, SDNDefender not only efficiently validates spoofed packets and withstands well-known attacks but also defends unknown attacks according to the target's available resources. Besides, SDNDefender could significantly reduce TCP half-open connections and improve detection accuracy, alleviating attack influences that exhaust the server's resources and network bandwidth.

1. Introduction

With the rapid growth of networking technologies, current network systems and data centers are becoming more and more complex and data excessive. For system designers, ensuring the high availability and security of resources and services is still a crucial and challenging issue. Among different kinds of cyber-threats, DDoS attacks are most harmful, attracting tremendous research efforts from both academia and industry. DDoS is a large-scale and distributed network attack, which launches illegal requests to attack targeted hosts and exhaust their bandwidth and service resources. The fundamental cause of a DDoS attack is that in TCP/IP protocol stack, connection requests of any proxy program could be sent to the targeted server without identity authentication so that attackers could configure proxy servers to attack victims. At present, scholars carry out further research on DDoS attack defense technology. Ref. [1] presents a detection method based on the Pearson correlation coefficient to discriminate anomaly traffic from normal traffic. Ref. [2] builds a traffic self-similar model with chaos theory, which utilizes the Lyapunov equation to detect attack behavior and trains samples to filter anomaly traffic using the BPNN network. Ref. [3] proposes a sliding window-based flow interaction feature algorithm that could lower interruptions of background flows and raise attack detection rates. Ref. [4] designs a Naive Bayes classification method that makes use of datasets from discrete Wavelet transform training and discrete Fourier transform training to identify attacking traffic. Experiment results show that this method has better calculation accuracy than that of those methods using threshold values. Ref. [5] uses a dynamic game defense measure to deal with DDoS attacks in the botnet environment. Ref. [6] builds a cooperation

defense mechanism with Network Function Virtualization (NFV). The mechanism sets up a virtual intrusion detection system and a resource allocation model to filter and redirect anomaly traffic.

In summary, current DDoS defense mechanisms mainly choose attack features and congestion patterns as detection evidence and further design models and algorithms by statistics, machine learning, data mining, and so forth. However, under the Internet architecture, building network topology and forwarding packets seriously rely on network protocols and there are no uniform norms between ports of network devices, so these flaws constrain executions of these defense strategies.

SDN paves a new way to tackle the abovementioned problems. First, the controller as the core component separates control function from network devices and takes charge of global topology management, creating flow entries and updating link status, whereas network devices merely realize data forwarding. Second, SDN adopts flow forwarding rather than packet forwarding. When detecting attack behavior, the controller could make decisions and install flow entries on switches to block attacks. According to the analysis above, we present SDNDefender, a SDN-based DDoS detection and defense mechanism. Compared with other DDoS defense strategies for SDN environments, SDNDefender is a comprehensive detection and defense mechanism that can counter different kinds of DDoS attacks such as ARP/IP spoofing and TCP SYN flood. It not only uses dynamic flow entries and an intrusion prevention library to track source address and match well-known cyber-attacks but also defends unknown attacks by monitoring the available host resource. SDNDefender consists of two core components: Anti-IP spoofing and Anti-TCP SYN flood. When receiving incoming packets, the Anti-IP spoofing component validates whether they have forged source IP addresses by calculating their forwarding paths, and if confirmed, it will immediately install flow rules on affected OF switches to directly drop these packets. In addition, this component is also responsible for examining packet's legitimacy through building an intrusion prevention rules (IPR) library and designing a host-based anomaly detection algorithm for suspicious access. For the Anti-TCP SYN flood component, it firstly verifies SYN packets' transmission rate with univariate Gaussian distribution. Once suspicious TCP flows are recognized, it calculates entropy based on destination IP address and the network self-similarity index to go on further detecting the onset of the SYN flood attack. As flood traffic is detected, the component starts to install flow rules and deploy priority queues on OF switch ports to block attack packets and allocate the least bandwidth resource for a small quantity of anomaly traffic into the network to ensure that normal flows can be forwarded timely. We implement simulation experiments to evaluate SDNDefender and compare it with other defense algorithms using many metrics. Our results show that under IP spoofing attacks, SDNDefender can reduce the maximum average packets per flow and the server's CPU load by 30% and 37%, respectively. Also, under TCP SYN flood attacks, SDNDefender not only slows down SYN packets' transmission rate and decreases the maximum TCP half-open connections by 35% but also observably improves detection accuracy and lowers the false positive rate.

The main contributions of this paper are summarized as follows:

(1) We present a dynamic entries match (DEM) algorithm to validate incoming packets' source addresses. Leveraging SDN's centralized control, DEM algorithm uses path-based routing rules to match a packet and decides the next action for it.

(2) Inspired by misuse detection, we build an IPR library to check intrusion behavior. IPR extracts signatures of well-known cyber-attacks and stores them in a rule tree. Besides, we design a host-based anomaly detection algorithm to counter unknown attack events.

(3) On detecting TCP SYN flood attack, we employ univariate Gaussian distribution to examine SYN packets' transmission rate. For dubious flows, we propose a detection algorithm to determine whether there exists flood traffic using both entropy based on destination IP address and network self-similarity index.

(4) We present a SYN flood mitigation method based on DiffServ model. It takes advantage of ingress port blocking and priority queue scheduling to relieve flood traffic's effects on the network.

The rest of this paper is organized as follows. We first introduce background knowledge of DDoS and SDN, and describe current research progress in Section 2. Next, we expound the design principles of SDNDefender in Section 3. Then, we implement simulation experiments and discuss experimental results in Section 4. Meanwhile, we analyse those factors affecting our experiments in Section 5. Finally, we conclude with a summary of the main points in Section 6.

2. Background and Related Work

2.1. DDoS Attack. Compared to the single attack method of DoS attacks, DDoS is a large-scale and distributed network attack that is more detrimental to Internet services. It utilizes multitudinous controlled nodes to launch DoS attacks on targets and brings victims a breakdown by exhausting their network bandwidth and system resources. Figure 1 illustrates an entire DDoS attack process that is made up of an attacker, masters, zombies, and victims. First of all, the attacker needs to gather system configuration and run state of targets. Second, the attacker makes use of loophole scanners or backdoor programs to intrude as many hosts as possible, and then these affected hosts are divided into masters and zombies. Each master controls a range of zombies and on receiving attack codes from the attacker, the master forwards these commands to its zombies and informs them of attacking targets.

2.2. SDN. Legacy IP networks are complex and difficult to manage because their control and data planes are vertically

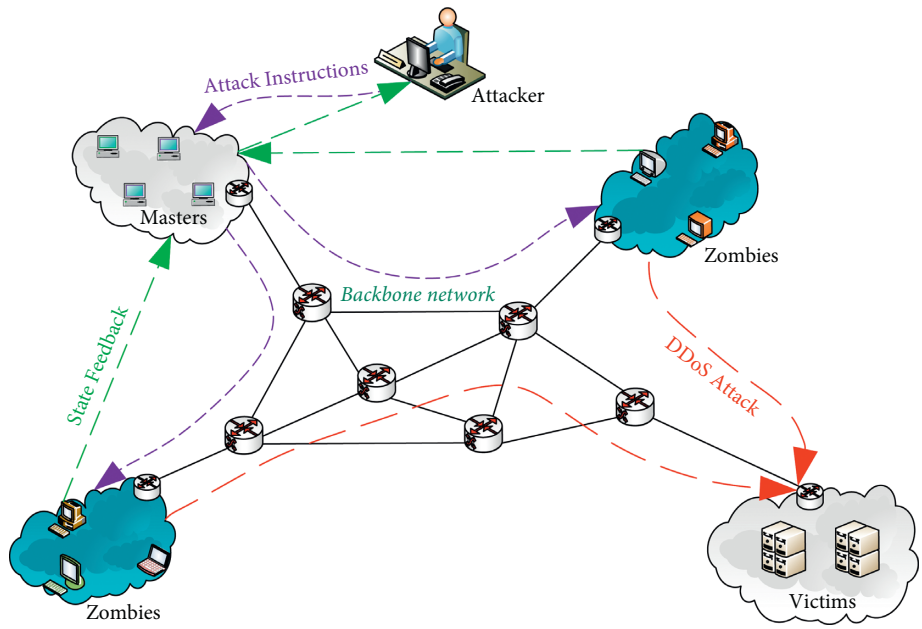

Figure 1: An entire DDoS attack process.

integrated. What is more, they involve network devices that run distributed control software that is closed and proprietary. SDN is an emerging software-based network architecture that supports centralized network control and programmability and offers rich APIs to lift automatic network management level and meet various business demands. As seen from Figure 2, a basic SDN architecture comprises infrastructure, control, and application layers. The infrastructure layer includes a group of one or more network devices such as OpenFlow (OF) switches, routers, and terminals, each of which contains a set of traffic scheduling and gathering statistics resources that directly tackle end user traffic, along with the necessary supporting resources to ensure proper connectivity, security, and quality. The responsibilities of the control layer are mainly programming and managing the infrastructure layer. The control layer is composed of a set of SDN controllers that translate the upper applications' requirements and exert low-level control over the network devices through standardized interfaces while providing relevant information up to the SDN applications. The application layer supplies network applications that can bring new network features or services, such as traffic engineering, security and network verification. In addition, the application layer can receive a global view and overall statistics of the network from the controllers and use this information to provide appropriate guidance to the control layer.

As the most commonly used protocol for the southbound interface of SDN, OpenFlow decides basic components, core functions, and protocols. As shown in Figure 3, an OF switch is a forwarding device that deals with incoming packets in terms of its flow tables that consist of a set of flow entries, each of which is made up of match fields, counters, and instructions. The match fields in a flow entry can match various protocols, such as Ethernet, IPv4, ICMP, UDP, or TCP, depending on the OpenFlow specification. The counters are used to track statistics relevant to a flow, such as how many packets and bytes have been forwarded for this flow, as well as the duration of this flow. The instructions prescribe what the switch should do with a packet matching this entry. Common instructions are "forward," "drop," "modify field," etc.

SDN provides users with diverse network functions, yet also faces many cyber-security challenges such as DDoS attacks. In the infrastructure layer of SDN, OF switches and their flow tables are likely to be a major target, because they maintain information related to network management, traffic scheduling, and fine-grained access control. More specifically, the main task of an OpenFlow controller is gathering the run state of underlying devices, making forwarding policies and instructing the actions of network elements. It manages the switch flow table by adding or removing flow entries via the secure channel. To make the forwarding fast and efficient, the flow table usually uses ternary content addressable memory (TCAM). However, TCAM is usually very costly and only supports limited flow entries. When attackers utilize network protocol flaws, such as address spoofing or packet flood technique, to compromise targets and even make them unavailable, there will be a large number of useless flow rules and the flow table might be unable to store normal flow rules. Therefore, the first and foremost security challenge is distinguishing malicious rules from genuine flow rules. The second security challenge is about the number of flow entries a switch can accommodate. In OpenFlow, every switch needs to buffer unsolicited flows until the controller releases new flow rules. Due to the limited memory of the switches, the infrastructure layer is prone to suffer saturation attacks. In the past few years, many researchers have discussed DDoS attacks in the SDN environment and put forward a lot of solutions. Authors in

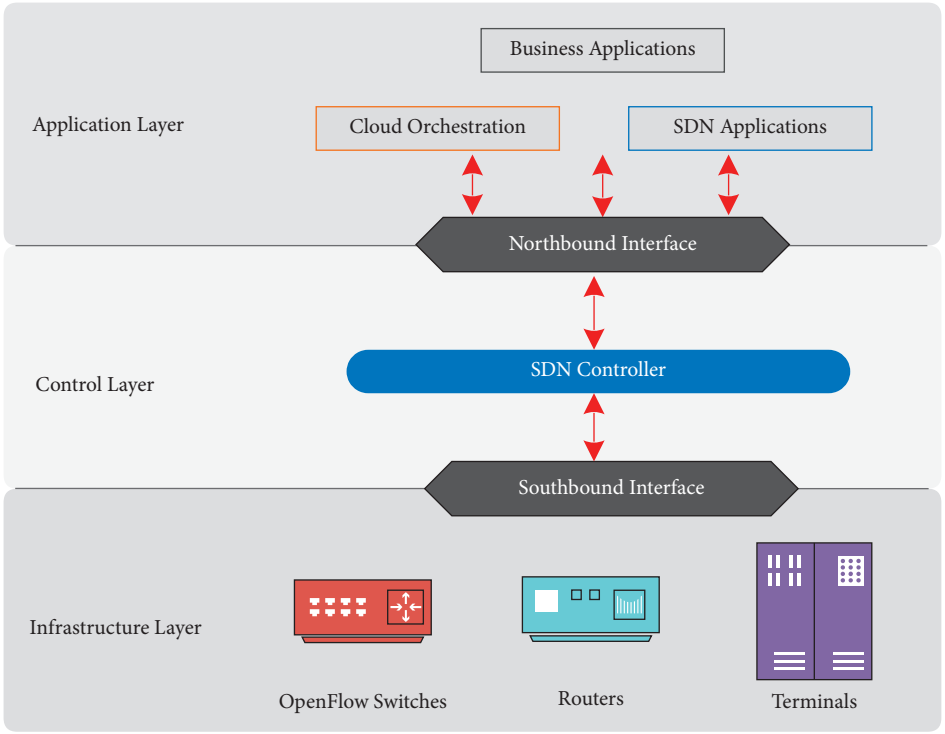

Figure 2: A 3-layer SDN architecture.

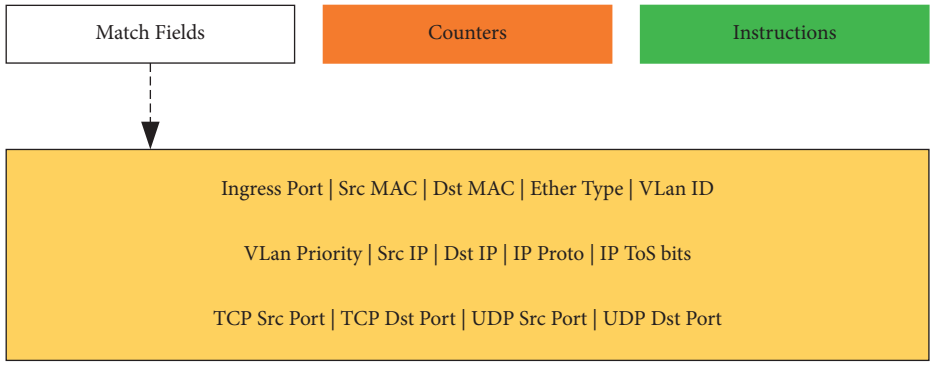

Figure 3: Main components of a flow entry.

Ref. [7] present a SDN-based security policy enforcement framework for ships' communication networks. This framework uses Event-Condition-Action (ECA) paradigm for high-level security policies to define flow types and corresponding actions. With network service header (NSH), network traffic can be monitored from end to end. To defend slow-running attacks in SDN, authors in Ref. [8] present a three-phase mitigation framework, which detects potential attackers according to the servers' state and identifies them with multiple metrics. The framework leverages the SDN controller to send RST packets to release the resources occupied by attackers. Ref. [9] employs both sFlow and adaptive polling sampling to propagate all related traffic statistics to the IDS for classifying different feature extractions. Meanwhile, a deep neural network algorithm-Stacked AutoEncoder is used to determine whether there exists DDoS attacks in the network. In this paper, we specifically focus on those solutions towards defending against IP spoofing attack and TCP SYN flood attack.

2.3. Proposals for IP Spoofing Attack. IP spoofing refers to an IP source address forgery or host files hijacking technique in which attackers can conceal their real identities and launch attacks to gain unauthorized access and sensitive data of the targeted host. This threat derives from the shortcoming that Internet packet forwarding in routers merely depends on each packet's destination IP address but neglects the validation of the packet's source IP address to verify sender authenticity. At present, most of the anti-spoofing solutions can be categorized into path-based filtering, end-to-end, and traceback. Path-based filtering proposals verify and filter

packets by the paths they flow through, and, to a large extent, ensure that targeted hosts avoid receiving attack packets. Ingress Filtering [10] is a typical and lightweight filtering solution, which is deployed into a router or a firewall between two networks and can restrain attack packets always staying in the subnet to which attackers belong. However, this solution only defends attacks from the intra-domain and is not beneficial for ISPs. End-to-end technique adds a signature into a packet at the sender side and verifies the packet's genuineness using the signature at the receiver side. IPSec [11] is a host-level identification mechanism that uses a private key to encrypt a packet and decrypt it by a sender's public key. Although IPSec can examine a packet's integrity, its high overhead cost cannot efficiently defend against DDoS attacks. APPA [12] employs a single Hash function to create a signature for both sender and receiver, and each signature can only be used once. This not only decreases interactions of both sides but also reduces communication complexity. Compared with the two types of solutions above, traceback is used for tracing attackers' real addresses through packet mark. Packet mark technique adds router marks into a packet and the receiver gets the packet's practical path using these marks. But this technique is not suitable for large-scale networks because the more routers a packet traverses, the longer the packet's length is, and when a new mark is added, the earliest marks will be erased. In summary, these solutions still face some problems to solve, although they achieve good effects for anti-spoofing.

As discussed before, SDN owns the global topological view and the centralized control capacity. This can interpret access control list (ACL) by flow rules in OF switches, and bring a new opportunity to solve the IP spoofing issue and overcome shortcomings in traditional solutions. So far, there are some SDN-based solutions to the IP spoofing problem. SAVSH [13] aims to achieve the best trade-off between packet filtering accuracy and deployment cost. It converts the hybrid network into a sink-tree and locates deployment nodes by computing the minimum OF switches deployment ratio with all legal IP prefix pairs. It also utilizes the SDN controller to generate filtering rules and deal with network dynamics. However, SAVSH needs to adapt to topological change in advance and take a high delay for redistributing new filtering rules. SIPAV-SDN [14] maintains a HostTable, where the switch id, switch port, host IP, and MAC are recorded as an item by extracting details from Packet_In messages. The scheme validates each incoming packet with the table and only installs flow entries for legitimate packets to forward them. Ref. [15] presents DDAH, a host-based dynamic detection method for IP spoofing in SDN. DDAH uses access ports to find anomaly hosts and periodically calculates their weight and activity. If a port's activity exceeds the threshold, DDAH will select some flow features and employ machine-learning algorithms for further detection and mitigation. BASE [16] takes BGP information to calculate path-based marking values and distributes them into BGP routers. Each BGP router updates its flow rules according to the marking value and filters every incoming packet with an incorrect mark. For BGP protocol, any route change brings a recalculation of the routing table, so multiple recalculations will overload the router. Also, the filtering accuracy of BASE relies on the network scale.

2.4. Proposals for TCP SYN Flood Attack. TCP SYN flood is one of the most widespread DDoS attacks, in which attackers exploit the 3-way handshake mechanism in TCP, and send a mass of spoofed SYN packets to consume resource on the targeted server and degrade its availability. Most of the existing proposals for defending against SYN flood attacks in legacy networks use end host-based methods. SYN cookie, SYN cache [17], and SYN proxy [18] all need a modification in the host's TCP stack so as to change algorithms and data structures used for connection lookup and establishment, yet the main limitation of these solutions is that they can only detect attacks at the receiver side. In this way, the network will be seriously congested due to fierce attack traffic. Other solutions like firewalls and intrusion detection system (IDS) are usually deployed on the edge of the targeted network and employ static rules or signature-based strategies to tackle anomaly traffic. Although they can detect and block trivial DDoS attacks, they will be compromised as attackers launch attacks using spoofed IP addresses or different TCP source ports.

In summary, none of the solutions above can be directly applied to SDN, because SYN flood attacks can destroy the entire SDN network rather than a single targeted host. Specifically, the controller has global network information, so it is apt to suffer the saturation attack due to receiving a huge number of PACKET_IN messages at a high speed and becomes a single point of failure for the network. In the last few years, many solutions have been proposed in the literature to counter SYN flood attacks in SDN networks. AVANT-GUARD [19] uses each OF switch as a proxy to inform the controller about the connection requests that successfully perform TCP 3-way handshake via an extension message. To mitigate SYN flood attacks in SDN, AVANT-GUARD needs to modify the controller and OF switches together. Moreover, it requires additional resources to establish new connections for legitimate TCP requests. This will cause long delays and introduce a buffer saturation attack. LineSwitch [20] also uses the SYN proxy technique to maintain a minimum number of TCP connections and adopts probabilistic blacklisting of network traffic for mitigating the buffer saturation attack. Like AVANT-GUARD, LineSwitch requires an upgrade of data plane switches. Authors in Ref. [21] propose a SYN flood mitigation algorithm based on machine learning. The algorithm calculates the entropy of ports per each IP address and discerns anomaly traffic using K-Nearest Neighbors (KNN) method and CAIDA dataset. Experiment results show that the algorithm mitigates SYN Flood attack up to more than to 96% while normal traffic is almost not affected. SAFETY [22] is composed of two components: a detection component and a mitigation component. By calculating the entropy of both destination IP and specific TCP flags and comparing it with an adaptive threshold, SAFETY can complete attacker identification and stop ongoing TCP connections until corresponding SYN-ACK packets arrive from the server.

However, SAFETY lacks of the ability to deal with plenty of TCP half-open connections existing in the network. Authors in Ref. [23] use TCP timeout mechanism and Packet Round-Trip Time (RTT) against SYN flood attacks. The scheme drops the first SYN packet from a sender and calculates RTT based on the time between the first and the retransmitted SYN packets. If an ACK packet arrives later than the previous RTT, the ACK packet is dropped and this half-open session will be removed. SLICOTS [24] verifies the legitimacy of the connection establishment request using the TCP 3-way handshake process. When receiving a new SYN packet, SLICOTS extracts necessary information including MAC address and TCP ports into a pending list, which records uncompleted TCP connections. Moreover, SLICOTS calculates a path and instructs OF switches to forward the SYN packet. Once SLICOTS receives the ACK packet from the requested host, it recognizes that this is a normal TCP connection, and otherwise if the number of half-open TCP connections for the requested host exceeds a threshold, SLICOTS will install a flow rule for blocking anomaly traffic at the edge switch. However, SLICOTS does not take MAC spoofing into account and this issue will make SLICOTS inactive. SRL [25] is comprises a hash module and a flow aggregator module. For an incoming SYN packet, if its requests exceed the threshold, SRL will calculate the hash value based on IP address and maximum segment size (MSS) and use flow aggregator to install hash priority-based flow rules in edge switches for eliminating those fraudulent rules with lower hash values.

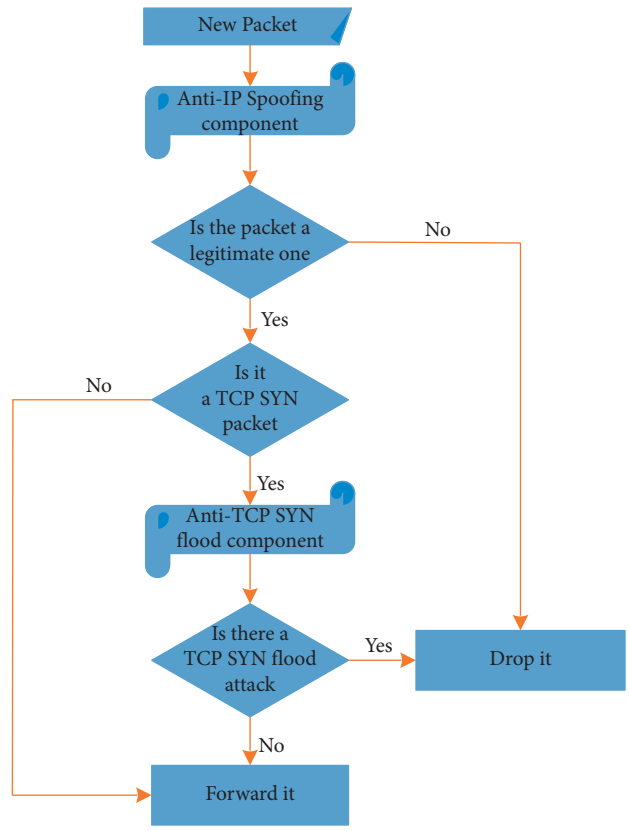

Figure 4: The workflow of SDNDefender.

3. Proposed Mechanism: SDNDefender

To address security challenges discussed in Section 2, we propose SDNDefender as a scalable, efficient, and defensive mechanism for countering IP spoofing attack and TCP SYN flood attack in SDN networks. Particularly, SDNDefender is implemented as an extension of the SDN controller to protect from cyber-attacks. As shown in Figure 4, SDNDefender consists of two components: Anti-IP spoofing and Anti-TCP SYN flood. The Anti-IP spoofing component is responsible for validating a packet's source address authenticity and its legitimacy. Anti-TCP SYN flood component conducts multistep detection on TCP flows to determine whether SYN flood traffic exists. In this section, we describe the design principles of each component in detail.

3.1. Anti-IP Spoofing Component. As shown in Figure 5, when receiving a new PACKET_IN message, the Anti-IP spoofing component validates the packet's source address and decides how to handle the packet. Besides, the component also checks destination host's performance for detecting unknown attacks as soon as possible.

3.1.1. Dynamical Entries Match. In SDN, topology discovery is an essential service and consolidates many higher layer services. On startup, the SDN controller requests configuration information from OF switches using OFP_FEATURES_REQUEST messages, and this initial switch-controller handshake tells the controller about the existence of the nodes in the network. Currently, most of the controllers leverage the Link layer discovery protocol (LLDP) [26] to explore the available links between switches in the network. LLDP is a neighbor discovery protocol working on the data link layer, which is composed of a header and a LLDP data unit (LLDPDU). The link discovery process is shown in Figure 6. The controller sends a LLDP packet to each port on switch S_1 via PACKET_OUT messages. Each of these LLDP packets includes Chassis ID and Port ID accordingly. Suppose that switch S_1 forwards a LLDP packet to port3 of switch S_2 via port2, and then S_2 puts its metadata and this packet together into a PACKET_IN message and sends it to the controller. The controller parses the PACKET_IN message and infers the existence of a link between (S_1, port2) and (S_2, port3) from the LLDPDU and the metadata. This process is repeated for each switch in the network, and as a result, the controller gets a global network view. For host discovery, we assume that host1 pings host2 and an ARP packet will be sent to the controller, which learns host1's IP address, MAC address, and ingress port. Due to the lack of knowledge of host2, the controller installs flow rules on S_1 to order it to flood the ARP packet, and this packet reaches S_2 via port3 and is received by host2. Similarly, host2 sends an ARP response packet to the controller, and thus the controller records host2's location and floods the ARP response packet to host1.

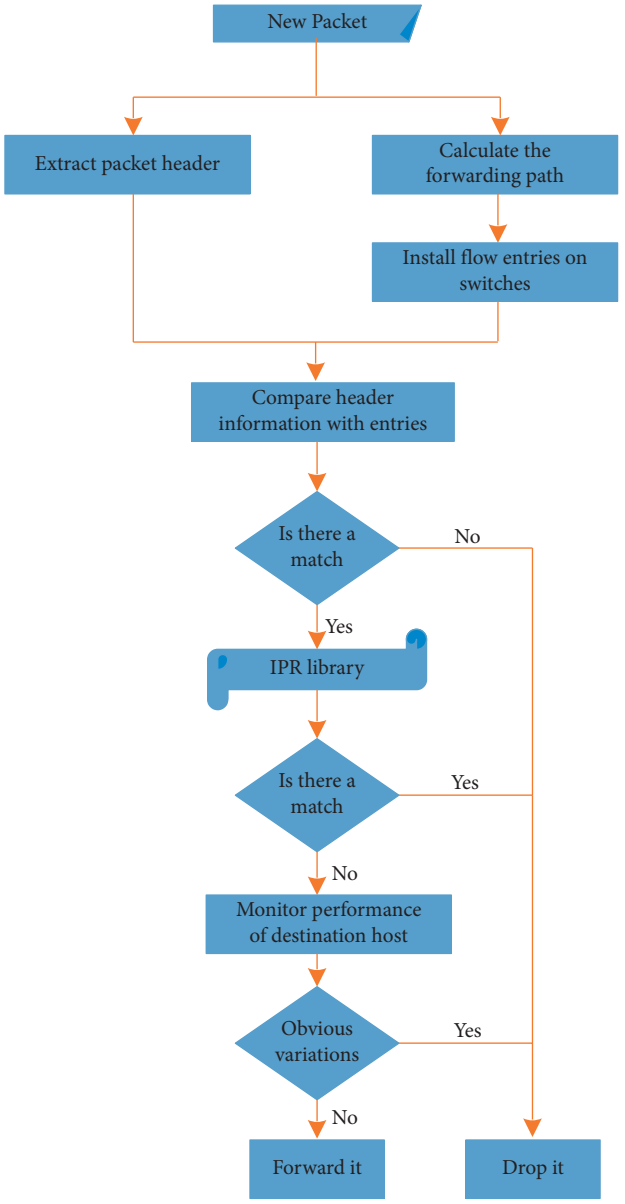

Figure 5: The workflow of the Anti-IP spoofing component.

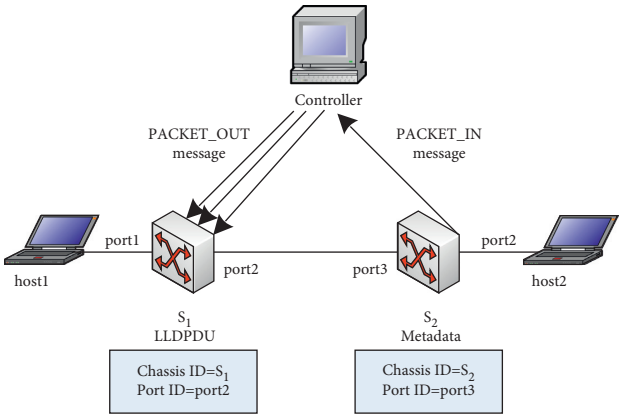

Figure 6: Topology discovery in SDN.

As described above, the controller can dynamically acquire the current network topology view and load status. More importantly, the controller can calculate a path between any two hosts in the network using some routing algorithms like open shortest path first (OSPF) and routing information protocol (RIP) [27]. Therefore, we present a DEM algorithm based on calculating the forwarding path. Table 1 lists notations used in this subsection.

To get the forwarding path of a pair of hosts, we select the shortest path algorithm called Floyd [28] to realize this goal. First, we calculate the link weight W. We use OFP_-PORTSTATS messages to periodically query transmission bytes of each port on each switch, and calculate bandwidth available of each link, which can denote W. For a link l ($l \in E$), we assume that in a transmission direction, l forwards b bytes in t seconds, so W_l can be depicted by Equation .

$$w_1 = C_1 - \frac{b}{t}. \tag{1}$$

Second, we define matrixes D and P as (2) and (3), respectively. In (2), we let d_{ij} ($1 \le i \le m$, $1 \le j \le m$) be the distance from switch i to switch j. Obviously, when $i = j$, $d_{ij} = 0$. If there exists a link between switch i and switch j, $d_{ij} = W_{ij}$. Otherwise, $d_{ij} = \infty$. In Equation (3), we let p_{ij} ($1 \le i \le m$, $1 \le j \le m$) be a node number on the shortest distance from switch i to switch j. If there is a link between switch i and switch j, $p_{ij} = i$, otherwise $p_{ij} = 0$.

$$D = \begin{bmatrix} d_{11} & \cdots & d_{1m} \\ \vdots & \ddots & \vdots \\ d_{m1} & \cdots & d_{mm} \end{bmatrix}, \tag{2}$$

$$P = \begin{bmatrix} p_{11} & \cdots & p_{1m} \\ \vdots & \ddots & \vdots \\ p_{m1} & \cdots & p_{mm} \end{bmatrix}. \tag{3}$$

Third, we update D by adding an intermediate element from V, and this process will generate $D^{(1)}$, where $d^{(1)}{}_{ij}$ can be described by Equation (4). By analogy, we can deduce $d^{(k)}{}_{ij}$ ($k = 2, \ldots, m$) as Equation 5. If $k = m$, $d^{(k)}{}_{ij}$ is the final shortest distance between switch i and switch j. On this basis, we can update $p^{(k)}{}_{ij}$ according to Equation (6). In Algorithm 1, matrix initialization is a 2-layer loop that is described from line 5 to line 8, and its time complexity is $O(m^2)$. The forwarding path calculation is a 3-layer loop that is described from line 10 to line 15, and its time complexity is $O(m^3)$. Therefore, the total time complexity of Algorithm 1 is $O(m^3)$.

$$d_{ij}^{(1)} = \mathrm{Min}\left\{d_{ij}^{(0)}, d_{i1}^{(0)} + d_{1j}^{(0)}\right\}, \tag{4}$$

$$d_{ij}^{(k)} = \mathrm{Min}\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\} \quad k = 2, \ldots, m, \tag{5}$$

$$P_{ij}^{(k)} = \begin{cases} k, & d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ P_{ij}^{(k-1)}, & \text{otherwise.} \end{cases} \tag{6}$$

As two hosts begin to communicate with each other, the controller records some necessary information, such as

Table 1: Main notations and descriptions.

Notation	Description
G	Network topology
V	Set of switches in the network
E	Set of links in the network
M	Number of switches
C	A link's capacity
W	A link's weight
D	The adjacency matrix of G
P	Path matrix

source/destination IP, source/destination MAC, ingress port, etc. Then, the controller calculates a forwarding path and divides the path into several 7-tuple flow entries and each flow entry can be defined as < source IP, source MAC, ingress port, datapath id, egress port, destination MAC, destination IP>. Finally, the controller deploys these entries into switches of the path. Figure 7 demonstrates how DEM algorithm deals with spoofed packets. When $host_1$ attempts to contact $host_4$, the controller will calculate the forwarding path and seriatim install flow entries on s_1, s_2, and s_3. For s_1, the flow entry is recorded as <10.0.0.1, 00 : 00 : 00 : 00 : 00 : 01, p_1, s_1, p_3, 00 : 00 : 00 : 00 : 00 : 04, 10.0.0.4>. However, if $host_2$ launches a conversation by disguising its IP address as $host_1$'s, then the flow entry of s_1 will be < 10.0.0.1, 00 : 00 : 00 : 00 : 00 : 02, p_2, s_1, p_3, 00 : 00 : 00 : 00 : 00 : 04, 10.0.0.4>, so the controller knows that $host_2$ forges an IP address. Similarly, if $host_3$ pretends itself as $host_1$ and communicates with $host_4$, the $route_3$ is not the same as the $route_1$, and the controller decides that the IP address is also illegal.

3.1.2. Intrusion Prevention Rules Library. On detecting the existence of IP spoofing, the Anti-IP spoofing component immediately sends an OFP_FLOWMOD message to the affected switch and drops forged packets according to the entry. However, even if a packet owns an authentic source address, it still needs to be examined for its legitimacy. For instance, an intra-host was infected by worms and started to send malformed packets. Usually, vicious packets are created by setting mistaken control bits, specific port, or oversize payload. Once these packets are received by targeted hosts, the hosts will have a breakdown immediately. Misuse detection searches for the appearance of specific traffic patterns in a ruleset of attack signatures. It can recognize all well-known cyber-attacks in an accurate and efficient way and the ruleset can be updated by capturing new forms of attacks. According to this analysis, we build an IPR library to validate intrusion behavior. IPR library consists of a big number of rules and each rule is made up of a rule header (RH) and some rule options (RO). The structures of a RH and a RO are shown in Tables 2 and 3, respectively. We can write a rule according to an attack's specific features. Taking the UDP Chargen attack for example, we define a rule as: "**alert udp any any - > any any (msg:'Chargen attack'; rid:10001; clstype:udp-event; filter: dst_port = 19;)**". This rule allows the Anti-IP spoofing component to check each UDP packet's destination port, and if there is an alert, the component will drop the packet with an OFP_FLOWMOD entry. Furthermore, we summarize common malformed packet attacks the IPR library can detect, which are listed in Table 4.

With the rapid increase of attack types, we need to periodically update the IPR library to raise the match speed and accuracy. However, considering that the match time of different attribute sequence varies widely, we design a three-layer rule tree to reduce unnecessary match operations. As shown in Figure 8, the rule tree contains protocol layer, port layer, and rule layer.

We classify rule protocols into 4 groups: IP, TCP, UDP, and ICMP. According to the type of rule port, we build three rule port sets by the following steps:

(1) If the source port is a specific value and the destination port is "any," we put the rule into the source port set.

(2) If the source port is "any" and the destination port is a specific value, we put the rule into the destination port set.

(3) If the source port and the destination port are both of "any," we put the rule into the general port set.

(4) If the source port and the destination port are specific values, we put the rule into both the source port set and the destination port set.

On receiving a packet, the Anti-IP spoofing component decides the port set according to the packet's protocol and ports. Then, the component matches the packet with each rule in the set in sequence. If there is a full match, the component drops the packet. Otherwise, the component will take further actions to detect attack behavior.

3.1.3. Host-Based Anomaly Detection. For some novel cyber-attacks, IPR library may lack the capacity to detect them because there are no corresponding rules in the library. This not only compromises targeted hosts but also exhausts their resources such as CPU and bandwidth. Usually, a host's security is closely related to its system resource usage. If the host's resources have been excessively consumed for some time, the host is subject to serious attacks. So, we propose a host-based anomaly detection algorithm to counter suspicious access or unknown attacks. First of all, we define a host resource availability system X that contains some indicators which reflect a host's performance from different aspects, such as CPU load, memory usage, system processes, bandwidth, I/O, etc. Second, we use SNMP protocol to collect these indicator values in nonattack scenarios and constitute n evaluation schemes. For each $X_i = (a_{1i}, a_{2i}, \ldots, a_{mi})^T$, where $i = 1, 2, \ldots, n$, a_{mi} denotes the mth indicator value of the ith scheme. Thus, we get an $m \times n$ indicator matrix EV. Meanwhile, we define normal value range of the jth indicator as $[u_j - d * s_j, u_j + d * s_j]$ $(j = 1, 2, \ldots, m)$, where u_j and s_j represent the mean and the standard variance of the indicator and d is a constant.

```
(1) Input: G, D, W, m, src, dst
(2) Output: path
(3) nextt = [][]
(4) Function initial ()
(5)    for i = 0; i < m; i++
(6)       for j = 0; j < m; j++
(7)          D[i][j] = W_ij
(8)          nextt[i][j] = j
(9) Function Floyd (G, m, nextt)
(10)   for k = 0; k < m; k++
(11)      for i = 0; i < m; i++
(12)         for j = 0; j < m; j++
(13)            if D[i][j] > D[i][k] + D[k][j]
(14)               D[i][j] = D[i][k] + D[k][j]
(15)               nextt[i][j] = nextt[i][k]
(16) Function Path (nextt, src, dst)
(17)   P = [src]
(18)   While src! = dst
(19)      src = nextt[src][dst]
(20)     P.append(src)
(21)   return P
```

Algorithm 1: Calculate the forwarding path.

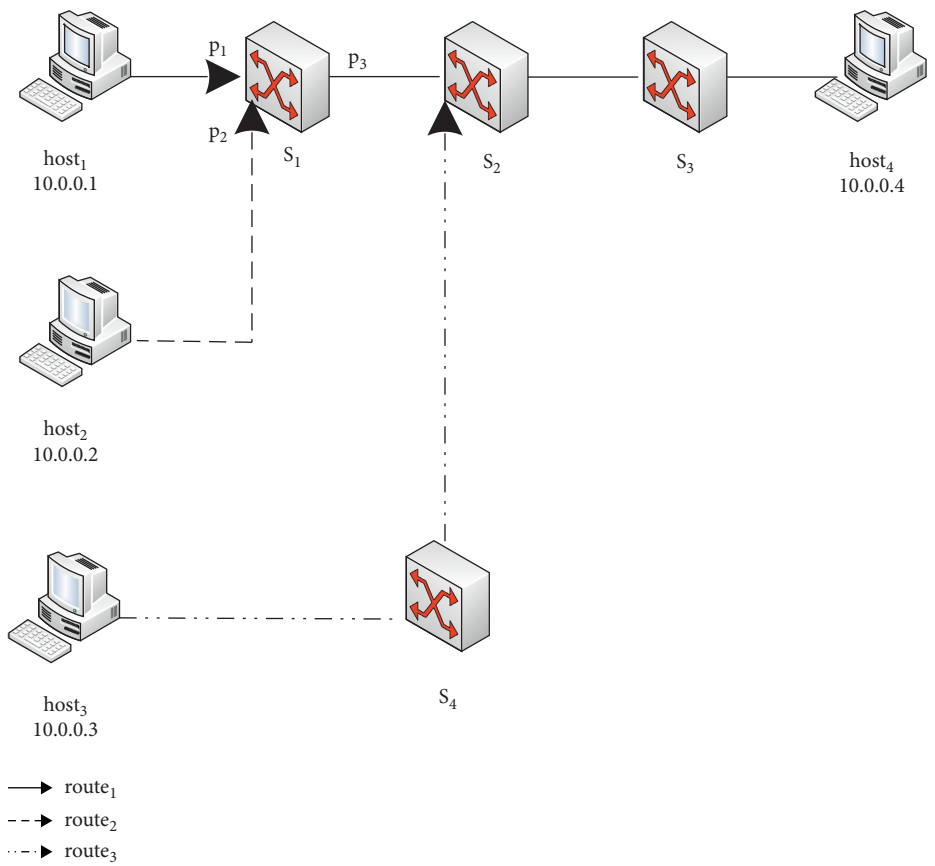

Figure 7: DEM algorithm for source address validation.

Table 2: The structure of a rule header.

Field	Value
Action	Alert, log
Protocol	TCP, UDP, ICMP, and IP
Source address	A specific IP address, if any
Source port	A specific port number, if any
Direction	->, < >
Destination address	A specific IP address, if any
Destination port	A specific port number, if any

Table 3: The structure of a rule option.

Keyword	Description
Rid	Rule ID
Msg	A message for rule matching
Pri	Rule priority
Clstype	The type of an attack event
Filter	Parameters for detecting an attack
Content	Targeted patterns in a packet's payload
Dsize	Payload size for matching
Nocase	Case-insensitive matching
Reg	Wildcard-supported matching

$$EV = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}. \tag{7}$$

In EV, indicators can be divided into two types: cost and profit. Cost indicators perform well as they have smaller values, which are opposite to the profit indicators'. For $X^* = (x_1^*, x_2^*, \ldots, x_m^*)$, if $x_i^* (i = 1, 2, \ldots, m)$ satisfies Equation (8), X^* is the best scheme in X. For $X_* = (x_{*1}, x_{*2}, \ldots, x_{*m})$, if $x_{*i} (i = 1, 2, \ldots, m)$ satisfies Equation (9), X_* is the worst scheme in X.

$$x_i^* = \begin{cases} \max\limits_{1 \le j \le n} \{a_{ij}\}, & x_i^* \in \text{profit}, \\ \min\limits_{1 \le j \le n} \{a_{ij}\}, & x_i^* \in \text{cost}, \end{cases} \tag{8}$$

$$x_{*_i} = \begin{cases} \max\limits_{1 \le j \le n} \{a_{ij}\}, & x_{*_i} \in \text{cost}, \\ \min\limits_{1 \le j \le n} \{a_{ij}\}, & x_{*_i} \in \text{profit}. \end{cases} \tag{9}$$

On this basis, we calculate relative deviation matrixes $R^* = (r^*_{ij})_{m \times n}$ and $R_* = (r_{*ij})_{m \times n}$. Next, we calculate c_i, the included angle cosine of r^*_{ij} and r_{*ij}. At last, we figure out each indicator's weight Φ_i. As shown in Equations (12) and (13), the bigger c_i is, the more important the ith indicator is, so it should be endowed with a higher weight.

$$r_{ij}^* = \frac{\left| x_i^* - a_{ij} \right|}{\max\{a_{ij}\}_{1 \le i \le m, 1 \le j \le n} - \min\{a_{ij}\}_{1 \le i \le m, 1 \le j \le n}}, \tag{10}$$

$$r_{*_{ij}} = \frac{\left| x_{*_{ij}} - a_{ij} \right|}{\max\{a_{ij}\}_{1 \le i \le m, 1 \le j \le n} - \min\{a_{ij}\}_{1 \le i \le m, 1 \le j \le n}}, \tag{11}$$

$$c_i = \frac{\sum_{j=1}^{n} r_{ij}^* \times r_{*ij}}{\sqrt{\sum_{j=1}^{n} \left(r_{ij}^*\right)^2} \sqrt{\sum_{j=1}^{n} \left(r_{*ij}\right)^2}} \quad i = 1, 2, \ldots, m, \tag{12}$$

$$\phi_i = \frac{c_i}{\sum_{i=1}^{m} c_i}. \tag{13}$$

To detect whether there exists unknown attack behavior, we need to evaluate the targeted host's current status through periodically collecting its performance indicators and calculating its abnormal degree $\mathrm{ad}(X)$. For an indicator x in X, We let $\mathrm{ad}(x)$ denote this indicator's abnormal value. If the observed value of x is in its normal value range, $\mathrm{ad}(x) = 1$, otherwise, $\mathrm{ad}(x) = 0$. Thus, we can get $\mathrm{ad}(X)$ by Equation (14) and if $\mathrm{ad}(X)$ is bigger than a threshold θ, the targeted host is in anomaly status. Furthermore, if the number of anomaly status exceeds a threshold ρ during a given time slot, the targeted host is suffering from attacks. Once potential attacks are detected, the Anti-IP spoofing component will log these events and send an OFP_PortMod message to provisionally block the switch port connected with the targeted host so as to ensure that the host can recover as soon as possible.

$$\mathrm{ad}(X) = \sum_{i=1}^{m} \phi_i \times \mathrm{ad}(x_i). \tag{14}$$

3.2. Anti-TCP SYN Flood Component. TCP SYN flood attacks that occurred in the SDN environment are similar to those in the legacy networks except for actions taken by the controller for incoming SYN packets. As illustrated in Figure 9, when the Anti-TCP SYN flood component receives a SYN packet, it tests the packet's transmission rate and decides whether there is a need for further detection. Once this packet is judged as an abnormal one, the component calculates current entropy based on destination address and network self-similarity index. As the two metrics exceed their thresholds, the component knows that there has been flood traffic in the network, so it will install flow entries to block ingress port and put those attack packets in flight into a low priority queue to limit bandwidth occupation.

3.2.1. Abnormal Detection for SYN Packet's Transmission Rate. In SDN, the controller uses OFP_FLOWSTATUS messages to obtain a flow's statistics in real time. The

Table 4: Common malformed packet attacks.

Attack	Protocol	Feature
IP option	IP	DF flag = 1, MF flag = 1 or DF flag = 1, offset > 0
Teardrop	IP	Offsets are overlapped or staggered
IP null	IP	Protocol field in the packet header is set to 0
CharGEN	UDP/TCP	Destination port is set to 19
Fraggle	UDP	Destination IP is a broadcast one, destination port is set to 7 or 19
Snork	UDP	Source port is set to 7, 19, or 135, destination port is set to 135
Smurf	ICMP	Source IP is set to a broadcast one
Ping of death	ICMP	Packet's length is bigger than 65535 bytes
Land	TCP	SYN packet's source IP is same to its destination IP
WinNuke	TCP	Destination port is set to 139, URG flag = 1
TCP option	TCP	SYN flag = 1, FIN flag = 1 or FIN flag = 1, ACK flag = 0 or SYN = ACK = FIN = RST = PSH = 0

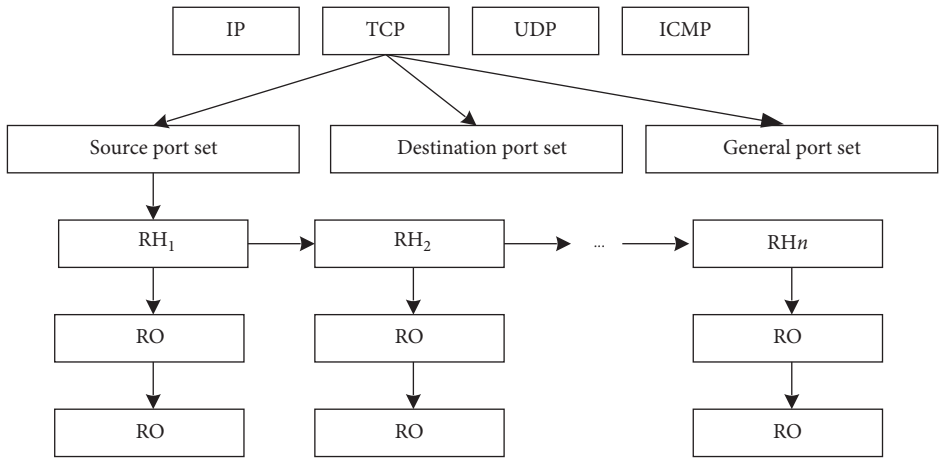

Figure 8: A three-layer rule tree.

status information includes transmitted packets, transmitted bytes, duration time, etc. On this basis, the controller can calculate the instantaneous transmission rate of the flow. $A(t)$ and $F(t)$ are functions about variable t From t-$T/2$ to $t + T/2$, $F(T)$ can be expressed as the following equation.

$$F(T) = \frac{1}{T}\int_{t-T/2}^{t+T/2} A(t)\mathrm{d}t. \tag{15}$$

Upon this condition, we leverage the CIC-DDOS2019 dataset [29] to build a practical flow throughput sample that comprises many elements. CIC-DDOS2019 contains benign and the most up-to-date common DDoS attacks and also includes the results of the network traffic analysis. We let S denote the sample and select n TCP records as sample elements from the SYN subset. When the value of n is big enough, we consider that S nearly coheres with normal distribution. For each record, we can calculate the flow's goodput via the bytes the flow has transmitted and the duration of the flow. Thus, we define S as $S = \{s_1, s_2, \ldots, s_n\}$. If μ and σ^2 represent the mean and the variance of S, respectively, the probability density function of S can be expressed as

$$f(s) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(s-\mu)^2}{2\sigma^2}\right), \tag{16}$$

$$\mu = \frac{1}{n}\sum_{i=1}^{n} s_i, \tag{17}$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(S_i - \mu). \tag{18}$$

For a new TCP flow fl, the controller will examine whether fl's goodput is abnormal in real time. For S, we define θ $(0 < \theta < 1)$ as the significance level. Given a parameter ϑ, if $P\{x_1 < \vartheta < x_2\} = 1 - \theta$, $[x_1, x_2]$ is the confidence interval of ϑ. Due to $S \sim N(\mu, \sigma^2)$, we can get $\bar{S} \sim N(\mu, \sigma^2/n)$ according to the central-limit theorem. We define a random variable as $\overline{S} - \mu/\sqrt{\sigma^2/n} \sim N(0, 1)$, and solve Equation (19) to determine the confidence interval of μ, namely $[\overline{S} - \sigma/\sqrt{n}\,Z_{\theta/2}, \overline{S} + \sigma/\sqrt{n}\,Z_{\theta/2}]$. After this, we calculate $F_{\mathrm{fl}}(t)$ and if the value is not in the confidence interval, the value is an outlier. The Anti-TCP SYN flood component will launch further actions to validate the onset of SYN flood traffic.

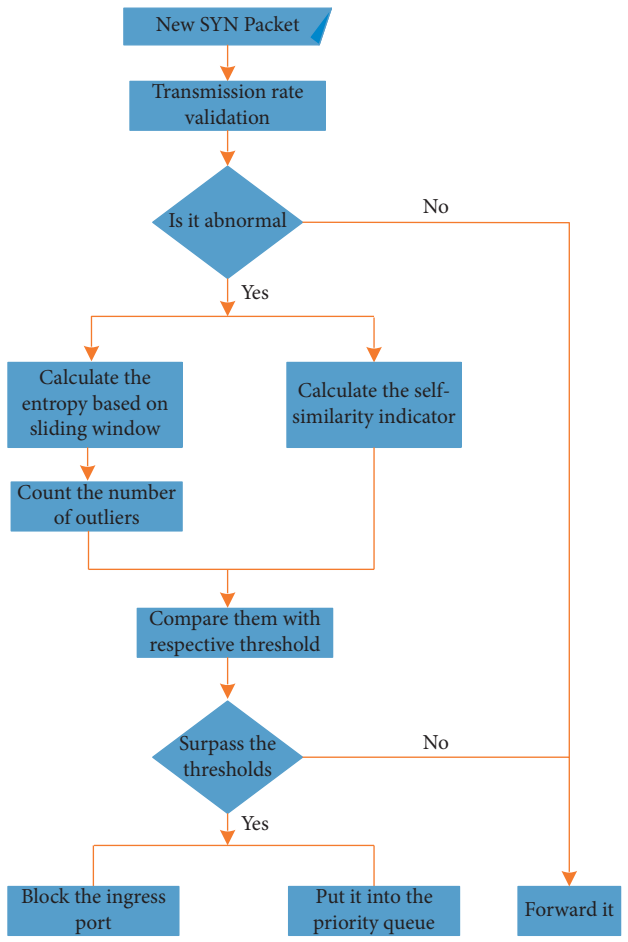

Figure 9: The workflow of the Anti-TCP SYN flood component.

$$P\left\{\left|\frac{\overline{S}-\mu}{\sqrt{\sigma^2/n}}\right| \le Z_{\theta/2}\right\} = 1-\theta. \tag{19}$$

3.2.2. Measuring Entropy Based on Destination IP Address. The concept of entropy comes from information theory and is used to measure the randomness associated with a random variable. According to the Shannon definition, if a random variable Z owns n values whose happening probability are ψ_1, ψ_2, . . ., ψ_n, respectively, the entropy of Z can be expressed as Equation (20). Among the n values of Z, when only one value's happening probability equals 1 and all other values' happening probabilities are equivalent to 0, Z has a minimum of 0. When $\psi_1 = \psi_2 = \ldots = \psi_n = 1/n$, Z has a maximum of $\log_2 n$.

$$Z = -\sum_{i=1}^{n} \psi_i \log_2 \psi_i. \tag{20}$$

In the case of nonattack scenarios, hosts communicate with each other, and packets are randomly distributed in the network. However, under TCP SYN flood attacks, there will be plenty of SYN packets that own the same destination IP address in the network, so in such scenarios, the packet randomness is limited due to the nature of the attack. The Anti-TCP SYN flood component employs a sliding window to collect destination IP addresses and calculates the current window entropy. The bigger the entropy value is, the bigger the packet randomness is, and vice versa. Table 5 lists main notations used in this subsection.

Table 5: Main notations for entropy calculation.

Notation	Description
I	Set of destination IP addresses
Win	The current window
I_f	The first IP address in the current window
I_o	The first subsequent IP address out of the current window
ω	A destination IP address's happening probability
e	The current window's entropy value
e_f	I_f's entropy value under the current window
e_o	I_o's entropy value under the current window
es	Normal window entropy series
a	The mean of es
g	The triple standard deviation of es
λ	Outlier threshold

We describe the window entropy calculation process according to Figure 10. In Figure 10(a), the window includes 4 packets: p1, p2, p4, and p1. After moving towards right, the window, in Figure 10(b), removes the first packet p1 and adds a new packet p3. Thus, packets going into the current window are p2, p4, p1, and p3. This repeatable process is the workflow of the sliding window.

More generally, for a sliding window win, we assume that its capacity is η. If there are n ($n \le \eta$) different IP addresses in I_{win} and their emergence probability is denoted as $\{\omega_i | i = 1, 2, \ldots, n\}$, we can figure out e_f, e_o and e_{win} in terms of Equation (20). Next, we move the window to right, and thus I_f will be cleared and I_o is added into the new window I'_{win}. Therefore, we need to recalculate ω'_f and ω'_o. The calculation for ω'_f and ω'_o can be expressed as Equations (21) and (22), respectively. After these, we calculate e'_{win} according to Equation (23). Finally, we can get a window entropy series es by implementing the above process repeatedly.

$$\omega'_f = \begin{cases} \omega_f, & I_f = I_0, \\ \omega_f - \dfrac{1}{\eta}, & I_f \neq I_0, \end{cases} \tag{21}$$

$$\omega'_0 = \begin{cases} \dfrac{1}{\eta}, & I_0 \notin I_{\text{win}}, \\ \omega_0, & I_0 \in I_{\text{win}}, I_f = I_0, \\ \omega_0 + \dfrac{1}{\eta}, & I_0 \in I_{\text{win}}, I_f \neq I_0, \end{cases} \tag{22}$$

$$e'_{\text{win}} = e_{\text{win}} - e_f - e_0 + e'_f + e'_0. \tag{23}$$

As aforementioned, under nonattack scenarios, there is only a little bit of change among window entropies, so we can take advantage of some software tools like TCPreplay to reimplement the flow throughput sample S and thus obtain a normal window entropy series es. To determine the onset of SYN flood attack, we adopt PauTa criterion to examine each newly generated window entropy. PauTa criterion adopts a given confidence probability of 99.7% to detect outliers

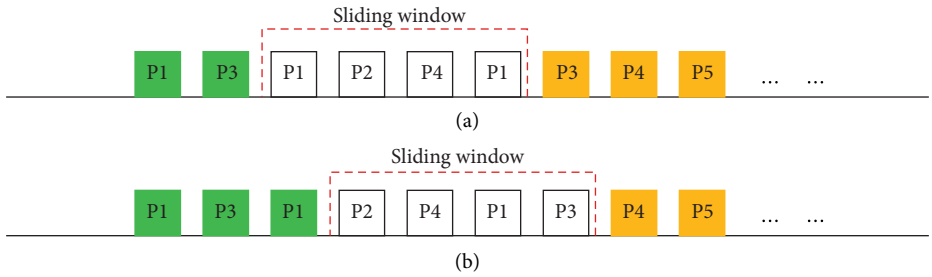

Figure 10: The workflow of the sliding window. (a) Initial window. (b) Window after moving towards right.

```
(1) Input: es, e_new, a, g, λ
(2) Output: result
(3) counter = []
(4) result = ''
(5) while True:
(6)     put e_new into es
(7)     if abs(e_new − a) > g
(8)         mark e_new as an outlier
(9)         counter.add("1")
(10)        remove e_new from es
(11)    else
(12)        counter.add("0")
(13)    if counter.length < λ
(14)        pass
(15)    else
(16)        for i = 0; i < counter.length + 1 − λ; i++
(17)            if counter.find(i, λ, "1") == True
(18)                result = "done"
(19)                break
(20)    if result == "done"
(21)        break
(22) return result
```

Algorithm 2: Examination for SYN flood traffic.

according to the triple standard deviation of a group of normal data. As shown in Algorithm 2, when verifying a new window entropy e_{new}, we put it into es and calculate the absolute value of the difference of both e_{new} and a. If the absolute value is bigger than g, e_{new} is regarded as an outlier and will be removed from es. Meanwhile, we update the anomaly counter for recording the outlier status. If there are consecutive outliers and the count of outliers reaches the threshold λ, the Anti-TCP SYN flood component will acquire current status of the network self-similarity index. This is because only using entropy to detect flood traffic may lead to a misjudgment. For example, some services in the data centers like Hadoop and MapReduce need multi-client to communicate with a single server simultaneously. Under this many-to-one pattern, the destination IP-based entropy will be similar to that under attack scenarios.

3.2.3. Calculating the Network Self-Similarity Index. A lot of research literatures have proved that network traffic owns an attribute of self-similarity. Suppose that $X = \{X_i \mid i = 1, 2, \ldots\}$ is a stationary random process, we let μ, σ^2 and $f(k)$ $(k = 0, 1, \ldots)$ be the mean, variance, and autocorrelation function of X, respectively. We assume that $f(k) \sim k^{-\beta} L(k)$, $k \longrightarrow \infty$, $0 < \beta < 1$ and $\forall\ x > 0$, $\lim L(tx)/L(t) = 1$ $(t \longrightarrow \infty)$. We let $X^{(m)}$ be the m-order sequence of X where $X^{(m)}{}_i = (X_{im-m+1} + \ldots + X_{im})/m$ $(m = 1, 2, \ldots)$ and also let the autocorrelation function of $X^{(m)}$ be $f^{(m)}(k)$. If $f^{(m)}(k) = f(k)$, X is a self-similarity process that possesses a unique self-similarity index H, which is also called Hurst index and is equivalent to $1 - \beta/2$. Under nonattack scenarios, network traffic keeps its self-similarity stationary, yet encountering SYN flood attacks, and the Hurst index will obviously descend. Therefore, we can continuously monitor the variation of the Hurst index and detect anomaly traffic. Considering that SYN flood traffic sends a huge number of packets in several seconds, we adopt each packet's arrival time to calculate the index H. Among different solutions, we select Rescaled Range (R/S) analysis to achieve this goal and describe the calculation process as follows:

(1) We set a fixed size time window to collect each SYN packet's arrival time and thus get a time series.

(2) We divide the series into $v$parts $\{I_i \mid i = 1, 2, \ldots, v\}$.

(3) For each part $I_i = \{I_{a,i} | a = 1, 2, \ldots, n\}$, we let q_i and s_i denote their the mean and standard variance. We create another series of deviations for each part by $I'_i = I_{a,i} - q_i$.

(4) We calculate a running total for each part's deviations from the mean according to $\sum_{i=1}^{v} I'_i$.

(5) As shown in Equation (24), we pick up both the maximum and minimum values in the updated series of each part to calculate the widest difference. Thus, we can get the rescaled range of each part. That is, y_i/s_i.

$$y_i = \max\{I'_i\} - \min\{I'_i\} \quad i = 1, 2, \ldots, v. \tag{24}$$

(6) We get a rescaled range series $\{y_i/s_i | i = 1, 2, \ldots, v\}$. The mean of this series is $(y/s)_v = 1/v \sum_{i=1}^{v} y_i/s_i$. Obviously, as v is set as different values, $(y/s)_v$ is a function about v. Many researches show that the relationship between v and $(y/s)_v$ can be expressed as $(y/s)_v = cv^H$, where c is a constant. We take the logarithm on both sides of this formula, and it will be Equation (25). For all [log v, log $(y/s)_v$], we employ Ordinary Least Squares to fit a straight line and the slope of this line is index H.

$$\log\left(\frac{y}{s}\right)_v = \log c + H \log v. \tag{25}$$

We can monitor sequential windows and calculate their Hurst values to detect the variation of network traffic. We set two thresholds $\pounds_1$ and $\pounds_2$. For two adjacent Hurst values h_1 and h_2, if $|h_2 - h_1| \geq \pounds_1$ and $h_2 < \pounds_2$, network traffic is under abnormal state. As the entropy and the Hurst value of network traffic all overstep the respective threshold, the Anti-TCP SYN flood component decides the onset of SYN flood attacks and takes safeguards to mitigate attack influences.

3.2.4. DiffServ-Based Mitigation Method for SYN Flood Attack. Under TCP SYN flood attacks, there is a mixture of flows in the network, so how to separate anomaly traffic from normal traffic timely is a major concern. There has been a typical technique called traffic redirection. This technique manages a pool of public IP addresses and selects another IP address in the same subnet for the victim so that normal traffic could be redirected from the attacked address to the new address. However, in large-scale networks, selecting an additional IP address for new route calculation and traffic redirection may bring a long network latency and a very high system cost. Fortunately, SDN provides lots of controller-to-switch messages to enable users deploy QoS strategies on OF switches according to their demands. The Anti-TCP SYN flood component adopts the DiffServ model [30] to deal with anomaly traffic. That is to say that the component carries out traffic classification and management only on edge switches and implements traffic scheduling algorithms on inner switches.

(1) Traffic Classification. As described before, SDNDefender calculates a path for a pair of hosts by extracting the packet header. Therefore, we define a TCP flow as <source IP, ingress port, egress port, destination IP, and status>. Ingress port means the port connected with the source host and egress port means the port linked to the destination host. The parameter "status" decides the flow's attributes. If its value is 0, the flow is normal. Otherwise, if the value is 1, the flow is considered as an anomaly one. For all normal flows, the Anti-TCP SYN flood component will set the highest priority in flow entries and forward them. On the contrary, for each abnormal flow, the component changes a flow entry's priority field to the lowest value and rewrites actions as "Drop," and then this flow entry is installed on the flow's ingress port. As a consequence, a majority of attack packets of the flow can be directly dropped on the source host side. However, there is still a small quantity of attack packets forwarded to the destination host, because of the workflow of detecting SYN flood attacks; so, the Anti-TCP SYN flood component needs to deal with these packets and keep normal flows unaffected.

3.3. Queue Scheduling. Queue scheduling is a major method for solving competition on shared resources among multiple data flows. To mitigate the effect of the remaining attack traffic in the network, we adopt the priority queue-based scheduling strategy to ensure that normal traffic can be firstly dealt with. Priority queue can lead the classified flows into the queues with different priorities and forward the flows according to the level of their priorities. As illustrated in Figure 11, on the switch's egress port connected with the destination host, we create two priority queues: QN for normal traffic and QA for attack traffic. SDN provides a SET_QUEUE action to forward a packet to the queue to which the packet belongs, so when installing flow entries on the egress port, the Anti-TCP SYN flood component sets a high priority value for QN as well as a low priority value for QA. Thus, scheduling on QA has to wait for the completion of that on QN.

3.4. Traffic Shaping. To decrease bandwidth waste of the queue QA, there is a need to implement traffic shaping on QN and QA to smooth packet transmission and reduce traffic burst. We assume that there are x anomaly flows arriving at the egress port at a moment t, and each such flow's throughput is expressed as $\{\varepsilon_i | i = 1, 2, \ldots, x\}$. If the egress port's output bandwidth is B, QN's available bandwidth rate γ_t can be expressed as Equation (26). To allocate more bandwidth resources for the queue QN, we have to restrain QA's bandwidth occupation, so we preset a bandwidth factor ξ $(0 < \xi < 1)$ for QA. We compare $1 - \gamma_t$ with ξ and if the former is less than the latter, we reset ξ as $1 - \gamma_t$. Thus, the maximum bandwidth QA can take is $B\xi$. Correspondingly, available bandwidth QN can obtain is $B(1 - \xi)$. At present, SDN offers many application interfaces to realize traffic engineering on OF switches, so we can easily implement traffic shaping towards output traffic.

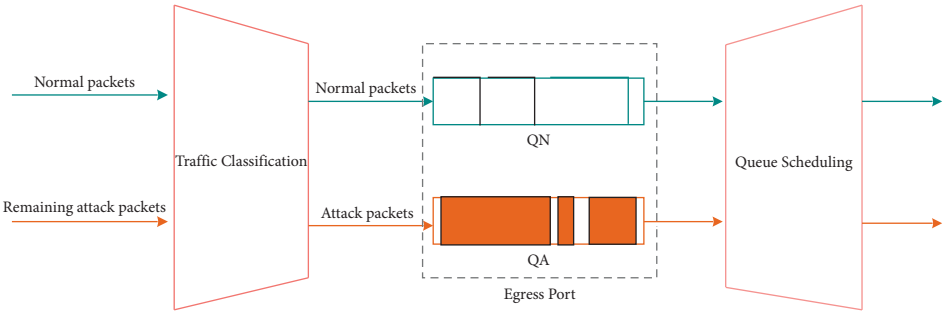

Figure 11: Queue scheduling for different kinds of flows.

$$\gamma_t = \frac{B - \sum_{i=1}^{x} \varepsilon_i}{B}. \tag{26}$$

4. Performance Evaluation

In this section, we carry out a comprehensive simulation study and analyse the results to evaluate the performance of SDNDefender against other defense mechanisms and normal SDN. We will demonstrate how SDNDefender lifts the security of normal SDN.

4.1. Experimental Setup and Simulation Scenarios. We compile SDNDefender as a security module into the RYU controller, an open source SDN controller platform written in Python language. To simulate the data plane of our experimental network, we make use of Mininet, which can be rapidly deployed into hardware environments and emulate complex topologies that include a huge number of hosts and network devices. We choose Ubuntu 14.04 LTS as the system environment because this Linux-based operation system provides lots of third-party plugins for developing SDN applications and complex data processing. As seen from Figure 12, we build a 4-pod Fattree topology with 4 core, 8 aggregation, and 8 Top-of-Rack (ToR) switches and 8 virtual hosts using a testbed that owns a dual-core processor with 3.2 GHz, 4 GB memory, and 250 GB disk space. In this topology, OF switches and virtual hosts are connected via 10 Mbps links and the default latency of each link is set as 2 ms.

Our simulation experiments can be divided into two groups: one for IP spoofing attack and another for TCP SYN flood attack. For each group experiment, we classify experimental data from the angles of accuracy, True positive rate (TPR) and False positive rate (FPR). There are several relevant concepts as follows:

(1) True Positive (TP): SDNDefender correctly identifies positive class.

(2) True Negative (TN): SDNDefender correctly identifies negative class.

(3) False Positive (FP): SDNDefender incorrectly identifies negative class.

(4) False Negative (FN): SDNDefender incorrectly identifies positive class.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}},$$

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{27}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

In the first group experiment, we set the run time of the simulation scenario as 40 s. We compare SDNDefender with normal SDN, SIPAV-SDN, and DDAH, using diverse performance metrics. Specially, we use CIC-DDOS2019 dataset and the random forest algorithm for DDAH to detect anomalies.

(i) Scenario 1. We categorize virtual hosts into benign users, attackers, and the server. Next, we let benign users and attackers to concurrently communicate with the server. As shown in Table 6, for benign ones, we use Ping command to generate legal connection requests, and for attackers, we adopt Hping to falsify their IP and implement specific attacks.

In the second group experiments, we will simulate TCP SYN flood attack. Considering that flood traffic can quickly exhaust target's system resources and break it down, we set simulation time as 70 s and only record valid data for the first 40 s. We compare SDNDefender with SRL, SLICOTS, and normal SDN from many aspects.

(ii) Scenario 1. Firstly, we let all virtual hosts be benign ones and each host starts a TCP conversation with another one in a random way. This process lasts for the first 10 s. From then on, we let H_1, H_2, H_3, and H_4 be attackers and let H_5, H_6, and H_7 be benign ones. We use Scapy to send legitimate TCP SYN packets and employ Hping to launch SYN flood attacks towards the server H_8. Table 7 lists simulation parameters for this scenario.

(iii) Scenario 2. We choose H1–H6 as potential attackers and gradually increase an attacker in each simulation.

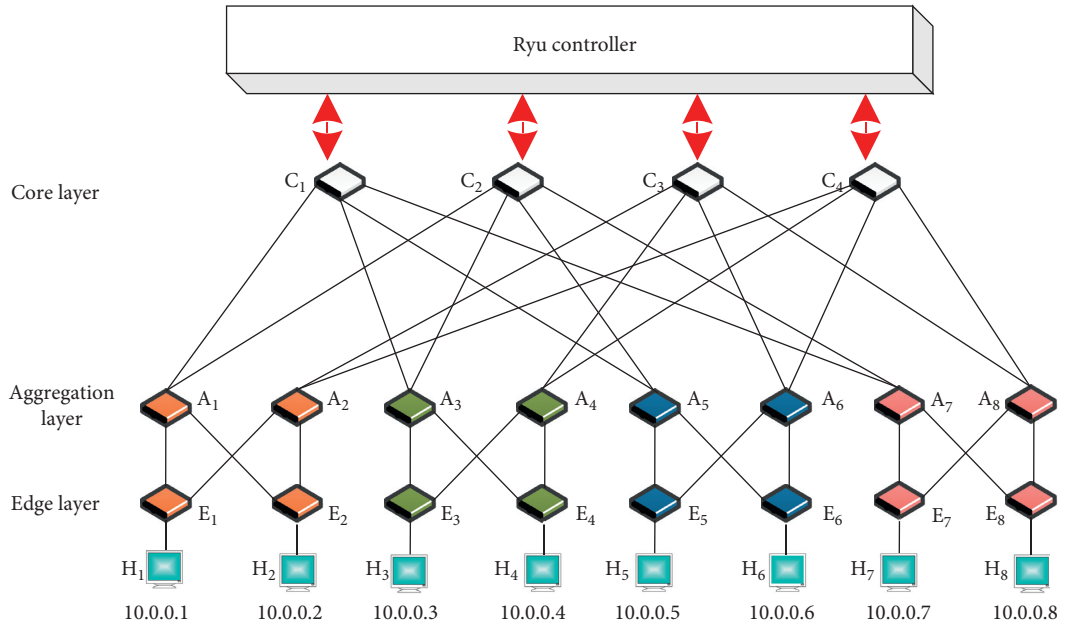

Figure 12: The testbed network topology.

Table 6: Settings for emulating IP spoofing attack.

Host	Role	Operation	Packet (count/size)
H_1	Benign	Ping command	20/128 bytes
H_2	Benign	Ping command	20/128 bytes
H_3	Benign	Ping command	10/256 bytes
H_4	Attacker	DNS amplification attack	200/64 bytes
H_5	Attacker	ARP spoofing	200/64 bytes
H_6	Attacker	Land attack	100/40 bytes
H_7	Attacker	BlackNurse attack	200/64 bytes
H_8	Server	Handle connection requests	None

Table 7: Settings for emulating TCP SYN flood attack.

Parameter	Value
TCP SYN packets, a benign host sends	30
TCP SYN packets, an attacker sends	1000
The payload of a TCP SYN packet	64 bytes
The capacity of the sliding window	6
The threshold of sequential entropy outliers	3
The difference threshold of two hurst indexes	0.12
The normal hurst threshold	0.5
The significance level of the flow goodput set	0.05
The confidence upper limit of the flow goodput set	10000 bps
Bandwidth factor for anomaly traffic	0.1
Limit of acceptable SYN requests in SRL	200
Limit of replaced flow entries in SRL	15
Illegal request threshold of a host in SLICOTS	50

4.2. Results Analysis

4.2.1. Experimental Results for IP Spoofing Simulation. In Figure 13(a), the APPF curves continuously fluctuate with the elapse of simulation time. We can see that SDNDefender keeps its APPF values much lower than that in the other three situations. The main cause of this phenomenon is that normal SDN does not provide any defense measure to counter IP spoofing, so that a massive number of attack packets can be directly forwarded to the server without further validation by the controller. Although SIPAV-SDN can block spoofed packets, it cannot filter ARP spoofing due to its imperfect way of validation. DDAH decides an anomaly host by updating access port's weight and activity and evaluating a flow's status with several selected features. Obviously, DDAH will launch the detection process until a port's activity exceeds the threshold. However during this period, there has to be lots of attack packets in the network. Figure 13(b) shows that each situation's MFE value gradually increases with the course of experiments. It is clear that the timeliness and detection accuracy of each algorithm decide the number of vicious entries. In normal SDN, there is no countermeasure against attackers, so the anomaly entries increase fast. SIPAV-SDN keeps its flow table much capacious than DDAH, because the former can drop most of the attack packets using a host table, yet the latter needs some time to find suspicious ports and train specific features to detect attacks. On the contrary, SDNDefender strikingly eliminates attack entries, because it implements route-based validation for each flow before forwarding it, and thus, IP spoofing and ARP spoofing can be blocked in advance, so that the controller dedicates itself to matching normal packets. Figure 13(c) displays the variation of the server's CPU load under different situations. Obviously, the four situations get a maximum CPU load of 1.06, 1.22, 1.33, and 1.27, respectively. In normal SDN, a huge number of attack packets quickly consume the server's CPU resources and impose a heavy load on it. For SIPAV-SDN, it drops the majority of attack packets with illegal IP addresses while forwarding vicious ARP packets with legitimate IP and MAC addresses. This also increases the server's CPU usage rate.

Compared with such three situations, SDNDefender significantly reduces the server's work load due to its all-round validation for spoofed packets and quickly blocking attack traffic on ingress ports. IP spoofing also influences the server's receiving capacity. As shown in Figure 13(d), under normal SDN, SIPAV-SDN, and DDAH, there is a mixture of legitimate packets and attack packets, which surges to the server, so the received byte curve greatly ascends until the attack traffic becomes faint. In contrast, SDNDefender keeps the server's receiving traffic below 270 bps.

As shown in Figure 14 and Table 8, SDNDefender performs much better in attack detection than other two algorithms do. SDNDefender not only blocks spoofing attacks but also provides the ability to defend those specific DoS attacks. On the contrary, SIPAV-SDN fails to detect ARP spoofing. This is why its detection accuracy and FPR are poor. For DDAH, its detection effect relies on the delay of recognizing an abnormal port and checking a flow's status. In another word, the delay imposes a significant influence on the detection accuracy and true positive rate.

4.2.2. Experimental Results for TCP SYN Flood Simulation. Figure 15 shows the variation of the entropy and the Hurst index under normal SDN and SDNDefender, as TCP SYN flood attacks are launched. In Figure 15(a), the entropy value fast descends from 10 s under normal SDN. This suggests that as SYN flood attacks are launched, the window entropy becomes much smaller and IP addresses in the current window are almost the server's IP address. By comparison, the entropy decreases little by little from 10 s to 20 s with SDNDefender, and then keeps a stationary state, in which the entropy value is much bigger than that in normal SDN. This is because SDNDefender directly drops most of the attack packets on ingress ports and ensures normal traffic is at a predominant level. In Figure 15(b), the SYN flood traffic makes the network self-similarity degree obviously go down from 12 s and network traffic shows inverse persistence. As SDNDefender detects the onset of flood traffic at 14 s, it reduces the proportion of anomaly traffic in the network as much as it could. Therefore, the Hurst curve returns a normal tendency again.

As illustrated in Figure 16, SDNDefender shortens the first successful detection time by 15% and by 21%, respectively, compared with SLICOTS and SRL. SLICOTS judges a TCP request's legitimacy by monitoring the completeness of the TCP three handshakes, and this process influences the timeliness of attack detection. SRL detects flood traffic by measuring both SYN requests and the number of replaced entries. This obviously increases detection time due to entries update inconsistency in switches and simplicity of flow's hash calculation. On the contrary, SDNDefender not only measures SYN packet transmission rate but also calculates network entropy and self-similarity index in real time, and thus flood traffic can be detected fast.

Figure 17 demonstrates the variation of AFE under different situations. It is clear that SDNDefender sustains its AFE at a much lower level, because SDNDefender can quickly discern SYN flood attack and block anomaly traffic on edge switches on time. This notably decreases flow entry installations. However, SLICOTS and SRL block a source address only when its TCP illegal requests exceed the threshold. During this period, there have been a lot of flow entries in switches.

In Figure 18, SDNDefender brings a stationary variation on SYN packets' transmission rate due to its port-based packet filtering and priority queue scheduling. Conversely, under normal SDN, the curve values continuously rise until there are a small number of flows left in the network. The main cause is that when SYN flood attacks occur, the proportion of SYN packets and ACK packets becomes seriously unbalanced. SLICOTS examines whether a TCP connection successfully establishes in a given time, and if not, this TCP request is seen as an illegal one; however, this method cannot ensure that all flood traffic is detected because those temporary flow rules expire at a given time, in which some TCP half-open connections have turned into the establishment state. For SRL, it needs to judge whether the replaced flow entries surpass the threshold after updating the SYN counter. Therefore, if SRL cannot detect the onset of SYN flood attacks in time, there will be a huge number of SYN packets in the network in several seconds.

In Figure 19, each situation's TCP half-open connection curve steeply varies owing to the timeout mechanism of the TCP protocol. SYN flood traffic sends tremendous SYN packets in several seconds and leads the server to be busy in answering with SYN + ACK packets and waiting for ACK packets. At this moment, the server generates many new TCP half-open connections and puts them into its SYN queue. Thus, those old ones will be released from the queue once their survival time is over. It is distinct that SDNDefender enables the number of TCP half-open connections to be in a rational range because of its quick detection and efficient mitigation towards SYN flood attack. On the contrary, SLICOTS and SRL suffer from detection delays, which do not let the server spend time removing the half-open connections, so the server has to wait for the timeout of these connections.

As shown in Figure 20, SDNDefender reduces the server's response time by 84% and by 68%, respectively, compared with SLICOTS and SRL. This is because SLICOTS and SRL have certain detection delays, and during these periods, flood traffic still largely consumes the server's resources, so each TCP request's round time will be much longer.

In Figure 21, SDNDefender and SRL keep their detection accuracy above 85% and 70%, respectively. However, SLICOTS's detection efficiency descends along with the increase of attackers. The main cause is that SLICOTS judges a TCP request's legitimacy through monitoring the completeness of TCP handshakes within the threshold time, yet this time is big enough so that more half-open connections turn into establishment state; therefore, there exists a high false positive rate.

Tables 9 and 10 show each algorithm's TPR and FPR, respectively. We can see that SLICOTS' FPR is much higher than that in SRL and SDNDefender. Similarly, SDNDefender's TPR performs much better than the other two algorithms'. This is to say that real-time attack detection and

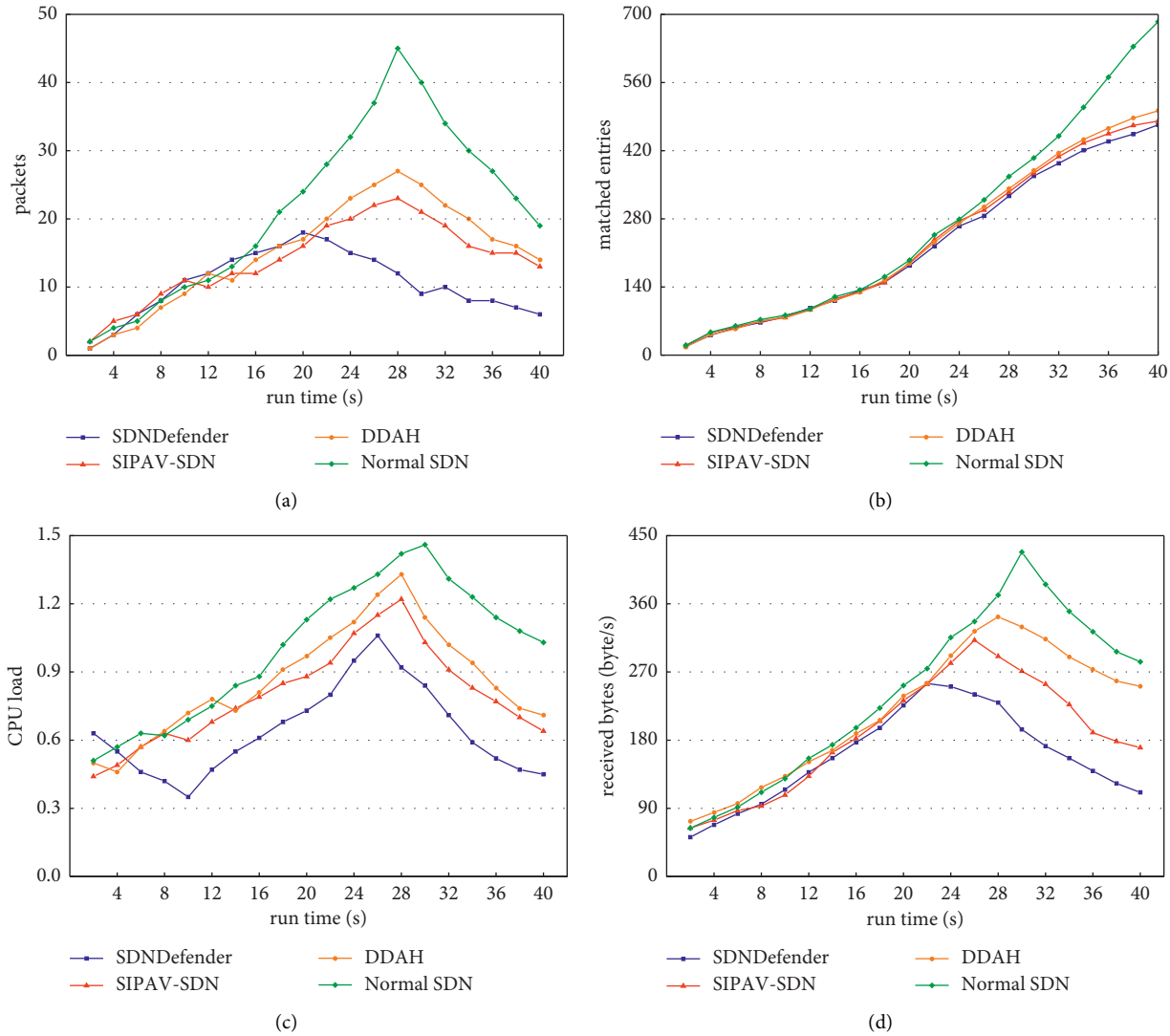

Figure 13: Metrics comparison of Scenario 1. (a) Average packets per flow. (b) Matched flow entries. (c) The server's CPU load. (d) The server's received bytes.

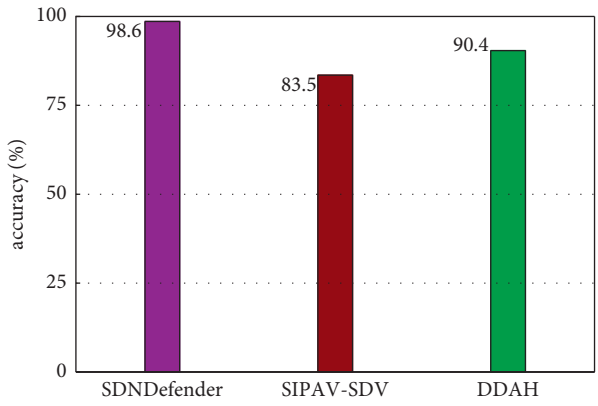

Figure 14: Comparison of attack detection accuracy.

detection method are two critical factors to improve detection accuracy, and the higher the detection delay is, the lower the TPR is, and vice versa.

Table 8: Detection metrics.

Algorithm	TPR	FPR
SDNDefender	0.98	0.0029
SIPAV-SDN	0.96	0.29
DDAH	0.88	0.073

5. Limitation

There are several factors that affect our experiments. First, we use a single testbed and a DDoS attacks dataset to carry out simulations. This experimental topology is different from practical data center environments. Moreover, real network traffic usually appears more distinct self-similarity and burstiness. These features test the network's robustness and the SDN controller's performance. Second, we set some thresholds for proposed algorithms in the literature to trigger detection activities. These threshold values decide detection timeliness and accuracy.

Table 9: True positive rate.

Attackers	1	2	3	4	5	6
SLICOTS	0.66	0.84	0.65	1	0.8	1
SRL	0.87	0.76	0.8	1	0.69	0.84
SDNDefender	0.93	0.88	0.75	1	0.82	1

Table 10: False positive rate.

Attackers	1	2	3	4	5	6
SLICOTS	0.27	0.66	0.46	0.36	0.33	0.54
SRL	0.25	0.32	0.30	0.23	0.25	0.25
SDNDefender	0.13	0.07	0.10	0.12	0.09	0.11

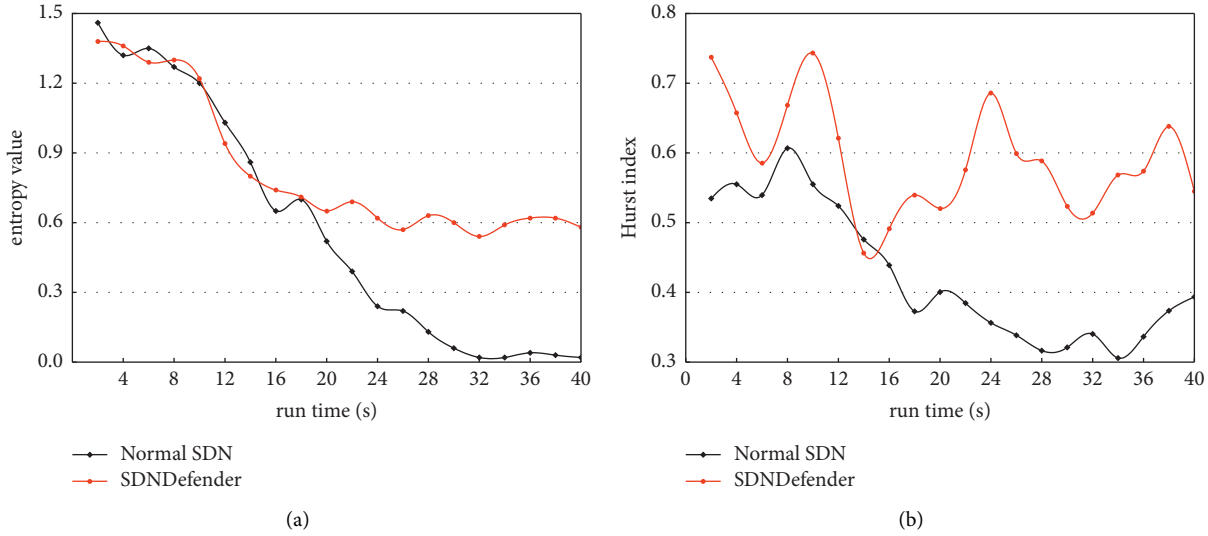

Figure 15: The variation of network traffic metrics. (a) The entropy based on destination IP. (b) The Hurst index.

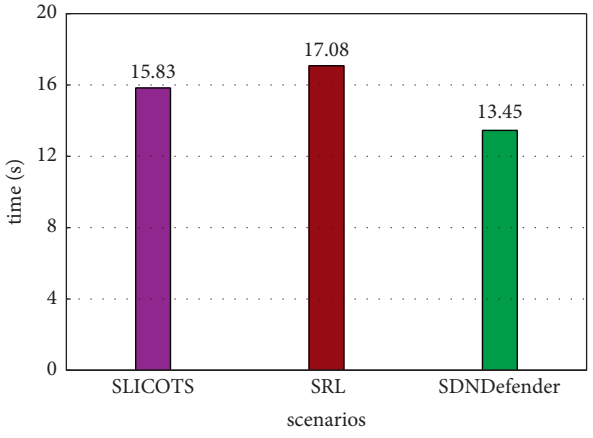

Figure 16: Comparison of the first attack detection time.

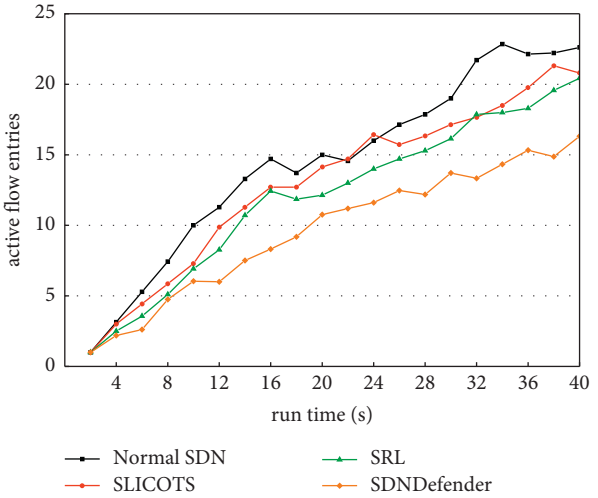

Figure 17: Comparison of active flow entries in switches.

Third, to simulate literature [15] we use random forest algorithm for training samples to go anomaly detection. This ML algorithm needs some explicit parameters, such as the number of a decision tree, split criterion, the max depth of a decision tree, etc. Besides these, the training sample size is also an important role in the classification process.

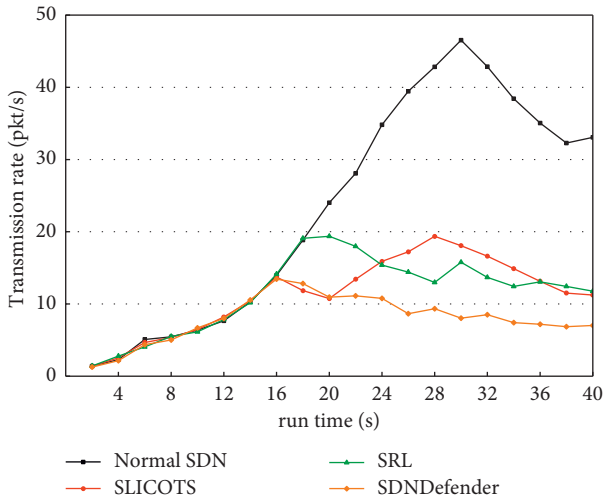

Figure 18: Comparison on SYN packets' transmission rate.

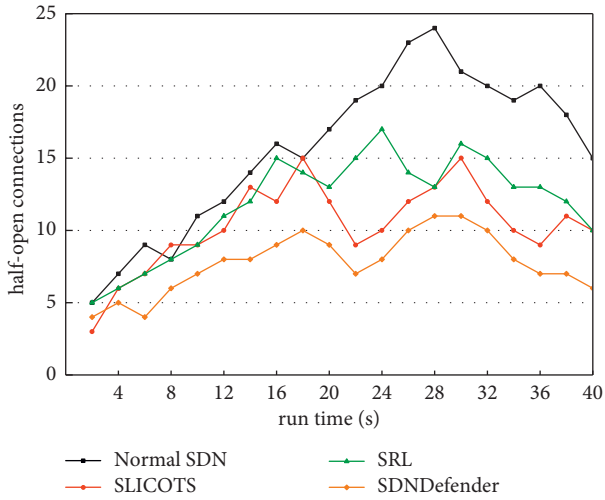

Figure 19: Comparison of TCP half-open connections.

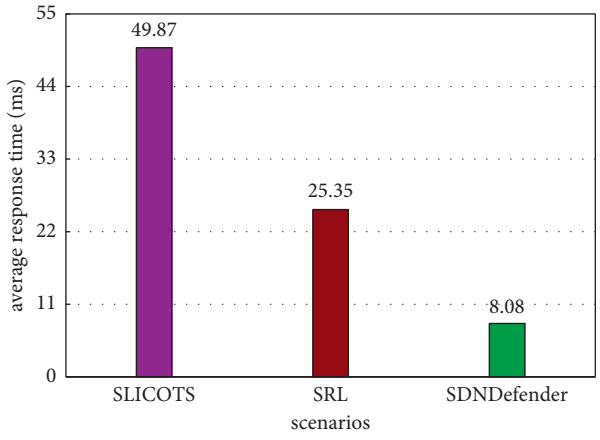

Figure 20: Comparison of HTTP response time.

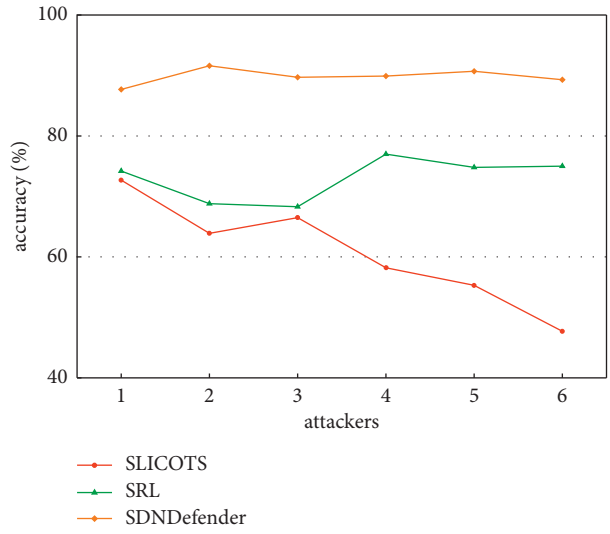

Figure 21: Comparison of detection accuracy of SYN flood attacks.

6. Conclusion

Despite the large number of traditional defense solutions that exist now, DDoS attacks still grow in frequency and severity. This requires an innovative network technique to address today's challenging security threats. With the superiorities of possessing a global view and centralized control mode, SDN offers a new angle to review how this problem is solved with SDN patterns. In this paper, we present SDNDefender, a SDN-based DDoS detection and defense mechanism against IP spoofing attack and TCP SYN flood attack. SDNDefender includes two core components: Anti-IP spoofing and Anti-TCP SYN flood. Anti-IP spoofing component adopts dynamic entries that match to validate a packet's source address and build an intrusion prevention rules library to defend those well-known attacks. Besides, the component employs a host-based anomaly detection algorithm to counter unknown attacks. Anti-TCP SYN flood component measures SYN packet's transmission rate and once suspicious flows are detected, the component calculates the current network entropy and self-similarity index. As these metrics exceed their thresholds, the component installs entries on switches to block flood traffic and uses a DiffServ-based mitigation method to reduce attack influences on the network.

To evaluate SDNDefender in the round, we implement two group experiments. Our experimental results prove that SDNDefender outperforms other algorithms from many metrics and poses a desirable performance. In future work, we will pay more attention to the functional extension of SDNDefender and enable it to defend against some new-style attacks. At the same time, we plan to use more complex network topology and realistic traffic for security experiments.

Data Availability

The experimental results (xlsx format) of this study have been deposited in the repository (https://pan.baidu.com/s/1s36uxFEj5evsdA71oTrPrA. fetch code: xc8g).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The article was partly funded by National Natural Science Foundation of China under Grant no. 61702048.

References

[1] T. Thapngam, S. Yu, W. Zhou, and S. K. Makki, "Distributed Denial of Service (DDoS) detection by traffic pattern analysis," *Peer-to-Peer Networking and Applications*, vol. 7, no. 4, pp. 346–358, 2012.

[2] A. Chonka, J. Singh, and W. Zhou, "Chaos theory based detection against network mimicking DDoS attacks," *IEEE Communications Letters*, vol. 13, no. 9, pp. 717–719, 2009.

[3] J. Cheng, X. Tang, X. Zhu, and J. Yin, "Distributed denial of service attack detection based on IP flow interaction," in *Proceedings of the International Conference on E-Business and E-Government (ICEE)*, IEEE, Shanghai, China, May 2011.

[4] R. F. Fouladi, C. E. Kayatas, and E. Anarim, "Frequency based DDoS attack detection approach using naive Bayes classification," in *Proceedings of the International Conference on Telecommunications and Signal Processing (TSP)*, IEEE, Vienna, Austria, June 2016.

[5] Y. Wang, J. Ma, L. Zhang, W. Ji, D. Lu, and X. Hei, "Dynamic game model of botnet DDoS attack and defense," *Security and Communication Networks*, vol. 9, no. 16, pp. 3127–3140, 2016.

[6] R. Bahman and F. Carol, "CoFence: a collaborative DDoS defence using network function virtualization," in *Proceedings of the International Conference on Network and Service Management (CNSM)*, IEEE, Montreal, Canada, October 2016.

[7] R. Sahay, W. Meng, D. A. S. Estay, C. D. Jensen, and M. B. Barfod, "CyberShip-IoT: a dynamic and adaptive SDN-based security policy enforcement framework for ships," *Future Generation Computer Systems*, vol. 100, pp. 736–750, 2019.

[8] T. Lukaseder, L. Maile, B. Erb, and F. Kargl, "SDN-assisted network-based mitigation of slow DDoS attacks," in *Proceedings of the International Conference on Security and Privacy in Communication Systems (SecureComm)*, Springer, Singapore, August 2018.

[9] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sFlow and adaptive polling sampling for deep learning based DDoS detection in SDN," *Future Generation Computer Systems*, vol. 111, pp. 763–779, 2020.

[10] P. Du and A. Nakao, "DDoS defense deployment with network egress and ingress filtering," in *Proceedings of the IEEE International Conference on Communications (ICC)*, IEEE, Cape Town, South Africa, May 2010.

[11] F. Hauser, M. Häberle, M. Schmidt, and M. Menth, "P4-IPsec: site-to-site and host-to-site VPN with IPsec in P4-based SDN," *IEEE Access*, vol. 8, pp. 139567–139586, 2020.

[12] J. Bi, B. Liu, J. Wu, and Y. Shen, "Preventing IP source address spoofing: a two-level, state machine-based method," *Tsinghua Science and Technology*, vol. 14, no. 4, pp. 413–422, 2009.

[13] G. Chen, G. Hu, Y. Jiang, and C. Zhang, "SAVSH: IP source address validation for SDN hybrid networks," in *Proceedings of the IEEE Symposium on Computers and Communication (ISCC)*, IEEE, Messina, Italy, June 2016.

[14] R. C. Meena, M. Nawal, and M. Bundele, "SIPAV-SDN: source Internet protocol address validation for software defined network," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 12, pp. 3386–3393, 2019.

[15] R. Zhao, S. Wei, and M. Ren, "Combating DDoS attack with dynamic detection of anomalous hosts in software defined network," in *Proceedings of the International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, IEEE, Mysore, India, September 2017.

[16] J. Kwon, D. Seo, M. Kwon, H. Lee, A. Perrig, and H. Kim, "An incrementally deployable anti-spoofing mechanism for software-defined networks," *Computer Communications*, vol. 64, pp. 1–20, 2015.

[17] Sreeramoju and A. Kumar, *Techniques for Improving SYN Cache Performance*, San Jose, CA, USA, 2017, https://www.freepatentsonline.com/9560173.html.

[18] V. T. Dang, T. T. Huong, N. H. Thanh, P. N. Nam, N. N. Thanh, and A. Marshall, "SDN-based SYN proxy-a solution to enhance performance of attack mitigation under TCP SYN flood," *The Computer journal*, vol. 62, no. 4, pp. 518–534, 2019.

[19] 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*, Berlin, Germany, November 2013.

[20] M. Ambrosin, M. Conti, F. D. Gaspari, R. Poovendran, and " LineSwitch, "Efficiently managing switch flow in Software-Defined Networking while effectively tackling DoS attacks," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, Singapore, April 2015.

[21] N. Tuan, P. Huang, N. Nghia, V. T. Nguyen, and V. P. Trung, "A robust TCP-SYN flood mitigation scheme using machine learning based on SDN," in *Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC)*, IEEE, Jeju Island, South Korea, October 2019.

[22] P. Kumar, M. Tripathi, A. Nehra, M. Conti, and C. Lal, "SAFETY: early detection and mitigation of TCP SYN flood utilizing entropy in SDN," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1545–1559, 2018.

[23] A. Basheer, A. Eslam, and A. Yazeed, "ISDSDN: mitigating SYN flood attacks in software defined networks," *Journal of Network and Systems Management*, vol. 28, no. 4, pp. 1366–1390, 2020.

[24] M. Reza, J. Reza, and C. Mauro, "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.

[25] U. Tushar and J. A. Kumar, "SRL: an TCP SYNFLOOD DDoS mitigation approach in software-defined networks," in *Proceedings of the International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, IEEE, Coimbatore, India, March 2018.

[26] S. Popic, M. Vuleta, P. Cvjetkovic, and B. M. Todorović, "Secure topology detection in software-defined networking with network configuration protocol and link layer discovery protocols," in *Proceedings of the International Symposium on Industrial Electronics and Applications (INDEL)*, Banja Luka, Bosnia and Herzegovina, November 2020.

[27] M. Jayakumar, N. Rekha, and B. Bharathi, "A comparative study on RIP and OSPF protocols," in *Proceedings of the International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, IEEE, Coimbatore, India, March 2015.

[28] X. Bai, G. Yang, and Z. Jiang, "Research and application of Floyd algorithm based on SDN network," in *Proceedings of the International Conference on Intelligent Computation Technology and Automation (ICICTA)*, IEEE, Hunan, China, October 2019.

[29] CIC-DDoS2019 Dataset, 2019, https://www.unb.ca/cic/datasets/ddos-2019.html.

[30] U. Tushar and J. A. Kumar, "A survey of quality of service (QoS) protocols and software-defined networks (SDN)," in *Proceedings of the 2018 Computing Conference*, London, UK, July 2018.