Hindawi
Security and Communication Networks
Volume 2021, Article ID 6688489, 23 pages
https://doi.org/10.1155/2021/6688489

Research Article

A Lightweight SDN Fingerprint Attack Defense Mechanism Based on Probabilistic Scrambling and Controller Dynamic Scheduling Strategies

Tao Wang[1,2] **and Hongchang Chen**[1,2]

[1]*PLA Strategic Support Force Information Engineering University, Zhengzhou, China*
[2]*National Digital Switching System Engineering and Technological Research Center, Zhengzhou, China*

Correspondence should be addressed to Tao Wang; wangtaogenuine@163.com

Received 9 October 2020; Revised 31 December 2020; Accepted 15 January 2021; Published 27 January 2021

Academic Editor: Jin Cao

Software-defined networking (SDN) decouples the control plane from the data plane, which increases network flexibility and programmability. However, the "three-layer two-interface" architecture of SDN introduces new security issues. Attackers can collect fingerprint information (such as network types, controller types, and critical flow rules) by analyzing round-trip time (RTT) distribution of test packets. In order to defend against the fingerprint attack with limited attack time, we first design a probabilistic scrambling strategy. This strategy not only interferes with the delay distribution of probe packets in attack flow but also reduces the negative impact on the performance of legal packets in normal flow. However, if fingerprint attackers have unlimited attack time, it is not enough to defend against the attack only by this strategy. Therefore, we further propose a controller dynamic scheduling strategy to change SDN fingerprint information actively. Because scheduling different types of controllers to work in different periods will generate costs, the scheduling strategy is also responsible for determining the optimal switching time point to balance security benefits and costs. At last, we implement the defense mechanism on different types of controllers and verify its effectiveness in experimental scenarios. The experimental results show that the mechanism can effectively hide the SDN fingerprint information while reducing the negative impact on network performance.

1. Introduction

In recent years, SDN [1] has received widespread attention as new network architecture and deployed in large-scale data center scenarios, such as Google B4, Microsoft Azure, and Amazon EC2. Different from the traditional network architecture, SDN separates the control plane from the data plane, which significantly improves network flexibility and programmability. The function of SDN control plane is mainly realized by a logically centralized controller. The controller interacts with the data plane through a secure two-way channel to maintain state information and formulate network policies. The SDN data plane is responsible for realizing the corresponding network function according to flow rules generated by the controller. Let us take the OpenFlow protocol as an example. The specific packet processing flow is shown in Figure 1. When a packet arrives at the switch, the switch first parses the header space and checks whether the corresponding flow rule exists in flow table. If there is a matching flow rule, the switch will forward the packet (Path 1). Otherwise, the controller generates a flow rule to instruct the switch to forward the packet (Path 2).

By analyzing the abovementioned OpenFlow packet processing flow, it can be seen that the decoupling structure of the control plane and the data plane provides the possibility for an attacker to launch SDN fingerprint attack. When there is a flow rule matching the packet in the flow table of the switch, the packet is directly forwarded by the switch at a high rate. However, when there

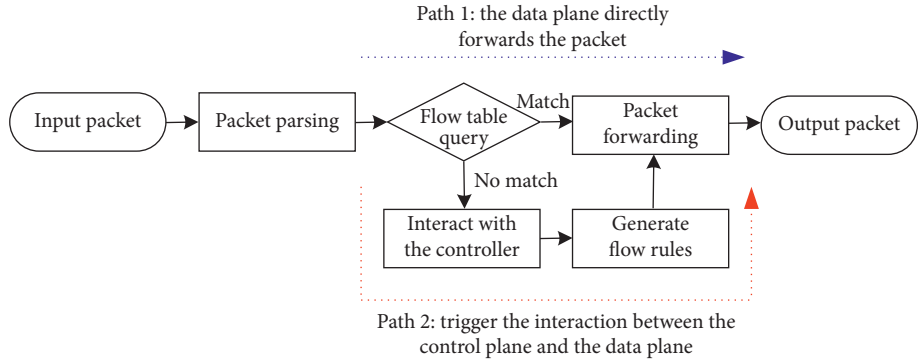


Figure 1: A typical OpenFlow packet processing flow.

is no flow rule matching the packet in the flow table of the switch (or the controller needs to perform a more advanced policy on the packet), the switch triggers the table-miss event to notify the controller to install the corresponding flow rule. Then, the switch forwards the packet based on the flow rule. Since the packet forwarding rate of the switch is several orders of magnitude faster than the flow rule generating rate of the controller, there is a significant difference in the forwarding delay between the two packet forwarding processes (Path 1 and Path 2).

This delay difference can be used as an important indicator for attackers to launch fingerprint attack. By analyzing the delay distribution of the specially constructed probe packets, the attacker can know whether the packet is only processed in the data plane or triggers the interaction between the data plane and the control plane. Then, the critical network parameters such as network types, controller types, and critical flow rules can be identified. The attacker can use the fingerprint information to launch more threatening attack [2, 3]. For example, (1) when the attacker successfully identifies that the network type is SDN, the attacker can efficiently launch DDoS attack to overload the SDN controller. (2) When the attacker successfully identifies the controller type, the attacker can easily launch penetration attack based on known controller vulnerabilities. In this way, the attacker can take control of the controller and take over the entire network. (3) When the attacker successfully detects the critical flow rules in the data plane, the attacker can better grasp the network graph (including the node and link information). This helps the attacker to kill the key nodes or paths in the network accurately. Therefore, the research on SDN fingerprint attack and its defense is of great significance.

At present, the research on SDN fingerprint attack is still in a preliminary stage. Related attack and defense technologies are still immature (the specific analysis is detailed in Section 2). What is more, the existing researches mainly focus on how to use SDN fingerprint attack to mine more effective information, but there is almost no defense research work on SDN fingerprint attack. In order to solve this problem, this paper proposes an SDN fingerprint attack defense mechanism. The mechanism is based on probabilistic scrambling strategy and controller dynamic scheduling strategy in the dual-time dimension, which can make the fingerprint identification parameters deviate from the original distribution. Within a single-round defense time window, the SDN controller can add random perturbations (according to flow attributes) to confuse the attacker. By randomly scrambling different packets, the controller can interfere with fingerprint information and optimize network service quality. Within a multiround defense time window, the controller dynamic scheduling strategy can actively change the system's fingerprint information. In addition, it can also reduce the overhead of hiding SDN fingerprint information by selecting the optimal switching time point. In summary, the main research contributions of this paper are as follows:

(i) Construct a full-factor SDN fingerprint attack chain and discuss the fingerprint information extraction and utilization process in detail.

(ii) Design a collaborative obfuscation strategy in the dual-time dimension to effectively improve the hiding degree of SDN fingerprint information.

(iii) Propose a gradient probabilistic scrambling strategy in the single-round defense time window, which can reduce the negative impact on the performance of normal packet while preventing fingerprint attacks under limited time constraint.

(iv) Propose a controller dynamic scheduling strategy in the multiround defense time window, which can balance security benefits and scheduling overhead while preventing fingerprint attacks without attack time constraint.

(v) Implement the defense mechanism on different types of controllers and verify its effectiveness in actual experimental scenarios.

The remainder of this paper is organized as follows. In Section 2, we outline the related works. In Section 3, we construct the SDN fingerprint attack model. In Section 4, we design a lightweight SDN fingerprint attack defense mechanism based on probabilistic scrambling strategy and controller dynamic scheduling strategy. The implementation and evaluation of the defense mechanism are in Section 5. A conclusion is drawn in Section 6.

2. Related Works

Fingerprint attacks in traditional networks [4, 5] mainly identify information such as the operating system type or version number of the remote host. Then, the attacker uses the corresponding system vulnerabilities to launch more threatening attacks. At present, the research on traditional fingerprint attack is relatively mature. Many targeted anti-fingerprint schemes have been proposed successively [6–8]. Even some traditional defense technologies such as firewall or intrusion detection system (IDS) can be applied to traditional fingerprint attack scenarios. With the development of SDN technology, SDN fingerprint attack has gradually attracted widespread attention from researchers. Due to the decoupling architecture of SDN, the fingerprint attack in SDN is completely different from that in traditional networks. Their defense ideas are also quite different. In addition, traditional defense methods cannot be directly transplanted and reused in SDN. Therefore, it is urgent to propose an effective defense solution for SDN fingerprint attack.

Shin et al. developed the SDN Scanner tool [9] to start SDN fingerprint identification. This tool can continuously send packets to the SDN network after generating packet header fields and record the response time of these packets. Because the data plane requires additional flow rule installation time when the corresponding flow rule is not matched, the response time when there is no matching flow rule (T_1) is different from the response time when there is a corresponding flow rule (T_2). The SDN Scanner collects packet response times and uses statistical tests to compare their distribution. In this way, the attacker can clearly distinguish T_1 and T_2 and use this as an indicator to determine whether the network is an SDN network. Shin et al. qualitatively analyzed the possibility of SDN fingerprint attack, but they did not quantify specific indicators and test the attack process in the simulation environment. In addition, because there are many factors affecting the response time of packets, it is difficult to collect accurate T_1 and T_2 values in an actual WAN. Therefore, this method may not be efficient in a wide-area network. Reference [10] uses the packet-pair dispersion to identify whether a given packet triggers the interaction between the controller and the switch. This indicator reduces the negative impact of network jitter on the accuracy of fingerprint recognition. In reference [10], a possible defense mechanism was also proposed. It makes all received packets that need to be forwarded uniformly delayed. As far as we know, this is the only fingerprint attack defense method available. It effectively makes the fingerprint identification parameters deviate from the original distribution. However, uniformly delaying all packets may significantly reduce network performance. The abovementioned research works mainly determine whether the network type is SDN. The fingerprint information obtained is relatively rough.

With the development of SDN fingerprint attack, researchers can extract more fine-grained SDN fingerprint information based on more indicators. Azzouni et al. [11] found that different types of controllers have different processing speeds due to different programming languages, function libraries, and framework structures. Therefore, the controller type can be identified by recording the average processing time and comparing it to the constructed controller response time database. Sonchack et al. [12] used specially constructed probe packets and test packets to identify key flow rules in the flow table. Zhang et al. [13] conducted research on how to identify the flow matching domain information of SDN switches. Leng et al. [14] identified the flow table capacity and flow table usage of SDN switches. They believe that when the flow table is full, additional interaction between the controller and the switch is required to remove some existing flow rules to make room for new flow rules, which may lead to network performance degradation. After the attacker has grasped the capacity and usage of the flow table, he can accurately estimate how many packets he needs to generate per second to overload the flow table and the time required to overload it. The attacker can also correctly configure the attack tool based on the abovementioned information. Bilal and Nadeem [15] focused on a specific data plane attack referred to as Flow Table Entry Attack (FTEA) to infer the flow replacement policy in an SDN-based environment. Hou et al. [16] presented a fine-grained fingerprinting method that it can learn the match fields of flow rules by distinguishing the transmission delays of different packets. Cao et al. [17] designed a deep learning-based method to fingerprint SDN applications from mixed control traffic. However, this method requires the attacker to be able to eavesdrop control traffic between the controller and a switch, which limits its scope of application. By summarizing the abovementioned related works, we can find that many types of SDN fingerprint attacks can extract fine-grained fingerprint information. Therefore, we cannot ignore the threats they caused.

At present, most of the SDN security works still focus on some relatively mature attacks. For example, Shin et al. designed the Avant-Guard system to defend against DoS attack of the SDN control plane [18]. Hong et al. [19] proposed a defense scheme against network topology poisoning attack to prevent attackers from hijacking network connections. Shin et al. proposed the SDN security framework FRESCO [20], which can implement the modular operation of OpenFlow security components and help enhance SDN security. Dhawan et al. developed the SPHINX framework [21], which can detect malicious infiltration attack based on network flow graphs. Porras et al. [22] extended the control plane to detect and arbitrate flow rule conflicts from multiple applications. Considering that SDN fingerprint attack is quite different from other attacks against SDN, these solutions cannot prevent the SDN fingerprint attack studied in this paper. Therefore, it is of great significance to propose a defense solution specifically for SDN fingerprint attack.

3. Fingerprint Attack Model

3.1. Motivation. Considering that the form of SDN fingerprint attack is very abstract and the amount of research related to SDN fingerprint attack is also limited, we need to

introduce the fingerprint attack model in detail in this section to help readers establish an intuitive understanding of the attack process. However, as the existing researches on SDN fingerprint attack are messy and fragmented, there is no one paper that gives a comprehensive introduction to the SDN fingerprint attack model. Therefore, it is very meaningful to refine the attack process and highlight the key points. Specifically, in this section, we divide the types of fingerprint information into network types, controller types, and critical flow rules and classify the existing researches according to the types of fingerprint information. Then, we construct the full-factor SDN fingerprint attack chain based on the classification results of the existing researches. It is worth noting that although the specific technologies in the SDN fingerprint attack chain are based on the existing researches [9, 11, 12], the concept of the fingerprint attack chain is first proposed by us in this paper, so this is also one of the contributions of our paper.

3.2. Hypothesis. SDN fingerprint attackers mainly use the delay distribution of the specially constructed probe packets to infer fingerprint information. However, the delay distribution may also be affected by some irresistible factors, such as real-time network conditions and hardware performance. Although these factors may not have a decisive impact on the results of fingerprint identification, they will more or less reduce the accuracy of fingerprint identification. Therefore, in order to better study the impact of the decoupling structure of the control plane and the data plane on fingerprint attack results (excluding irrelevant factors), we make the following assumptions: (1) the real-time network conditions of the probe packets sent by the attacker are approximately the same; (2) the bottleneck of the control plane data packet processing time mainly depends on the controller software (rather than the hardware); (3) the forwarding performance of all devices in the data plane is approximately the same.

Based on the abovementioned analysis, we introduce the specific content of the SDN fingerprint attack chain as follows.

3.3. Network Type Fingerprint Information. Accurately identifying the network type is the first step in the SDN fingerprint attack chain. If the attacker knows that the network type is SDN, he can use the SDN unique characteristics to identify the other fingerprint information. As shown in Figure 2, in order to construct a fingerprint information recognition model for the network type, the routing link sequence between the attacker a and the server s in the network is defined as $R_{as} = < P_{as}^1, \ldots, P_{as}^n >$ (P_{as}^i represents the i^{th} node in the routing link sequence R_{as}). Similarly, its reverse path is $R_{sa} = < P_{sa}^1, \ldots, P_{sa}^m >$. Let τ_{P^i} be the transmission delay of the packet at the i^{th} hop and $d_{P^i}^j$ denote the extra delay experienced by the packet j when it is transmitted on the path P^i (this delay is usually caused by the background traffic at the i^{th} hop, resulting in additional queues for the packet j). According to the abovementioned definition, the round-trip times (RTT) of the packet i can be obtained as follows:

$$\text{RTT}_i = \sum_{j=1}^{n}\left(\tau_{P_{as}^j} + d_{P_{as}^j}^i\right) + \sum_{j=1}^{m}\left(\tau_{P_{sa}^j} + d_{P_{sa}^j}^i\right) + \max_{\forall k}\delta_k^i, \quad (1)$$

where δ_k^i represents the delay caused by possible communication between the SDN controller and the OpenFlow switch k (the switch is on the routing path between the attacker a and the server s). Since the SDN controller installs bidirectional flow rules on all switches at the same time, we only consider the maximum flow rule installation delay in this equation. If there is no communication between switch k and the controller (e.g., there is a matching flow rule), then $\delta_k^i = 0$. Considering that the RTT of a packet likely depends on other factors such as the geographical location of the host and real-time network conditions, we measure the RTT difference between the two probe packets sent by the attacker to eliminate these extraneous factors. The RTT difference is shown as follows:

$$\Delta\text{RTT} = \max_{\forall k}\delta_k^1 - \max_{\forall k}\delta_k^2 + \sum_{j=1}^{n} d_{P_{as}^j}^i + \sum_{j=1}^{m} d_{P_{sa}^j}^i. \quad (2)$$

As can be seen from the above formula, this indicator is mainly related to the network jitter and the overhead of the interaction between the SDN controller and the switch. It is also worth noting that this indicator does not depend on the attacker's location. Under normal network conditions (there are no extreme cases such as DoS attack), the impact of network jitter can be neglected compared to the interaction overhead [23], so formula (2) can be simplified as follows:

$$\Delta\text{RTT} = \max_{\forall k}\delta_k^1 - \max_{\forall k}\delta_k^2. \quad (3)$$

If neither packet causes any rule installation or the network type is a traditional network, then $\Delta\text{RTT} \approx 0$. If any one of the packets triggers the rule installation ($\max_{\forall k}\delta_k^1 \gg 0$ or $\max_{\forall k}\delta_k^2 \gg 0$), it proves that the network type is SDN ($|\Delta\text{RTT}| \gg 0$). It can be seen from the abovementioned analysis that the indicator ΔRTT has significant differences in different networks. Let us take the SDN as an example. As shown in Figure 2, after obtaining the probability density function (PDF) of all ΔRTT values, the attacker can conclude that the ΔRTT of packets that trigger the controller to interact with the switch is much larger than zero. The ΔRTT distribution of packets that do not trigger an interaction can be fitted to a normal distribution with an average of zero. In general, a t-test on two types of samples [24] reveals that they are significantly different at the 1% level. Therefore, the target network type information can be effectively identified based on the ΔRTT distribution.

3.4. SDN Controller Type Fingerprint Information. After the attacker judges that the target network type is SDN, he can further launch efficient and accurate attacks on the target network based on SDN characteristics. SDN controller is the "brain" of the entire network. When the attacker successfully identifies the controller type, he can easily obtain the actual

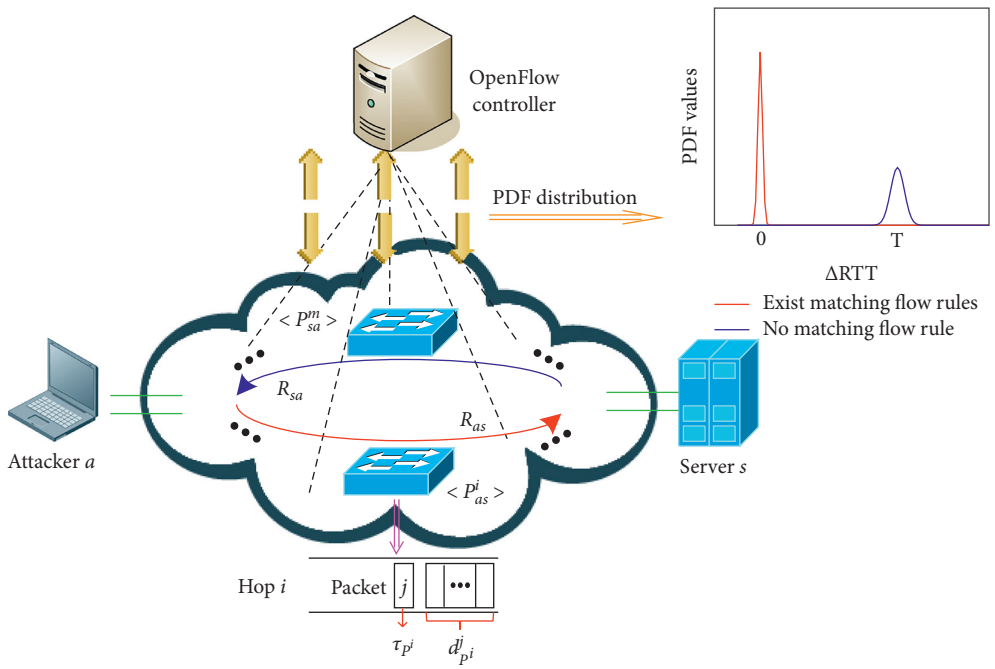


Figure 2: Schematic diagram of the fingerprint attack for network type.

control right of the controller based on the known controller vulnerabilities. This will have a devastating impact on the target network. Next, we will systematically describe the fingerprint attack for controller types in this subsection.

In order to accurately identify the SDN controller type, the attacker first needs to measure the timeout information of flow rule. The timeout of flow rule mainly includes an idle-timeout and a hard-timeout. They indicate when the switch deletes the flow rule without packet matching and when the flow rule is forcibly removed. The attacker can identify the timeout value of flow rule by adjusting the time interval between the probe packets. The specific process is shown in Figure 3.

As shown in Figure 3(a), in order to detect the idle-timeout value, the attacker first sends a packet that can trigger the switch to interact with the controller (its RTT value is T_2) and then sends the same packet again at a short interval ΔT_1. Because the matching flow rule already exists, the RTT value of the second packet is T_1. Subsequently, the attacker gradually increases the packet interval ($\Delta T_{i-1} < \Delta T_i$) using "binary search" or other algorithms until the RTT value of the n^{th} packet is observed to change from T_1 to T_2 (the flow rule is deleted due to the idle-timeout). At this time, ΔT_n is the idle-timeout value. As shown in Figure 3(b), in order to detect the hard-timeout value, the attacker also first sends a packet that can trigger the switch to interact with the controller (its RTT value is T_2) and then sends the same packet again at a time interval ΔT which is far less than the idle-timeout value. At this time, because a matching flow rule already exists, the RTT value of the second packet is T_1. Subsequently, the attacker continues to send packets at this interval until he observes that the RTT value of the n^{th} packet changes from T_1 to T_2 (the flow rule is deleted due to the hard-timeout). At this time, $N\Delta T$ is the hard-timeout value.

When the attacker grasps the effective time of the flow rule, he can launch the fingerprint attack for controller types in a round of the effective time to avoid the error caused by the flow rule timeout.

Because different SDN controllers use different programming languages, function libraries, and frameworks, there are certain differences in the execution speed of different controllers. This feature gives attackers chances to identify the SDN controller type. By measuring the response time of the target controller and comparing it to a precreated processing time database for different controllers, the attacker can draw conclusions. Attackers can use the ping tool to create the processing time database. It is worth noting that the interval between each two *ping* packets should be greater than the idle-timeout value. In the abovementioned steps, each *ping* will cause the switch to send a Packet-In message to the controller (the controller extracts the Packet-In message field value and installs the corresponding flow rule into the switch), and then the attacker calculates the average response time of these ping packets T_{pavg}. In the same environment, the attacker again measures the average RTT value RTT_{avg} of n packets in the presence of flow rules. The processing time of the current controller is $T_{\text{pavg}} - \text{RTT}_{\text{avg}}$. The attacker can repeat the abovementioned process for all types of controllers to get the controller processing time database (controller; processing time (T_p)). Finally, the controller type can be inferred by comparing the difference between the measured delay and the database delay Compare ($\text{RTT}' - \text{RTT}_{\text{avg}}$, processing time ($T_p$)), as shown in Algorithm 1. It is also worth noting that the result of the method is only a probabilistic result, which is not absolutely correct. This is because real-time network conditions, applications, and even hardware may affect the accuracy of controller type recognition. Therefore, this fingerprint attack

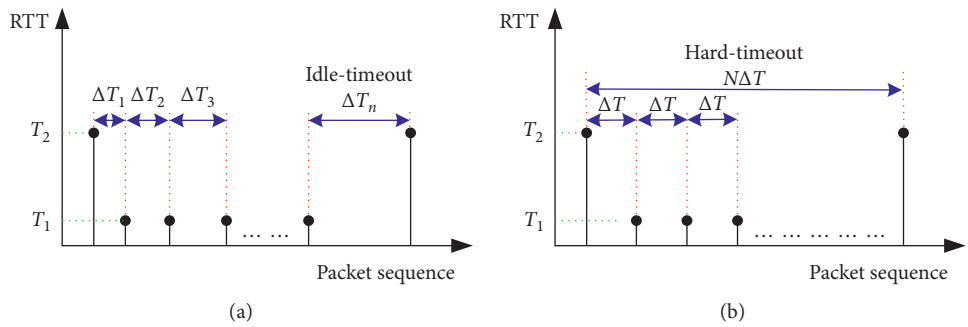


Figure 3: Flow rule timeout detection process. (a) Idle-timeout. (b) Hard-timeout.

(1) **Calculate** *idle-timeout* and RTT_{avg}
(2) **for** $i = 1$ to m **do**
(3) wait *period* > *idle-timeout* seconds
(4) send a *ping* and save *ping* time
(5) **end for**
(6) **Calculate** the average of saved ping-time values avg idle-timeout RTT′
(7) **Compare** $RTT' - RTT_{avg}$ to the processing time entries

Algorithm 1: Identify controller type fingerprint information.

method is not perfect and requires further research by related researchers who focus on how to use SDN fingerprint attack to mine more effective information.

3.5. Critical Flow Rule Fingerprint Information. SDN flow rules can specifically represent network policies such as forwarding and security. If the attacker could detect the critical flow rules (successfully launch the fingerprint attack for critical flow rules), the attacker may better understand the packet forwarding logic and accurately attack key nodes or paths in the network.

To identify critical flow rules, the attacker sends a time probe flow and a test flow to the target network at the same time. A specially constructed time probe flow can trigger the interaction between the controller and the switch, and its round-trip time (RTT) depends on the controller. The test packet is also a specially constructed packet. The attacker continuously adjusts the header field value of the test packet based on the attributes of the target flow rule. Then, the attacker observes the round-trip time of the time probe flow until the target flow rule can be detected through the distribution law. The test flow syntax template is shown in Figure 4.

If the test packet causes the RTT of the time probe flow to increase, the attacker can infer that the control plane is processing the test packet (i.e., the switch does not match the flow rule corresponding to the test flow). Otherwise, the attacker can infer that the switch matches the corresponding flow rule. To describe the specific process of the critical flow rule fingerprint attack, we assume that there are 4 hosts h1–h4 in the target network, and the MAC addresses are 00 : 00 : 00 : 00 : 00 : 01 to 00 : 00 : 00 : 00 : 00 : 04. An example of the topology and the flow rule table is shown in Figure 5.

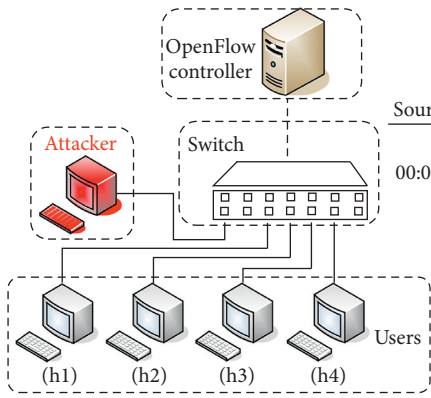
Test packet streams:
test_packet_stream::= <*template, size, transmit_rate*>
Stream templates:
template::= <*header_field=field_value, …*>
Header fields:
header_field::= mac_source | mac_dest | ip_source | ip_dest | …
Header values:
field_value::= C (*i.e.*, each packet in the stream will have the same constant value C for this field)
| *(*i.e.*, each packet in the stream has a random value for the header)
| $\vec{C}$ (*i.e.*, the header field value for the ith packet in the stream will be $\vec{C}[i]$)

Figure 4: Test flow syntax template.

Under the abovementioned conditions, the attacker constructs test packets with different destination MAC addresses (from h1 to h2, h3, and h4) based on the test packet syntax template. Specific examples are as follows:

<*mac_source* = 00:. . .:01, *mac_dest* = 00:. . .:02>;
<*mac_source* = 00:. . .:01, *mac_dest* = 00:. . .:03>;
<*mac_source* = 00:. . .:01, *mac_dest* = 00:. . .:04>;

The attacker records the RTT of the corresponding time probe flow and calculates its statistical distribution (example results are shown in Figure 6). The RTT of the test flow with the destination MAC address h2 is not



Source MAC	Destination MAC	Action	Priority
*	00:00: … :02	Output on port x	High
00:00: … :03	00:00: … :04	Output on port y	High
*	ff:ff: … :ff	Flood	Medium
*	*	Send to controller	Low

Figure 5: Example of the topology and the flow rule table based on MAC address.

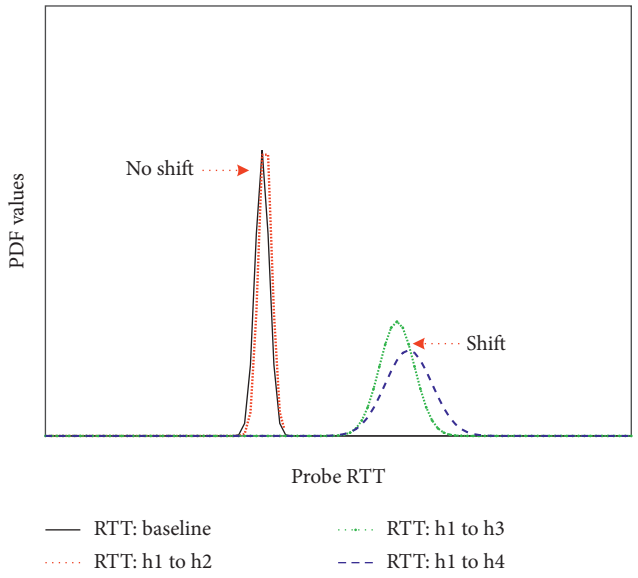


Figure 6: RTT distribution of the time probe flow.

significantly offset from the reference RTT distribution, which indicates that the switch has a matching flow rule and does not forward the test packet to the controller. However, the RTTs of the test flows with the destination MAC addressing h3 and h4 have significant distribution offsets, which indicates that the test packets do not match the flow rules and require further processing by the control plane. Based on the abovementioned analysis, it can be inferred that there is a flow rule (from h1 to h2) in the switch flow table. By continuously repeating the abovementioned process, the attacker can completely obtain the critical flow rules.

4. Fingerprint Attack Defense Mechanism

SDN fingerprint attackers can obtain fingerprint information such as network types, controller types, and critical flow rules based on the special delay attributes introduced by the SDN architecture. In order to solve this problem, we propose an SDN fingerprint attack defense mechanism based on probabilistic scrambling and controller dynamic scheduling strategies. This mechanism makes the fingerprint identification parameters deviate from the regular distribution in the dual-time dimension. Below, we will elaborate on the mechanism.

4.1. Probabilistic Scrambling Strategy

4.1.1. Motivation. From the third section, we can see that the delay difference can be used as an important indicator for attackers to launch fingerprint attacks. By analyzing the delay distribution of the specially constructed probe packets, the attacker can identify the key network parameter information (e.g., the network type, the controller type, and the key flow rule). As far as we know, there is only one defense method [10] that can solve this problem. The defense method makes all received packets that need to be forwarded uniformly delayed, which effectively makes the key network parameters deviate from the original distribution. However, standardizing each packet delay to the maximum interaction delay may inevitably lead to performance degradation of many normal packets. Therefore, how to reduce the negative impact on the performance of the normal packet is the motivation of this subsection.

4.1.2. Theoretical Basis. Because fingerprint attackers rely on the delay distribution of packets to identify the key fingerprint information, changing the delay distribution is a good way to solve SDN fingerprint attacks. If the delay distribution changes, the information inferred by the attacker will also be inaccurate. According to this principle, we can defend against this attack by changing the delay distribution. As for how to specifically change the delay distribution, we think that scrambling the packet delay is a direct way to change its distribution. However, different from the method of standardizing each packet delay to the maximum interaction delay (which seriously damages network performance), we need to selectively determine which packets to scramble (i.e., probabilistic scrambling strategy) based on the characteristics of the attack. In this way, our method can reduce the negative impact on the performance of the normal packet compared to the reference method. When an attacker launches an attack, the attacker generally compares the delay difference between the two

probe packets. Thus, theoretically, we only need to scramble the second probe packet to eliminate the delay difference. But the abovementioned scenario is too ideal. The defense system may face a variety of special attack cases (e.g., Case 1–3 in Section 4.1). In this way, only scrambling the second packet of each new flow cannot completely hide the fingerprint information, and it is necessary to scramble the subsequent packets with a certain probability. In the meanwhile, considering that the probability of occurrence of special cases is lower than that of the normal case, the demand for packet delay scrambling also decreases as the number of packets increases. Above all, the probabilistic scrambling strategy based on the abovementioned principles can effectively hide the fingerprint information and improve the quality of network services.

4.1.3. Detailed Design. As shown in Figure 7, the probabilistic scrambling strategy is mainly implemented by four modules: a monitor, a hash table, a policy generation module, and a probability scrambling execution proxy. The monitor is responsible for listening to flow events, collecting data plane state information, and storing this information into a hash table based on SDN programmability. The hash table extracts the source MAC address, destination MAC address, source IP address, and destination IP address of the flow as an index. If there is no matching index in the table, it creates a new entry and initializes its list of values (the number of packets in the flow; the number of advanced tags) to 0. If a related index already exists in the table, it updates the corresponding value list. The policy generation module is the core module of the probabilistic scrambling strategy. It mainly includes a probability decision component and a random delay disturbance component. The probability decision component determines the delay probability of specific packets in the flow, that is, which packets in the flow are to be delayed $Delay(Packet_i)$. The random delay disturbance component is responsible for determining the disturbance value of the packets that need to be delayed $Time(Packet_i)$. Based on the abovementioned operations, the policy generation module sends the $<Delay(Packet_i), Time(Packet_i)>$ policy combination, to the probability scramble execution proxy. The probability scrambling execution proxy is responsible for transforming the scrambling strategy into instructions that can be executed on the data plane. Specifically, the module marks different packets based on the probabilistic scrambling strategy. Then, to confuse the delay distribution, the module implements different delay operations on different packets by defining new action buckets selection logic for the group table. The specific process of this strategy is shown in Algorithm 2.

When a packet is received, the header information of the packet needs to be extracted and mapped into an index value through a hash algorithm. The index value is an important identifier to distinguish different flows and packets (lines 1–3). If there is no entry in the hash table that contains the index value, it creates a new entry with the index value as the key and initializes the packet count value and delay label count value in the value list to 0 (lines 4–6). If there is an entry in the hash table containing the index value, it updates the corresponding value list (lines 14-15). After performing the above mentioned steps, we need to design a probability decision model based on the characteristics of fingerprint attack. The probability decision model determines the packets that need to be disturbed and the amount of disturbance. Different from the method of standardizing each packet delay to the maximum interaction delay (which seriously damages network performance), we need to selectively determine which packets to scramble (i.e., probabilistic scrambling strategy) based on the characteristics of the attack. More specifically, considering SDN fingerprint attackers usually send a small number of probe packets in a forged flow to improve the efficiency of the attack, we can perform interference on the initial number of packets in the flow with a high probability and perform interference on the subsequent packets with an elastic gradient probability. Therefore, this paper designs a probability decision model that the interference probability changes with the number of packets in the flow. In this way, the probabilistic scrambling strategy can effectively hide the fingerprint information and reduce the negative impact on the performance of the normal packet compared to the reference method. The model is shown in the following formula:

$$P_c = \textbf{genProbability}(c) = \theta \cdot \left(\alpha \cdot \beta^{\gamma \cdot c} \cdot \cos \gamma c + \delta^c + \varepsilon\right), \tag{4}$$

where P_c is the scrambling probability value corresponding to the $packet_c$ in the flow, c is the sequence number of the packet, and $\theta, \alpha, \beta, \gamma, \delta, \varepsilon$ are the nonnegative coefficients for adjusting the gradient probability curve. Figure 8 shows the change of the probability value with the packet count value when $\theta = 0.78, \alpha = 0.2, \beta = 0.93, \gamma = 0.4, \delta = 0.91,$ and $\varepsilon = 0.1$. We interfere with the delay according to this probability decision model. On the one hand, it can specifically interfere with the packets in the attack flow (the interference probability of the initial packet is high) to obfuscate the delay distribution; on the other hand, it can also reduce the negative impact on the performance of packets in the normal flow (compared to the strategy of delaying all packets, the affected packets in this strategy are mainly concentrated on the initial packets in the flow).

In the case that there is no packet index in the hash table (lines 4–13) or in the case that there is a packet index in the hash table (lines 14–25), we determine the packets that need to be delayed based on the probability decision model and set the "need scrambling" tag (lines 8 and 18). After the random delay disturbance component receives the tag added by the probability decision component, it determines the random delay value of this packet and sends a probabilistic scrambling strategy to the probability scrambling execution proxy (lines 7–11 and 16–22). In order to avoid introducing the extra interaction overhead in the process of determining whether a packet needs to be delayed, this paper adopts an advanced decision mechanism. When a packet in a flow is received, a delay interference policy is formulated in advance for m subsequent packets of the flow. When the first packet in the flow is received, and the hash table entry is initialized,

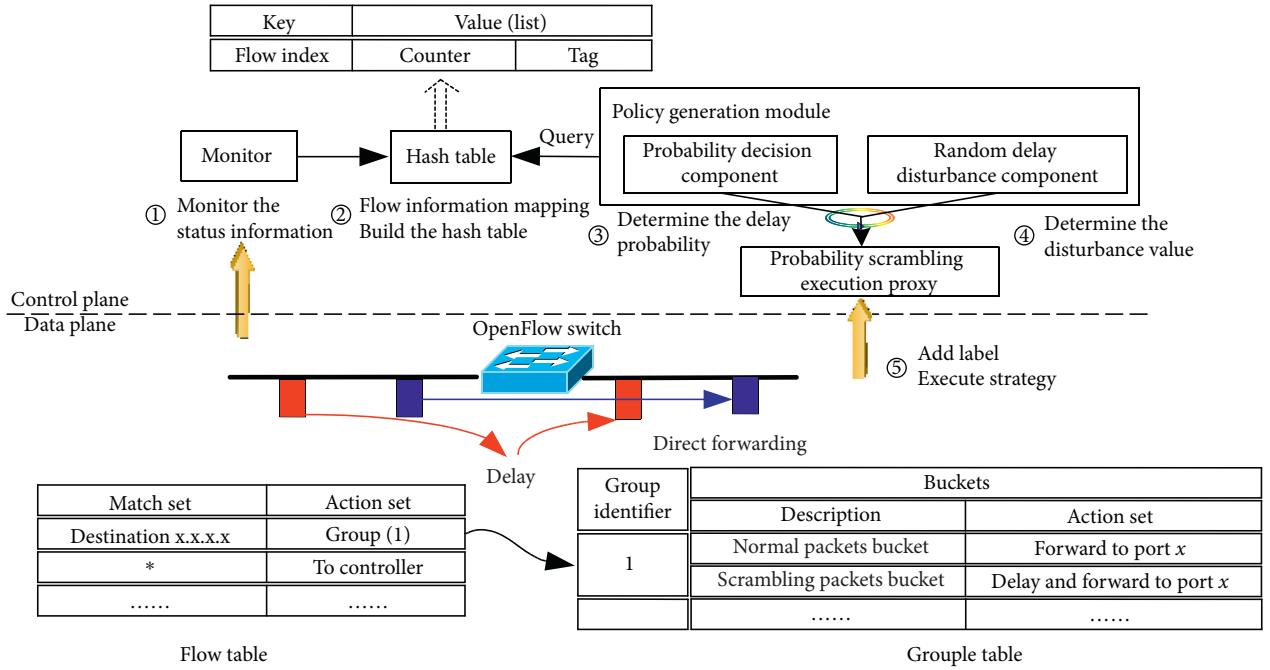


Figure 7: Workflow of the probabilistic scrambling strategy.

Input: hash table H; average RTT rtt
Output: scrambling packet with delay time dt ($packet_i$)

```
(1)  while TRUE do
(2)    packet_i ← receive a packet
(3)    index = hash(extractHeader(packet_i))
(4)    if the index of packet_i is not in the hash table H, then
(5)      H. add(index)
(6)      H(index).Counter ← 0 and H(index).Tag ← 0
(7)      for NumTag = 0 to m do
(8)        setDelayTag(Flow(index).Packet_NumTag, genProbability(NumTag)
(9)        if the label of Flow(index).Packet_NumTag is delayed, then
(10)         delay(Flow(index).Packet_NumTag, random(0.5, 1) * rtt) to proxy
(11)       end if
(12)     end for
(13)     H(index).Tag ← NumTag
(14)   else
(15)     H(index).Counter ← H(index).Counter + 1
(16)     if H(packet_i).Counter = H(index).Tag, then
(17)       for NumTag = H(index).Tag to H(index).Tag + m do
(18)       setDelayTag(Flow(index).Packet_NumTag, genProbanility(NumTag))
(19)       if the label of Flow(index).Packet_NumTag is delayed, then
(20)         delay(Flow(index).Packet_NumTag, random(0.5, 1) * rtt) ro proxy
(21)       end if
(22)     end for
(23)     H(index).Tag ← NumTag
(24)     end if
(25)   end if
(26) end while
```

Algorithm 2: Probabilistic scrambling strategy.

the probability decision component first sets the elements counter (indicating the packet count value in the flow) and tag (indicating the number of packets in the flow that are set tag in advance) in the value list to 0. Then, it cyclically sets the tags of the subsequent m packets (lines 7-8) and updates the value Tag to m (line 13). After receiving the subsequent

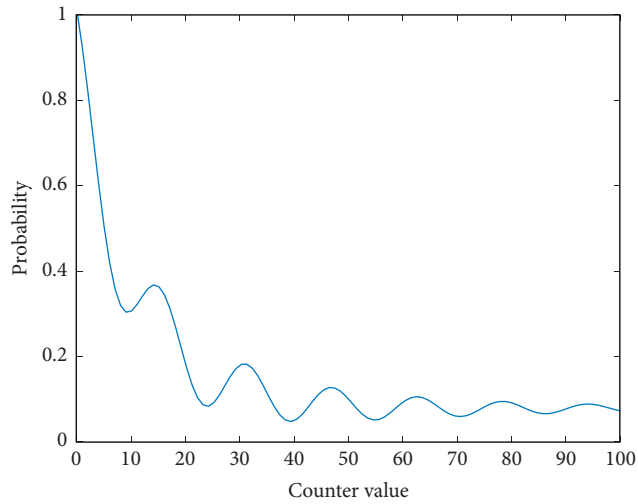


Figure 8: Gradient probabilistic scrambling model.

packets of the flow, the probability decision component updates the counter value and compares the value with the tag value. Once the two values are detected to be equal, a new round of packet advance decision is made (lines 16–18) and the tag value is updated (line 23). The advanced decision mechanism effectively avoids the controller to make fine-grained interference decisions for each packet in real time (if each packet in the flow requires a fine-grained decision by the controller, it means that each packet will be disturbed, which will degrade network performance). On the one hand, it reduces the theoretical complexity of the probabilistic scrambling strategy; on the other hand, it reduces the negative impact on the performance of normal packets while improving the concealment of fingerprint information. In addition, randomizing the interference delay in the probabilistic scrambling strategy can also help obfuscate the delay distribution calculated by the attacker (multipeak random distribution). Compared with a uniform interference delay, the randomized interference delay can effectively reduce the overall interference delay of a normal flow.

4.1.4. Normal Case. An SDN fingerprint attacker usually sends two probe packets in a short period of time ("back-to-back" attack mode). If the interval between two packets is too long, the delay will be easily affected by the network jitter, which will influence the detection effect. When an attacker attacks in this way, the first probe packet usually triggers operations such as creating new flow rules and initializing hash entries. Then, when the second probe packet is received, the probability decision component will interfere with the packet with a probability close to 1 according to the gradient probability curve ($P_1 \approx 1$), so the delay distribution difference will be significantly reduced. It can be known from the above-mentioned theoretical analysis that the probabilistic scrambling mechanism can improve the concealment of SDN fingerprint information under normal circumstances. However, in order to apply the probabilistic scrambling mechanism to more scenarios, we need to discuss the following cases.

Case 1: the attacker sends the first probe packet to cause a new flow rule operation. At this time, if other normal users happen to trigger the update operation of the packet counter value (*counter*) in the flow, the counter value corresponding to the second probe packet sent by the attacker will be slightly larger, which will reduce the probability of attack packet interference. However, considering that attackers usually send randomly forged packets, the probability of collision between the header information of normal packets and the header information of forged packets is only theoretically possible ($P(\text{collide}) \approx 0$). In addition, the time interval between the two probe packets sent by the attacker is very short. Even if a normal user collides with an attacker, the influence of the counter value deviation of the attack packet (Δcount) on the interference probability is relatively limited ($P(\text{collide}) \cdot (1 - P_{2+\Delta\text{count}}) \approx 0$). Moreover, the attacker needs multiple tests to get the delay distribution and infer the fingerprint information, so a theoretically possible single failure will not affect the global defense effect.

Case 2: assume that after a normal user sends a flow to trigger a new flow rule, the attacker uses the flow information to construct a probe packet and sends two probe packets in a short period of time. There are four subcases in this scenario: (1) when the two probe packets are not disturbed, their round-trip time difference is approximately equal to 0 ($\Delta\text{RTT} \approx 0$). Therefore, the fingerprint information cannot be detected at this time. (2) When both probe packets are disturbed, their round-trip time difference is less than *rtt* ($\Delta\text{RTT} \ll rtt$). Thus, the time distribution difference cannot be detected at this time. (3) When the first probe packet is not disturbed and the second probe packet is disturbed, the round-trip time $\text{RTT}_1 < \text{RTT}_2$. The attacker may think that there is network jitter in the detection process and classify the result as an abnormal situation. It also does not help fingerprint identification. (4) When the first probe packet is disturbed and

the second probe packet is not disturbed, the attacker may infer the correct fingerprint information once. However, the interference will multipeak the delay distribution obtained by the attacker. It deviates from the single-peak distribution introduced in Section 3. In addition, one successful detection does not mean that the fingerprint information is successfully identified. Only when the number of successful tests (in a certain level) reaches a certain threshold, the attacker can identify the fingerprint parameters. For example, the threshold of the number of successes is $n \gg 1$, the fingerprint identification success probability $\prod_{i=1}^{n} P^i_{packet_1} \cdot (1 - P^i_{packet_2})$ is low ($P^i_{packet_x}$ represents the probability that the x_{th} probe packet is disturbed during the i^{th} successful detection).

Case 3: assume that there is an innovative attacker with strong learning ability in the network or the principle of the probabilistic scrambling strategy is leaked to outsiders (this case is only theoretically possible). The attacker may adjust the traditional "back-to-back" attack mode (i.e., sending two adjacent probe packets). For example, the attacker can send the first probe packet to trigger the installation of a new flow rule and then continuously send probe packets to the target network at appropriate intervals (should avoid triggering IDS) and record the round-trip time of each packet. With the increase of the packet count value, the probability of interference decreases gradually. When the count value increases from small to a certain level, the number of uninterrupted packets will gradually increase. Although the fingerprint information cannot be effectively identified by two adjacent packets (there is a probabilistic scrambling strategy), the distribution can be clearly obtained by measuring the round-trip time of a large number of probe packets in the flow. The undisturbed packets are concentrated in the short delay range, while the disturbed packets are scattered in the long delay range. By extracting the longest and the shortest delay values, the attacker can also identify the SDN network type and even the controller type.

4.1.5. Security Proof. Although the SDN fingerprint attack process is complicated, its theoretical basis is simple. That is, when an attacker launches an attack, the attacker generally compares the delay difference between the two probe packets. Thus, theoretically, if we want to prove whether the relevant strategy can hide the fingerprint information, we only need to derive the probability that the delays of the two probe packets are equal. The higher the probability that the delays of the two probe packets are equal, the better the fingerprint information hiding effect, and vice versa. Therefore, the core idea of the security proof is to discuss the change rule of the probability that the delays of the two probe packets are equal under different strategies. In order to prove the effectiveness of the probabilistic scrambling strategy, we first define the relevant symbols. Suppose that the sequence numbers of attack packets $probe_1$ and $probe_2$ sent by the attacker during the i_{th} attack (i.e., the forged flow $flow_i$) are represented by $\text{seq}(probe^i_1)$ and $\text{seq}(probe^i_2)$, respectively, and the expressions are as follows:

$$\begin{cases} \text{seq}\left(probe^i_1\right) = 0, \\ \text{seq}\left(probe^i_2\right) = 1 + \Delta count, \end{cases} \tag{5}$$

where the counter value deviation of the attack packet $probe_2$ is $\Delta count$. Based on the above formulas, the delay values of attack packets $probe_1$ and $probe_2$ can be denoted as $T(probe^i_1)$ and $T(probe^i_2)$, respectively, and the expressions are as follows:

$$\begin{cases} T\left(probe^i_1\right) = t + \Delta t, \\ T\left(probe^i_2\right) = t + Filter_x \cdot \Delta t, \quad x \in \{1, 2, 3\}, \end{cases} \tag{6}$$

where t represents the time required for the switch to directly forward the packet; Δt represents the additional delay caused by the interaction between the switch and the controller; $Filter_x$ represents the function determined by the scrambling strategy x, which mainly includes the following three types: (1) no scrambling strategy ($x = 1$, reference method); (2) probabilistic scrambling strategy ($x = 2$); (3) deterministic scrambling strategy ($x = 3$, reference method). The expressions of the abovementioned three strategies ($Filter_x$) are as follows:

$$\begin{cases} Filter_1 = 0, \\ Filter_2 = I\left(u, P_{\text{seq}\left(probe^i_2\right)}\right) = \begin{cases} 1, & P_{\text{seq}\left(probe^i_2\right)} > u, \\ 0, & P_{\text{seq}\left(probe^i_2\right)} < u \end{cases}, \quad u \sim U(0, 1), \\ Filter_3 = 1. \end{cases} \tag{7}$$

Based on the abovementioned definitions, we continue to discuss the probability of successfully hiding fingerprint information (i.e., $p(T(probe^i_1) = T(probe^i_2)))$ in the i^{th} attack under the three strategies. When $\Delta count = 0$, the probability under the three strategies can be obtained by combining formulas (5)~(7):

$$\begin{cases} p_{Filter_1}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right)_{\Delta count=0} = 0, \\ p_{Filter_2}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right)_{\Delta count=0} = P_{\text{seq}\left(probe_2^i\right)} = P_1 \approx 1, \\ p_{Filter_3}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right)_{\Delta count=0} = 1. \end{cases} \tag{8}$$

When $\Delta count \neq 0$, the probability under the three strategies can be obtained by combining formulas (5)~(7):

$$\begin{cases} p_{Filter_1}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right)_{\Delta count \neq 0} = 0, \\ p_{Filter_2}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right)_{\Delta count \neq 0} = P_{\text{seq}\left(probe_2^i\right)} = P_{1+\Delta count} > 0, \\ p_{Filter_3}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right)_{\Delta count \neq 0} = 1. \end{cases} \tag{9}$$

According to formulas (8) and (9), the probability of successfully hiding the fingerprint in the i^{th} attack under the three strategies is as follows:

$$\begin{cases} p_{Filter_1}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right) = p(\Delta count = 0) \cdot 0 + p(\Delta count \neq 0) \cdot 0 = 0, \\ p_{Filter_2}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right) = p(\Delta count = 0) \cdot P_1 + p(\Delta count \neq 0) \cdot P_{1+\Delta count}, \\ p_{Filter_3}\left(T\left(probe_1^i\right) = T\left(probe_2^i\right)\right) = p(\Delta count = 0) \cdot 1 + p(\Delta count \neq 0) \cdot 1 = 1. \end{cases} \tag{10}$$

By analyzing the above formulas, we can conclude that $p_{Filter_1} < p_{Filter_2} < p_{Filter_3}$. Moreover, because $P_1 \approx 1$ and $p(\Delta count = 0) \gg p(\Delta count \neq 0)$, $p_{Filter_1} \ll p_{Filter_2} \approx p_{Filter_3}$. This proves that the defensive effect of the probabilistic scrambling strategy is similar to that of the deterministic scrambling strategy.

In addition to defensive effect, the impact of the abovementioned three strategies on normal packets is also an important indicator, and the expressions are as follows:

$$\begin{cases} overhead_{Filter_1} = 0, \\ overhead_{Filter_2} = \int_{i=0}^{flow\_num} \int_{c=0}^{\min(packet\_num,100)} P_c \cdot \Delta t \mathrm{d}c\mathrm{d}i, \\ overhead_{Filter_3} = \int_{i=0}^{flow\_num} \int_{c=0}^{packet\_num} \Delta t \mathrm{d}c\mathrm{d}i, \end{cases} \tag{11}$$

where P_c is the scrambling probability value corresponding to the $packet_c$ in the flow, c is the sequence number of the packet. Because $P_c < 1$ and $\mathbf{min}(packet\_num, 100) \leq packet\_num$, we can conclude that $overhead_{Filter_1} < overhead_{Filter_2} < overhead_{Filter_3}$. This proves that the overhead of the probabilistic scrambling strategy is less than that of the deterministic scrambling strategy.

4.2. MTD-Based Controller Dynamic Scheduling Strategy

4.2.1. Motivation. By analyzing the defensive effects of the probabilistic scrambling strategy in normal attack case and three special attack cases (Section 4.1, normal case and Cases 1–3), we can find that the probabilistic scrambling strategy is effective for the normal attack case and the first two special attack cases, but it is invalid for the third special attack case. That is to say, if there is a "hidden enemy" leaking the probabilistic scrambling strategy or there is an innovative attacker with strong learning ability (the scenario in Case 3), the probabilistic scrambling strategy may also fail. Moreover, as the attack time accumulates, the probability of the abovementioned situation will gradually increase. Therefore, the probabilistic scrambling strategy is only suitable for solving the fingerprint attack problem with a limited time constraint. If we want to defend against fingerprint attacks without time constraints (the scenario in Case 3), the target network needs to actively change its fingerprint information. More specifically, we design a controller dynamic scheduling strategy based on moving target defense (MTD) [25]. By scheduling different types of controllers to work at different time periods, fingerprint information can be mixed. The information accumulation of the attacker also disappears with each renewal process. Thus, the controller dynamic scheduling strategy can be useful to prevent SDN fingerprint attacks without time constraints. However, it is worth noting

that actively changing the fingerprint information will cause the corresponding overhead. The switching point of the system will directly affect the overhead and the security performance. If the system switches too frequently, the system overhead will increase sharply; but if the system switches too slowly, the security performance of the system will decrease. Therefore, how to select the optimal switching point to balance the benefits and costs in the controller dynamic scheduling strategy is the motivation of this subsection.

4.2.2. Theoretical Basis. Different from traditional defense ideas, MTD technology improves security by dynamically adjusting key elements of the system. The dynamic adjustment operation will continuously change the attack surface, increasing the difficulty and uncertainty of the attack. This reverses the situation where the advantage of the attacker increases over time. It essentially changes the asymmetry between the defender and the attacker. For SDN fingerprint attacks, the attack surface is determined by the controller. Different controllers will specifically reflect different fingerprint information. By scheduling different types of controllers to work at different time periods, fingerprint information can be mixed. In this case, the unlimited time resources owned by the attacker will be sliced. During each slicing cycle, the attacker faces an unknown network. In this way, each slicing cycle is equivalent to the renewal cycle. The information accumulation of the attacker also disappears with each renewal process. As different types of controllers switch in different time periods according to a certain strategy, the time domain of the defense operation is also divided into multiple rounds of defense time windows. In this process, the calculation of the optimal switching point is the key step. In order to calculate the optimal switching point, we establish an optimal time scheduling model. In this model, we first define the system scheduling loss and attack loss and then evaluate the overall scheduling cost expectation and the scheduling time expectation based on the input attack time distribution. Finally, the optimal switching point can be obtained by defining the unit time cost function (the ratio of the expected overall scheduling overhead to the expected scheduling time) and deriving it. Above all, the MTD-based controller dynamic scheduling strategy will be explained in detail as follows.

4.2.3. Detailed Design. Inspired by the master-slave switching scheme of the distributed controller, the architecture of the controller dynamic scheduling is shown in Figure 9. We introduce an intermediate scheduling layer to realize the controller dynamic scheduling strategy. The intermediate scheduling layer is mainly composed of a data proxy module, an evaluation module, and a scheduling module. The data proxy module is mainly responsible for transmitting the status and instruction interaction between the control plane and the data plane. In addition, since the controller deploys the probabilistic scrambling strategy, the data proxy module is also responsible for transmitting the status information (such as flow mapping information in the hash table) of the probabilistic scrambling strategy. The evaluation module consists of an attack loss evaluation component and a scheduling cost evaluation component. After receiving the state information of the control plane and the data plane collected by the data proxy module, this module evaluates the attack loss and the controller scheduling overhead, respectively, so as to balance the benefits and costs. The scheduling module is the core module of the controller dynamic scheduling strategy. The module mainly includes a scheduling time decision component and a scheduling execution component. The scheduling time decision component can calculate the optimal controller switching point based on the attack loss and scheduling costs to ensure the lowest comprehensive cost per unit time. The scheduling execution component selects a controller from the backup controller pool. It is worth noting that the type of the selected controller is different from the type of the current master controller. In this way, we can achieve the purpose of actively changing fingerprint information. The controller dynamic scheduling algorithm is shown in Algorithm 3.

In order to prevent the innovative attacker with strong learning ability from breaking through the barriers of the probabilistic scrambling strategy, we design a controller dynamic scheduling strategy. Due to the lack of obvious attack characteristics, SDN fingerprint attacks cannot be detected and recorded by various existing defense tools. In addition, the attack method of the innovative attacker is very special (detailed in Case 3), so it is difficult to collect the real fingerprint attack dataset and use it to guide dynamic scheduling. However, the probability that an innovative attacker appears in an SDN fingerprint attack is similar to the probability that the attacker successfully launches an attack in a normal network attack: (1) similar to the successful attack in a normal network, the occurrence of innovative attackers in SDN fingerprint attacks also means that the probability of successful fingerprint attacks has increased significantly; (2) with the increase of detection time, the probability of successful attacks in the normal network increases, and the probability of innovative fingerprint attackers in SDN fingerprint attacks also increases. Therefore, it is reasonable to use the time interval of successful attacks in the normal network to roughly simulate the appearance time interval of innovative fingerprint attackers in SDN fingerprint attacks. In this way, the controller dynamic scheduling strategy can take the common attack distribution as an input to calculate the optimal scheduling time series and actively change the fingerprint information based on the time series. Before controller scheduling, the intermediate scheduling layer first needs to collect key state information *INF* from the main controller and the data plane (network topology scale, throughput, and hash tables in the probabilistic scrambling strategy) and use this information to evaluate the scheduling cost C_s and attack loss L_a(lines 2-3). Then, the scheduling time decision component fits the input attack distribution dataset. If it cannot match any similar distribution, the component takes the average value of the attack time series in the dataset (i.e., the average value of the time interval when the attacker appears) as the

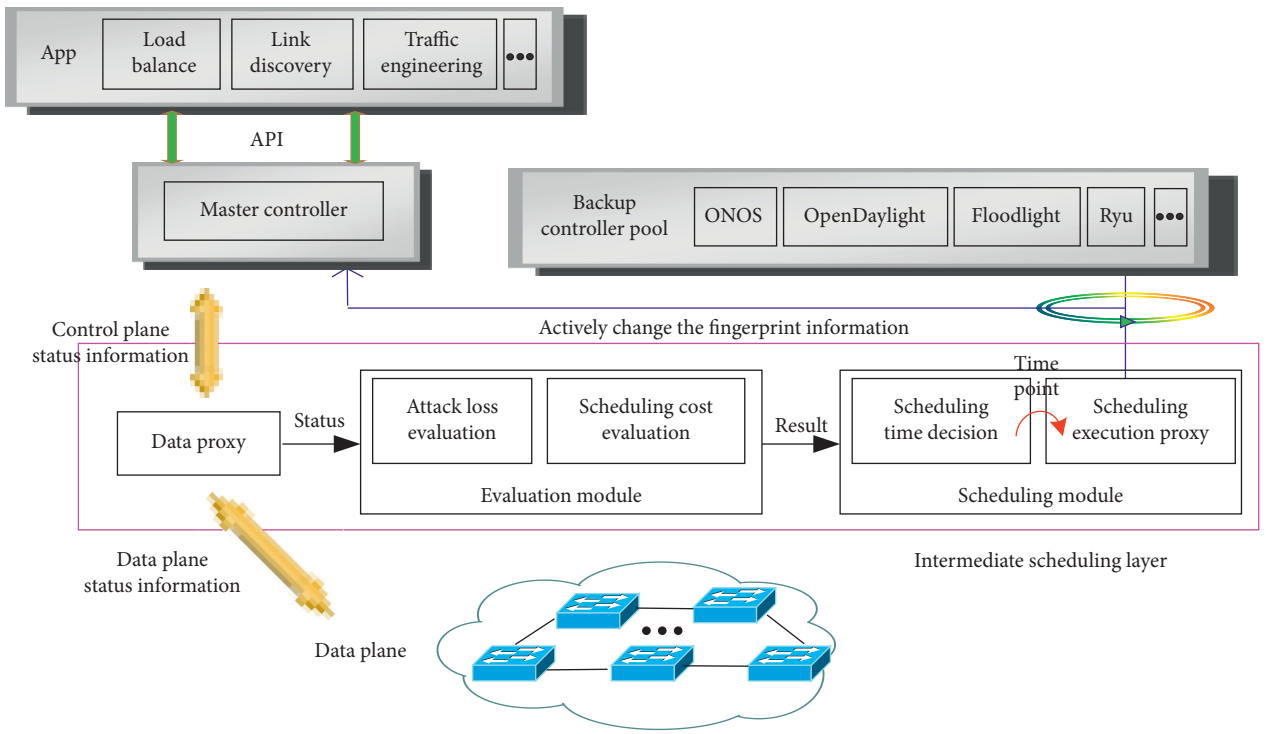


Figure 9: Controller dynamic scheduling architecture.

Input: the dataset (successful attack interval time series sample) D
Output: the list of scheduling time series ST_{list}
(1) **while** *TRUE* **do**
(2) Collect state information *INF* from master controller and data plane
(3) Estimate scheduling cost C_s and attack loss L_a based on Step 2
(4) Fit a distribution $F(t)$ based on D
(5) **if** cannot match any suitable distribution, **then**
(6) Scheduling time $T \leftarrow meanValue(D)$
(7) **else**
(8) $T \leftarrow deriveTime(D, C_s, L_a)$
(9) **end if**
(10) The elapsed time since the last controller scheduling $t_{elapse} \leftarrow 0$
(11) **while** $t_{elapse} < T$ **do**
(12) **if** $Percentage_{Flow(Counter<3)} >$ threshold in *INF(HashTable)*, **then**
(13) **break**
(14) **else**
(15) Update t_{elapse}
(16) **end if**
(17) **end while**
(18) Update $ST_{list} \leftarrow ST_{list} \cup \mathbf{min}(t_{elapse}, T)$
(19) Start the controller scheduling based on ST_{list}
(20) **end while**

Algorithm 3: Controller dynamic scheduling strategy.

scheduling time T. If a suitable distribution is matched (such as Poisson distribution, exponential distribution, uniform distribution, and normal distribution), then T is calculated according to the subsequent optimal scheduling time model (lines 4–9). After calculating the scheduling time T, the scheduling time decision component passes the value to the scheduling execution proxy component to execute the scheduling. It is worth noting that the scheduling execution proxy component only executes the scheduling by referring to the scheduling time and does not execute strictly according to that time. The input of this algorithm is the time interval distribution dataset of successful attacks in normal network. Although this dataset has a certain degree of similarity to the dataset of fingerprint attackers in real

scenarios, there are still inevitable differences. Therefore, if the scheduling execution component executes the scheduling unconditionally and strictly according to the results, it may face the risk of fingerprint information leakage in some cases. In order to eliminate the impact of dataset differences, we introduce a lightweight decision process in the controller dynamic scheduling strategy. When the time elapsed after the execution of the scheduling is less than the theoretical scheduling time, the component continuously monitors whether the proportion of the flows whose packet count value is less than 3 is greater than the alert threshold. If the value is greater than the alert value, it means that the innovative attacker is more likely to appear. The scheduling execution component needs to jump out of the loop and immediately execute the controller scheduling to actively change the fingerprint information. If the value is less than the alert value, it means that the innovative attacker is less likely to appear. The scheduling execution component can still guide the controller to schedule based on the scheduling time (lines 11–19).

The controller dynamic scheduling process is a typical renewal process. The intermediate dynamic scheduling layer calculates the controller scheduling time series based on the status information. Whenever the controller finishes scheduling, it will wait to start a new round of scheduling, so that the attacker's information accumulation in the previous scheduling cycle will also be invalidated. In this process, determining the optimal scheduling time is an important part of the controller dynamic scheduling strategy. To solve this problem, we construct the following optimal scheduling time model:

Let T^i_{smart} denote the detection time required for an innovative attacker to appear after the controller performs the i^{th} scheduling. $F(t)$ represents its probability distribution function, and $f(t)$ is the probability density function. The relationship between them is shown in the following formula:

$$F(t) = \int_0^t f(t)\mathrm{d}t, \tag{12}$$

where$F(t)$ is obtained by fitting the input dataset. The physical meaning of $f(t)$ in this scenario is the probability that an innovative attacker will appear after time t since the last controller scheduling. Let T^i_{defense} denote the theoretical time interval between the $i-1^{\text{th}}$ controller scheduling and the i^{th} controller scheduling calculated by the intermediate dynamic scheduling layer. Similarly, T^i_{actual} represents the actual time interval between the $i-1^{\text{th}}$ controller scheduling and the i^{th} controller scheduling. By analyzing Algorithm 3, it can be known that when the flow state does not satisfy the lightweight decision condition, the scheduling execution component schedules according to the calculated theoretical time interval. However, if the flow state meets the lightweight decision conditions, the scheduling execution component immediately dispatches the controller, so the i^{th} actual scheduling time is as shown in the following formula:

$$T^i_{actual} = \begin{cases} T^i_{de\,fense}, & \text{if } T^i_{smart} > T^i_{de\,fense}, \\ T^i_{smart}, & \text{if } T^i_{smart} < T^i_{de\,fense}. \end{cases} \tag{13}$$

In the case that the first inequality of formula (13) is satisfied, the defender actively changes the fingerprint information before the innovative attacker appears, so the cost of the defender in the i^{th} scheduling only includes the scheduling cost C_s.When the second inequality is satisfied, it means that there is an innovative attacker before the controller scheduling. In this way, the defender needs to bear an additional attack loss and then perform scheduling immediately. Therefore, the cost of the defender's i^{th} scheduling is shown in the following formula:

$$Cost_i = \begin{cases} C^i_s, & \text{if } T^i_{smart} > T^i_{de\,fense}, \\ L^i_a + C^i_s, & \text{if } T^i_{smart} < T^i_{de\,fense}. \end{cases} \tag{14}$$

We calculate the expected value of the i^{th} actual scheduling time T^i_{actual} and the i^{th} scheduling cost $Cost_i$ according to the following formulas:

$$\begin{aligned} E\left(T^i_{actual}\right) &= \int_0^{T^i_{de\,fense}} t \cdot f(t)\mathrm{d}t + \int_{T^i_{de\,fense}}^{\infty} T^i_{de\,fense} \cdot f(t)\mathrm{d}t \\ &= \int_0^{T^i_{de\,fense}} t \cdot f(t)\mathrm{d}t + T^i_{de\,fense} \cdot \left(1 - F\left(T^i_{de\,fense}\right)\right), \end{aligned} \tag{15}$$

$$\begin{aligned} E\left(Cost_i\right) &= C^i_s \cdot P\left(T^i_{smart} > T^i_{de\,fense}\right) + \left(L^i_a + C^i_s\right) \\ &\quad \cdot P\left(T^i_{smart} \le T^i_{de\,fense}\right) \\ &= C^i_s \cdot \left(P\left(T^i_{smart} > T^i_{de\,fense}\right)\right. \\ &\quad \left. + P\left(T^i_{smart} \le T^i_{de\,fense}\right)\right) \\ &\quad + L^i_a \cdot P\left(T^i_{smart} \le T^i_{de\,fense}\right) \\ &= C^i_s + L^i_a \cdot F\left(T^i_{de\,fense}\right). \end{aligned} \tag{16}$$

Controller dynamic scheduling will inevitably temporarily affect the overall network performance and quality of service. The cost mainly includes two parts: (1) when the underlying switch reestablishes the connection with the controller, some packets may be lost due to the controller dynamic scheduling, which will cause scheduling cost. (2) When the controller dynamically schedules, the value list in the hash table will be refreshed (only the index information is retained), so the probability scrambling mechanism will cause a performance loss to the flows involved in the refresh operation. Therefore, the controller scheduling cost is shown in the following formula:

$$C^i_s = \omega_s \cdot \kappa \cdot \mu\lambda + \sum_{x\in\Theta}\sum_{y=1}^{100} delay \cdot P_y^{Flow_x}, \tag{17}$$

where ω_s represents the weighting coefficient of the scheduling cost. The number of switches in the target network is represented by κ. μ is the number of users connected to a single switch. λ represents the arrival rate parameter of the user flow (the user-generated flow satisfies

a Poisson distribution with parameter λ). Θ represents the index set of the hash table before the controller scheduling. x is the flow index. y indicates the index of the packet in the flow. *delay* represents the expected packet delay. $P_y^{Flow_x}$ corresponds to the interference probability of the y^{th} packet in the x^{th} flow. In order to simplify the solution, this paper describes the first part of the scheduling cost as a linear form of network throughput (the larger the network size, the higher the network throughput and the scheduling cost). By adjusting the appropriate weighting coefficient ω_s, the cost and the network throughput can be reasonably mapped, and the weight of the two parts of the cost can be controlled at the same time. In addition, the delay cost introduced by refreshing the hash table is used as the second part of the scheduling cost. In the probabilistic scrambling strategy, the probability of interference is low when the packet count is greater than 100, so the packet index is increased to 100 in formula (17). Similarly, the attack loss is defined as follows:

$$L_a^i = \omega_a \cdot \kappa \cdot \mu\lambda + \sum_{x\in\Theta}\sum_{y\in\Phi} delay \cdot P_y^{Flow_x}, \tag{18}$$

where ω_a represents the weighting coefficient of the attack loss, and Φ represents the set of count values in the hash table before the controller scheduling. It can be seen from formula (18) that the first part of the attack loss is still described in the linear form of the network throughput rate, and the second part is the delay loss actually generated by the probabilistic scrambling strategy on the packets before the controller scheduling.

Based on the abovementioned analysis, this paper defines the ratio of the expected value of the i_{th} actual scheduling time T_{actual}^i to the i^{th} scheduling cost $Cost_i$ as the unit time cost and uses this as an indicator to measure the controller's dynamic scheduling effect. The unit time cost is shown in the following formula:

$$\begin{aligned}\xi\left(T_{de\,fense}^i\right) &= \frac{E\left(Cost_i\right)}{E\left(T_{actual}^i\right)} \\ &= \frac{C_s^i + L_a^i \cdot F\left(T_{de\,fense}^i\right)}{\int_0^{T_{de\,fense}^i} t \cdot f(t)dt + T_{de\,fense}^i \cdot \left(1 - F\left(T_{de\,fense}^i\right)\right)}.\end{aligned} \tag{19}$$

In order to obtain the optimal controller dynamic scheduling effect, the unit cost of scheduling needs to be minimized. Therefore, we continue to derive the derivative of $\xi(T_{de\,fense}^i)$ to obtain the optimal time $T_{de\,fense}^i$:

$$\min \xi\left(T_{de\,fense}^i\right) \Rightarrow \frac{\partial \xi\left(T_{de\,fense}^i\right)}{\partial T_{de\,fense}^i} = 0. \tag{20}$$

4.2.4. Security Proof. Since the optimal switching point has been specified in the optimal scheduling time model, it will not be repeated here. This part mainly measures the security gain caused by the dynamic scheduling process. More specifically, based on the MTD-I/O automata model, we define the attack surface as a triplet surf = <*I*, *O*, *C*>, which represents the entry point, exit point, and channel of the system, respectively [26]. The degree of the attack surface can be expressed as follows:

$$\text{DEG}_{\text{surf}} \langle \sum_{i\in I} \deg(i), \sum_{o\in O} \deg(o), \sum_{c\in C} \deg(c) \rangle, \tag{21}$$

where deg(*i*), deg(*o*), and deg(*c*) represent the security threat weights of elements *i*, *o*, and *c* in sets *I*, *O*, and *C*, respectively. Based on the abovementioned definition, we assume that the system attack surface during the x^{th} controller scheduling is $\text{surf}_x = <I_x, O_x, C_x>$. After the scheduling is completed, its attack surface is $\text{surf}_{x+1} = <I_{x+1}, O_{x+1}, C_{x+1}>$. Meanwhile, in order to measure the degree of the attack surface shifting, we define two attack surface operation rules as follows:

$$\begin{cases} \text{surf}_\cap = \text{surf}_x \cap \text{surf}_{x+1} = \langle I_x \cap I_{x+1}, O_x \cap O_{x+1}, C_x \cap C_{x+1} \rangle = \langle I_\cap, O_\cap, C_\cap \rangle, \\ \text{surf}_- = \text{surf}_x - \text{surf}_{x+1} = \langle I_x - I_{x+1}, O_x - O_{x+1}, C_x - C_{x+1} \rangle = \langle I_-, O_-, C_- \rangle, \end{cases} \tag{22}$$

where $I_x - I_{x+1} = \{i | i \in I_x, i \notin I_{x+1}\}$, $O_x - O_{x+1} = \{o | o \in O_x, o \notin O_{x+1}\}$, and $C_x - C_{x+1} = \{c | c \in C_x, c \notin C_{x+1}\}$. Combining formulas (21) and (22), we define the degree of the attack surface shifting as follows:

$$\begin{aligned}\Delta\text{DEG}_{\text{surf}_x} = \langle & \sum_{i\in I_\cap} \left(\deg_x(i) - \deg_{x+1}(i)\right) + \sum_{i\in I_\cap} \deg_x(i), \\ & \sum_{i\in I_\cap} \left(\deg_x(o) - \deg_{x+1}(o)\right) + \sum_{i\in I_\cap} \deg_x(o), \\ & \sum_{i\in I_\cap} \left(\deg_x(c) - \deg_{x+1}(c)\right) + \sum_{i\in I_\cap} \deg_x(c) \rangle.\end{aligned} \tag{23}$$

Therefore, the security gain of the controller dynamic scheduling strategy (CDS) is

$$\text{SG}_{\text{CDS}} \propto \sum_x \text{SG}_x = \sum_x \frac{\Delta\text{DEG}_{\text{surf}_x}}{\text{DEG}_{\text{surf}_x} \cdot T_{\text{defense}}^x} > 0. \tag{24}$$

If the system is completely static (the $\Delta\text{DEG}_{\text{surf}} = 0$ and $T_{\text{defense}} \longrightarrow \infty$), the security gain of the static strategy ($\text{SG}_{\text{static}}$) will be 0. Therefore, we can prove that $\text{SG}_{\text{CDS}} \gg \text{SG}_{\text{static}}$. However, because the MTD theory in the SDN scenario is very immature, the differences in *I*, *O*, and *C* sets of different types of controllers cannot be quantitatively measured. Therefore, we can only give the general mathematical model of the security gain in theory but cannot

perform specific numerical calculations. In the future work, we will focus on this research content.

5. Evaluation

In order to avoid attackers using SDN special delay attributes to infer fingerprint information, we propose the probabilistic scrambling strategy and the controller dynamic scheduling strategy in Section 4. In this section, we will design experiments to test the defense effect of this mechanism.

5.1. Probabilistic Scrambling Experiment. The experimental environment of the probabilistic scrambling strategy is shown in Figure 10. The experimental environment includes four physical servers (Intel(R) Xeon(R) CPU E5-2600 v4, 2.1 GHz, 16 GB memory, Ubuntu 16.04) and one Pica8 switch. Among them, the first to third servers, respectively, run four virtual machines as network users (including attackers), and the fourth server runs the Floodlight controller (including the probabilistic scrambling mechanism). We configured the Pica8 switch to generate six OVSs connected to each other, and each OVS connected two host users.

To test the defense effect of the probabilistic scrambling strategy, we first make the attacker construct and send probe *ping* packets containing different destination addresses without enabling the probabilistic scrambling strategy. For each destination address, the attacker sends two probe packets in a short period of time ("back-to-back" attack mode). Then, the attacker separately records the round-trip time of the first probe packet (FP) and the second probe packet (SP) in each flow. Similarly, we repeat the above-mentioned steps while the controller is applying the probabilistic scrambling strategy. The round-trip time results in both cases are shown in Figure 11.

We can qualitatively analyze Figure 11 to draw the following conclusions: when the probabilistic scrambling strategy is not deployed, there is a significant delay difference between the first probe packet and the second probe packet due to the interaction between the controller and the switch; after the probability scrambling strategy is deployed, this delay difference is clearly blurred. In order to quantitatively describe the defense effect of the probability scrambling strategy, we define the delay fuzzy ratio as shown in the following formula:

$$\text{FuzzyRatio} = \frac{\text{mean}\left(\Gamma_{\text{SP}}\right)}{\text{mean}\left(\Gamma_{\text{FP}}\right)} = \frac{\sum_{i=1}^{k} \text{RTT}_i^{\text{SP}}}{\sum_{i=1}^{k} \text{RTT}_i^{\text{FP}}}, \tag{25}$$

where Γ_{FP} and Γ_{SP} represent the round-trip time set of the first packet and the second packet, respectively. RTT_i^{FP} and RTT_i^{SP} represent the corresponding value of the i_{th} packet in the round-trip time set Γ_{FP} and Γ_{SP}, respectively. k is the number of elements in the set. The closer the value of the fuzzy ratio is to 0, the clearer the round-trip time difference. The closer the value is to 1, the more fuzzy the round-trip time difference. By analyzing the results of 100 flows in this experiment, we can calculate that the delay fuzzy ratio is about 0.3 when the probabilistic scrambling strategy is not used, but the delay fuzzy ratio increases to 0.92 when the probabilistic scrambling strategy is used. This proves that the defense effect of this mechanism is remarkable.

Taking the round-trip time of the first packet and the second packet of different probe flows as indicators can indirectly reflect the accuracy of fingerprint identification, to directly reflect the influence of the probabilistic scrambling strategy on the fingerprint recognition accuracy, we use the ΔRTT distribution as an indicator. In order to improve the reusability of the experimental environment, we first make the controller actively generate and transfer sample flow rules in batches (this operation is to simulate the flow corresponding to the flow rule as a flow in a normal network) and then continuously send packets corresponding to the flow rules and record the corresponding ΔRTT. Similarly, we let the attacker send flows to different destination addresses before and after the Floodlight controller deploys the probabilistic scrambling strategy (there is no matching flow rule before sending) and record the ΔRTT of the corresponding packet. At last, we repeat the above-mentioned experiments on different types of controllers (Ryu and OpenDaylight). The experimental results are shown in Figure 12.

In Figure 12, PDF_{Y} represents the distribution curve of ΔRTT in the presence of flow rules (this curve corresponds to the normal network type). PDF_{N}-ODL, PDF_{N}-Floodlight, and PDF_{N}-Ryu, respectively, represent the ΔRTTdistribution curves of the controllers OpenDaylight, Floodlight, and Ryu in the absence of flow rules and the probabilistic scrambling strategy (the curve corresponds to the SDN network type). PDF_{D}-ODL, PDF_{D}-Floodlight, and PDF_{D}-Ryu, respectively, represent the ΔRTT distribution curves of the controllers OpenDaylight, Floodlight, and Ryu after deploying the probabilistic scrambling strategy. We first analyze the relationship between the ΔRTT distribution curve of the normal network and the SDN network. Assume that the attacker does not know the corresponding network type before obtaining PDF_{Y} and PDF_{N}, so the attacker needs to further analyze these two datasets and classify them. Considering that the attacker can grasp the ΔRTT difference between the SDN network and the normal network, the attacker will classify the data samples whose ΔRTT is greater than the threshold as the SDN network type and classify the data samples whose ΔRTT is less than the threshold as the normal network type. In this process, two kinds of errors are inevitably introduced. The first type of error is the false alarm rate (FAR), that is, the data samples of normal networks are erroneously classified as SDN network data samples. The second type of error is the missing alarm rate (MAR), that is, the data samples of the SDN network are mistakenly classified as normal network data samples. The equal error rate (EER) can be calculated based on the confusion matrix which is composed of the correct classification rate, FAR, and MAR. [27]. The value range of EER is between 0% and 100%. When the EER value is close to 50%, it proves that the two data samples completely overlap and cannot be classified (that is, the common network type and the SDN network type cannot be distinguished). When the EER is close to 0%, the classification is completely correct. When the EER is

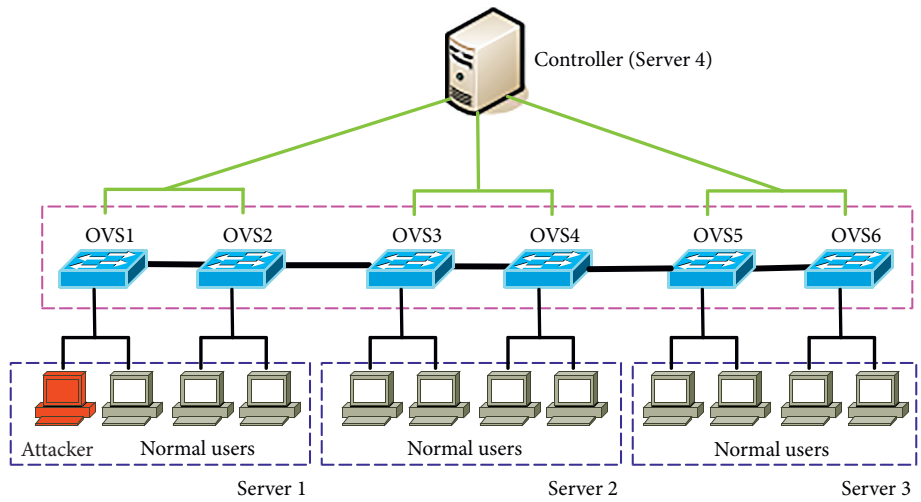


Figure 10: Probabilistic scrambling experiment environment.

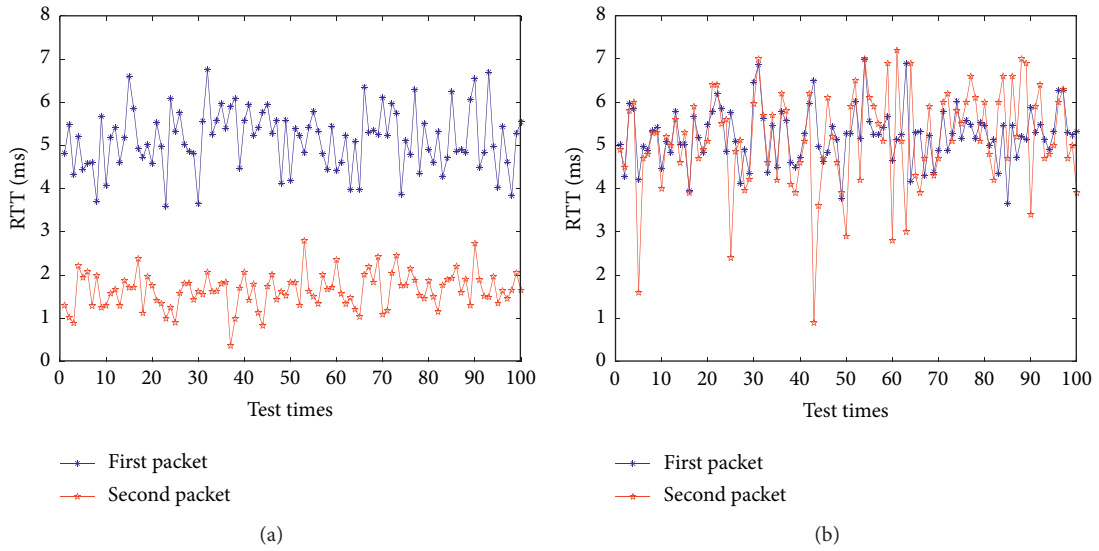


Figure 11: RTT comparison of the probe packets. (a) RTT of the FP and SP before deploying probabilistic scrambling strategy. (b) RTT of the FP and SP after deploying probabilistic scrambling strategy.

close to 100%, the classification results are completely inverted. Based on the abovementioned analysis, the EER values of PDF_Y and PDF_N are shown in the first row of Table 1. Similarly, when analyzing the relationship between the ΔRTT distribution curve before and after the deployment of the probabilistic scrambling strategy (i.e., PDF_N and PDF_D), we still use the EER value as an indicator. The results are shown in the second row of Table 1.

It can be seen from Table 1 that the EER values of PDF_N and PDF_Y or PDF_N and PDF_D are all less than 1%. This proves that there is a significant difference in the ΔRTT distribution before and after the deployment of the probabilistic scrambling strategy (just as the difference between the common network type and the SDN network type). The difference between the common network type and the SDN network type indicates that the attacker can fully identify the network type information through the probe information, so the defender must reverse this difference. Similarly, the difference before and after the deployment of the probabilistic scrambling strategy indicates that the strategy can effectively interfere with the SDN fingerprint information, and the degree of interference is sufficient to hide its attributes. In order to quantitatively measure the degree of interference, we analyze the relationship between PDF_D and PDF_Y of different types of controllers (OpenDaylight, Floodlight, and Ryu) and find that the EER values are 49.63%, 49.26%, and 49.05%. This value indicates that the ΔRTT distribution of the controller which applies the probabilistic scrambling strategy basically coincides with the ΔRTT distribution of the normal network. This strategy has successfully changed the SDN network type fingerprint information to the normal network type fingerprint

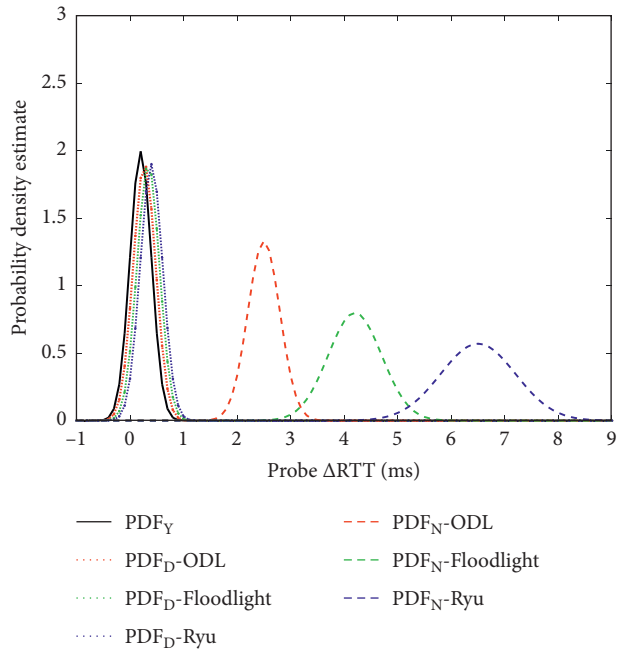


Figure 12: ΔRTT distribution comparison of the probe packets.

Table 1: Equal error rate comparison.

(EER)	PDF$_N$		
	PDF$_N$-ODL (%)	PDF$_N$-Floodlight (%)	PDF$_N$-Ryu (%)
PDF$_Y$	0.75	0.42	0.17
PDF$_D$-ODL	0.81	—	—
PDF$_D$-Floodlight	—	0.47	—
PDF$_D$-Ryu	—	—	0.20

information. Similarly, the probabilistic scrambling strategy can make the ΔRTT distribution in the absence of critical flow rules consistent with the ΔRTT distribution in the presence of critical flow rules. Therefore, this strategy can also significantly interfere with the critical flow rule fingerprint information.

In order to measure the degree of interference of the probabilistic scrambling strategy on the controller type information, we further analyze the relationship between PDF$_N$-ODL, PDF$_N$-Floodlight, and PDF$_N$-Ryu. The EER values between OpenDaylight and Floodlight, OpenDaylight and Ryu, and Floodlight and Ryu are 10.17%, 7.69%, and 9.14%. This value indicates that different controller types have strong distinguishable characteristics when the probabilistic scrambling strategy is not applied. However, when the probabilistic scrambling strategy is applied, the EER values of PDF$_D$-ODL, PDF$_D$-Floodlight, and PDF$_D$-Ryu are 49.52%, 49.46%, and 49.81%, which proves that the probabilistic scrambling strategy significantly hides the SDN controller type fingerprint information.

Through the abovementioned analysis, it can be found that the probabilistic scrambling strategy has a significant effect on hiding fingerprint information of network types, controller types, and critical flow rules. However, this mechanism may inevitably cause performance loss to some packets. In order to comprehensively measure the impact of this strategy on the overall network performance, we tested the average response time of all users' packets. Then, we adopted the strategy of delaying all packets (DEP) to repeat the experiment and compared with the results without adopting any strategy. The experimental results are shown in Figure 13. It can be seen from Figure 13 that when the DEP strategy is adopted, the packet delay is normalized to the maximum interaction delay, which is much higher than the average delay when the other two strategies are adopted. Therefore, the performance impact of the DEP strategy is much greater than the probabilistic scrambling strategy. Due to the unique properties of the gradient probabilistic scrambling curve, the delay when using the probabilistic scrambling strategy is slightly higher than the delay when no strategy is adopted in the period when the test is just started. However, as the test time passes, the delay gradually decreases until it is close to the delay when no strategy is adopted. Therefore, the negative impact of the probabilistic scrambling strategy on the overall network performance is relatively limited (almost no negative impact on the performance over time). This can meet the performance requirements of defenders.

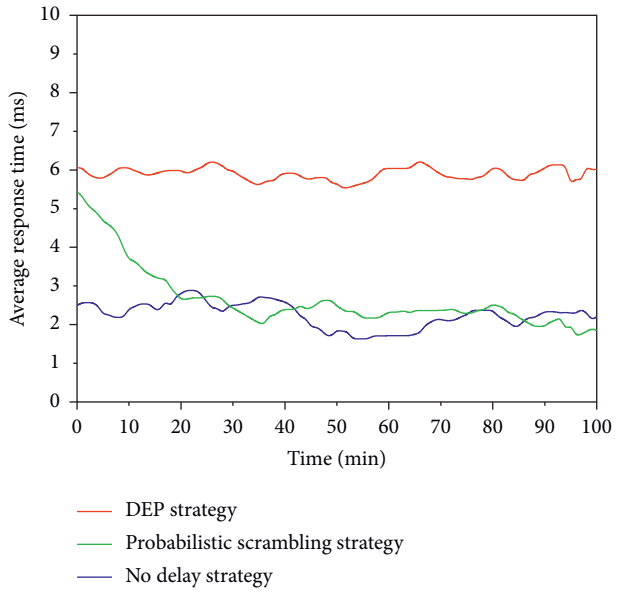


Figure 13: Average response time under different strategies.

5.2. Controller Dynamic Scheduling Experiment. In order to test the effectiveness of the controller dynamic scheduling strategy, we select three different types of controllers such as OpenDaylight, Floodlight, and Ryu to form a backup controller pool. All controllers use the probabilistic scrambling strategy. We use the attack dataset in [28] as an input sample. We also built three experimental topologies (including 100, 200, and 300 switches, respectively). Each controller randomly manages two users. The user sends data flows according to the Poisson distribution. The specific experimental parameters are shown in Table 2.

In order to compare the effectiveness of the controller dynamic scheduling mechanism horizontally, this paper selects seven common controller dynamic scheduling strategies as a control experiment. Specifically, the seven controller dynamic scheduling strategies take the maximum value ($CDS_{maximum}$), minimum value ($CDS_{minimum}$), average value ($CDS_{average}$), median value (CDS_{median}), upper quartile value ($CDS_{upper\text{-}quartile}$), lower quartile value ($CDS_{lower\text{-}quartile}$), and random value ($CDS_{random}$) of the attack time interval distribution as the switching time point, respectively. The abovementioned seven controller dynamic scheduling strategies are abbreviated as symbols MAX, MIN, AVG, MED, Q75, Q25, and RAN, respectively. According to the abovementioned dynamic scheduling strategies and the controller dynamic scheduling (CDS) mechanism proposed in Section 4.2, the unit cost results are shown in Figure 14.

It can be found from Figure 14 that compared with other scheduling strategies, the controller dynamic scheduling mechanism (CDS) has the optimal unit cost value. In addition, as the topology scale increases (100–300) and the flow arrival rate increases (0.5–10), the change in unit cost can be relatively stable within a reasonable range. This is because the CDS strategy can determine the optimal scheduling time point on the basis of comprehensive consideration of scheduling cost and attack loss. The unit cost of the $CDS_{minimum}$ strategy is the highest (the worst effect). Moreover, as the topology scale expands and the flow arrival rate increases, the unit cost increases significantly. This strategy does not comprehensively measure the attack time distribution of the attacker, resulting in the controller scheduling blindly in a short time. Especially when the topology scale is enlarged and the flow arrival rate is increased, the cost of blind scheduling is amplified, so the unit cost of this scheduling strategy is the highest. The unit cost of the $CDS_{maximum}$ strategy is significantly less than that of the $CDS_{minimum}$ strategy but greater than that of the CDS strategy. This is because the $CDS_{maximum}$ strategy can significantly reduce the scheduling cost compared to the $CDS_{minimum}$ strategy. However, due to the larger scheduling interval of the $CDS_{maximum}$ strategy, the probability of being attacked will also increase significantly compared to the CDS strategy, so its unit cost will be slightly higher than the CDS strategy. The strategy of randomly selecting the scheduling time point has a strong chance. If the scheduling interval is too large (similar to $CDS_{maximum}$), the defense failure rate will increase. If the scheduling interval is too small (similar to $CDS_{minimum}$), the scheduling cost will increase accordingly. Therefore, compared with the CDS strategy, the CDS_{random} cannot effectively balance the two costs and obtain a lower unit cost.

In order to further measure the defensive effect of the controller dynamic scheduling strategy (CDS), we launch a fingerprint attack in the manner of an innovative attacker. In an experimental environment where the number of switches is 200 and the flow arrival rate is 1, we calculate the RTT distribution before and after the deployment of the controller dynamic scheduling strategy (the initial controller is Floodlight). The experimental results are shown in Figure 15. It can be seen from the figure that before the controller dynamic scheduling strategy is deployed (only the probabilistic scrambling strategy exists), the innovative attacker can obviously obtain the double-peak RTT distribution curve through the attack method described in Case 3. The peaks are in the interval [2, 3, 6, 7], and the EER value is 0.67%, which indicates that the innovative attacker can completely infer the network type or even the controller type through the statistical results. After we deploy the controller dynamic scheduling strategy, the attacker also uses the attack method described in Case 3 to detect fingerprint information. Because the controller type is constantly changing, the response times of different types of controllers are intermixed so that there is no obvious RTT distribution law (the corresponding EER values are close to 50%). Therefore, the controller dynamic scheduling strategy can prevent the attacker from extracting fingerprint information such as controller types, thereby effectively improving the hiding degree of fingerprint information.

Finally, in order to intuitively reflect the impact of the change of weighting coefficient ratio on scheduling time, this paper will explore the change rule of scheduling time interval with weighting coefficient ratio under different scheduling strategies. For ease of operation, we first set the weighting coefficient of the attack loss to 1 and then gradually adjust

Table 2: Experimental parameter settings.

Symbol	Meaning	Value
ω_s	The weighting coefficient of the scheduling cost	2
ω_a	The weighting coefficient of the attack loss	1
κ	The number of switches in the target network	100–300
μ	The number of users connected to a single switch	2
λ	The arrival rate parameter of the user flow	0.5, 1, 10

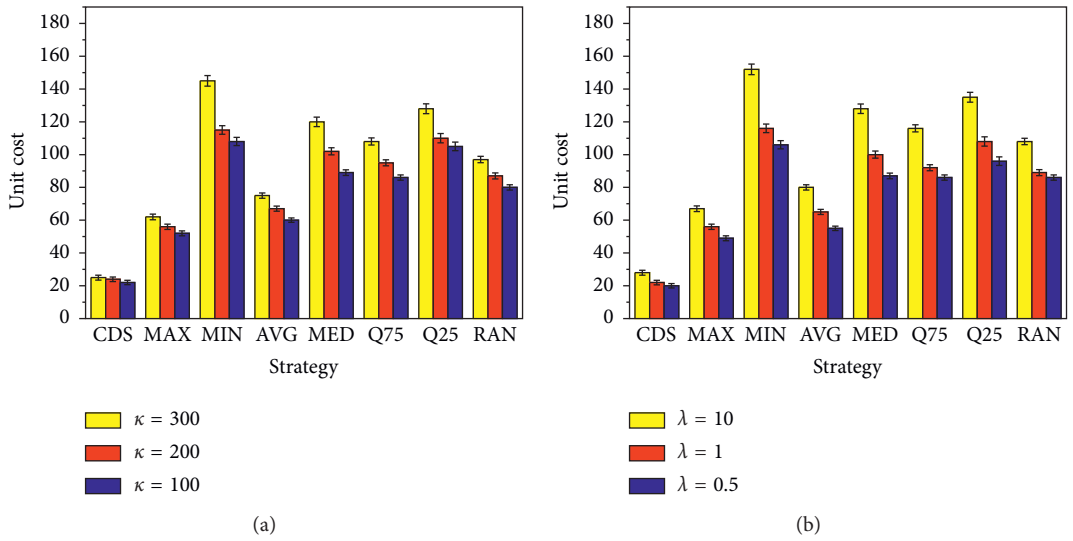


Figure 14: Unit defense cost under different strategies. (a) Under different topologies ($\lambda = 1$). (b) Under different λ ($\kappa = 200$).

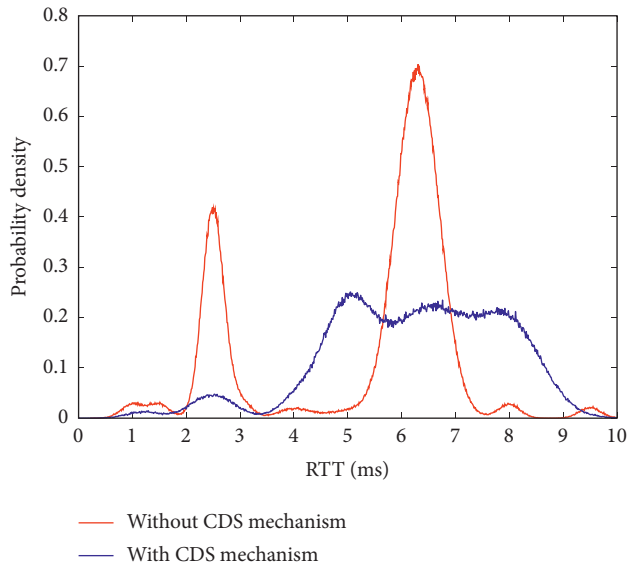


Figure 15: RTT distribution before and after controller dynamic scheduling.

the weighting coefficient of scheduling cost to make the weighting factor ratio between [1, 20]. The experimental results are shown in Figure 16.

It can be seen from Figure 16 that as the weighting coefficient ratio increases (i.e., the scheduling cost increases), the scheduling time interval of the controller dynamic scheduling mechanism also gradually increases. When the scheduling cost increases, the frequent controller scheduling will significantly increase the unit cost. Therefore, in order to ensure the optimal unit cost, the scheduling interval needs to be extended appropriately. The other seven scheduling strategies are based on a relatively fixed time strategy. Even

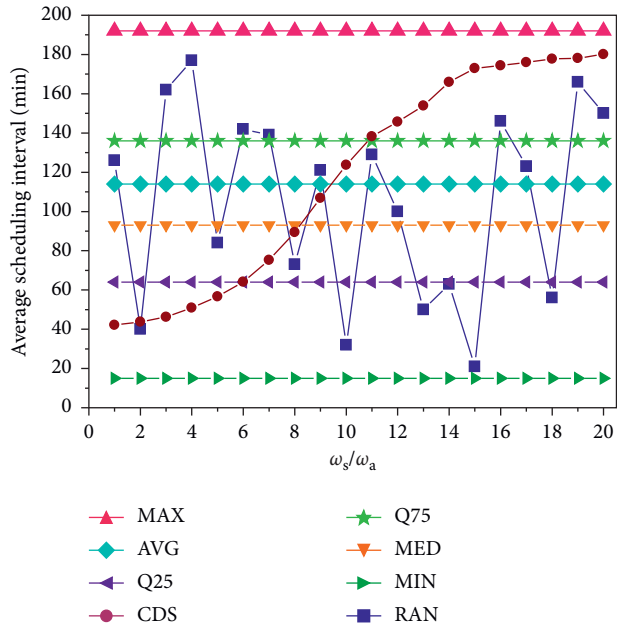


Figure 16: Scheduling time interval under different weighting coefficient ratios.

when the weighting coefficient ratio changes, the system still switches the controller at a relatively fixed time interval, which inevitably leads to a higher unit cost and affects the defense effect. In addition, the decision time of the controller dynamic scheduling mechanism fluctuates slightly with the size of the topology, but it can be controlled within 10 s, which can be approximately ignored compared to the scheduling interval. In summary, the controller dynamic scheduling mechanism has obvious defense effects and meets performance requirements.

6. Conclusion

With the large-scale deployment of SDN in data center and other scenarios, its security issues have received more and more attention. Forwarding packets in SDN requires frequent interactions between the control plane and the data plane. Therefore, compared with traditional networks, the delay distribution of SDN has typical properties. SDN fingerprint attackers can use this attribute to identify fingerprint information such as network types, controller types, and critical flow rules. In order to interfere with the attacker to obtain the correct fingerprint information, this paper proposes a probabilistic scrambling strategy and a controller dynamic scheduling strategy. In the single-round defense time window, the probabilistic scrambling strategy interferes with the delay of the packet with a certain probability according to the gradient probability curve, which greatly changes the delay distribution and reduces the negative impact on network performance. In the multiround defense time window, the controller dynamic scheduling strategy can select the optimal scheduling interval to ensure the lowest unit cost, which is beneficial to balance the defense benefits and costs. Multiple tests in different experimental scenarios show that the defense mechanism can effectively prevent SDN fingerprint attacks, and the defense overhead is within a reasonable range.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was partly supported by the National Key Research and Development Program of China (no. 2018YFB0804004) and the National Natural Science Fund (nos. 62072467, 61521003, and 62002383).

References

[1] N. Mckeown, T. Anderson, H. Balakrishnan et al., "OpenFlow," *Acm Sigcomm Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] J. C. C. Chica, J. C. Imbachi, and J. F. Botero, "Security in SDN: a comprehensive survey," *Journal of Network and Computer Applications*, vol. 159, p. 102595, 2020.

[3] T. Han, S. R. U. Jan, Z. Tan et al., "A comprehensive survey of security threats and their mitigation techniques for next-generation SDN controllers," *Concurrency and Computation Practice and Experience*, vol. e5300, pp. 1–23, 2019.

[4] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, 2005.

[5] P. Auffret, "SinFP, unification of active and passive operating system fingerprinting," *Journal in Computer Virology*, vol. 6, no. 3, pp. 197–205, 2010.

[6] M. Smart, G. R. Malan, and F. Jahanian, "Defeating TCP/IP stack fingerprinting," in *Proceedings of the Usenix Security Symposium*, Denver, CO, USA, August 2000.

[7] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *Proceedings of the International Workshop on Recent Advances in Intrusion Detection*, Springer, Menlo Park, CA, USA, September 2011.

[8] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the Workshop on Hot Topics in Software Defined Networks*, ACM, Helsinki, Finland, August 2012.

[9] S. Shin and G. Gu, "Attacking software-defined networks: a first feasibility study," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in softWare Defined Networking*, ACM, Hong Kong, China, August 2013.

[10] R. Bifulco, H. Cui, G. O. Karame et al., "Fingerprinting software-defined networks," in *Proceedings of the IEEE International Conference on Network Protocols*, Singapore, November 2016.

[11] A. Azzouni, O. Braham, N. T. M. Trang et al., "Fingerprinting OpenFlow controllers: the first step to attack an SDN control plane," in *Proceedings of the Global Communications Conference*, IEEE, Singapore, December 2017.

[12] J. Sonchack, A. J. Aviv, and E. Keller, "Timing SDN control planes to infer network configurations," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pp. 19–22, ACM, Scottsdale, AZ, USA, March 2016.

[13] M. Zhang, J. Hou, Z. Zhang et al., "Fine-grained fingerprinting threats to software-defined networks," in *Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS*, IEEE Computer Society, Sydney, Australia, August 2017.

[14] J. Leng, Y. Zhou, J. Zhang et al., "An inference attack model for flow table capacity and usage: exploiting the vulnerability of flow table overflow in software-defined network," https://arxiv.org/abs/1504.03095, 2015.

[15] A. Bilal and A. Nadeem, "Fingerprinting SDN policy parameters: an empirical study," *IEEE Access*, vol. 8, pp. 142379–142392, 2020.

[16] J. Hou, M. Zhang, Z. Zhang, W. Shi, B. Qin, and B. Liang, "On the fine-grained fingerprinting threat to software-defined networks," *Future Generation Computer Systems*, vol. 107, no. 107, pp. 485–497, 2020.

[17] J. Cao, Z. Yang, and K. Sun, "Fingerprinting SDN applications via encrypted control traffic," in *Proceedings of the USENIX Association 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pp. 501–515, Beijing, China, September 2019.

[18] S. Shin, V. Yegneswaran, P. Porras et al., "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*, ACM, Berlin, Germany, November 2013.

[19] S. Hong, L. Xu, H. Wang et al., "Poisoning network visibility in software-defined networks: new attacks and countermeasures," in *Proceedings of the Network & Distributed Security Symposium*, San Diego, CA, USA, February 2015.

[20] S. Shin, P. Porras, V. Yegneswaran et al., "FRESCO: modular composable security services for software defined networks," in *Proceedings of the Network & Distributed Security Symposium*, San Diego, CA, USA, August 2013.

[21] M. Dhawan, R. Poddar, K. Mahajan et al., "SPHINX: detecting security attacks in software-defined networks," in *Proceedings of the Network & Distributed Security Symposium*, San Diego, CA, USA, February 2015.

[22] P. Porras, S. Cheung, M. Fong et al., "Securing the software-defined network control layer," in *Proceedings of the Network & Distributed Security Symposium*, San Diego, CA, USA, February 2015.

[23] G. O. Karame, B. Danev, C. Bannwart, and S. Capkun, "On the security of end-to-end measurements based on packet-pair dispersions," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 149–162, 2013.

[24] R. J. Fox and M. W. Dimmic, "A two-sample Bayesian t-test for microarray data," *BMC Bioinformatics*, vol. 7, no. 1, p. 126, 2006.

[25] S. Jajodia, A. K. Ghosh, V. Swarup et al., *Moving Target Defense*, Springer, Berlin, Germany, 2011.

[26] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, 2011.

[27] K. P. . Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press, Cambridge, MA, USA, 2012.

[28] A. Wang, A. Mohaisen, W. Chang et al., "Delving into internet DDoS attacks by botnets: characterization and analysis," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems & Networks*, IEEE, Toulouse, France, December 2015.