Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# On the fine-grained fingerprinting threat to software-defined networks

Jianwei Hou, Minjian Zhang, Ziqi Zhang, Wenchang Shi *, Bo Qin, Bin Liang

*School of Information, Renmin University of China, Beijing, PR China*

A R T I C L E I N F O

*Article history:*
Received 15 March 2019
Received in revised form 3 January 2020
Accepted 25 January 2020
Available online 17 February 2020

*Keywords:*
SDN
Fingerprinting
Timing attacks
Information disclosure
Mitigation strategies

A B S T R A C T

Software-defined networking (SDN) is an emerging networking technology, which has attracted wide attention from academia and industry, playing a key role in enabling techniques of the 5th generation wireless systems (5G). The fundamental characteristic of SDN is that it decouples the control plane from the data plane, which can provide flexibility and programmability for 5G. Unfortunately, the separation of the two planes becomes a potential attack surface as well, which enables adversaries to fingerprint and attack the SDNs. Existing work showed the possibility of fingerprinting an SDN with time-based features. However, they are coarse-grained. This paper proposes a fine-grained fingerprinting approach that reveals the much more severe threats to SDN security and explores the mitigation strategies. By analyzing network packets, the approach can dig out sensitive and control-related information, i.e., match fields of SDN flow rules. The match fields of flow rules can be used to infer the type of an SDN controller and the security policy of an SDN network. With sensitive configuration information, adversaries can launch more targeted and destructive attacks against an SDN. We implement our approach in both simulated and physical environments with different kinds of SDN controllers to verify the effectiveness of our concept. Experimental results demonstrate the feasibility to obtain fine-grained and highly sensitive information in SDN, and hence reveal the high risk of information disclosure in SDN and severe threats of attacks against SDN. To mitigate the fine-grained fingerprinting threat we have revealed, we explore a lightweight countermeasure trying to hide the sensitive time-based features of SDN networks. Implementation and evaluation demonstrate that our countermeasure can play a role in mitigating the risk of SDN control information leakage with only minor overheads.

## 1. Introduction

Software-Defined Networking (SDN), an emerging network architecture, consists of application layer, control layer and infrastructure layer. It decouples controlling and forwarding functions of the network, introducing a more flexible way to manage network traffic with high programmability [1]. Therefore, SDN has become an enabler for the network evolution of the 5th generation wireless systems (5G) [2]. Meanwhile, the security issues of SDN raise more and more attention. There are studies analyzing and summarizing the security threats to SDN from different aspects [3–5]. The threat to control plane is the most striking. In SDN, the network status and control information are managed and issued by the control plane to ensure centralized network control, which opens a door for attackers. The key issue we concern about is whether it is possible for adversaries to get sensitive and fine-grained information about the SDN configurations, and to what extent they may reach?

Existing work has revealed the possibility to remotely confirm whether a target network is an SDN, which uses a time-based method for fingerprinting [6]. Based on the fingerprinting results, a targeted DoS attack can be launched. With the knowledge that a target network is an SDN, Cui et al. [7] choose two features, RTT and packet-pair dispersion of the exchanged packets, to fingerprint the SDN based on massive experimental data. Their work demonstrates high accuracy in deducing whether a given packet triggers the installation of a flow rule. Parallel research efforts try to infer whether the certain flow triggers the interaction between the data and control planes based on the hypothesis that the adversary is inside the network [8,9]. Based on their fingerprinting results, attackers are able to infer the type of existing rules (being aggregated rules or not [8]) and some network configurations [9].

Previous work has demonstrated the possibility of fingerprinting the SDN based on its characteristic of centralized control. However, most of their work is limited to determining whether a certain packet triggers the installation of a flow rule, and the information collected in this way is coarse-grained. What we want to discuss further is whether adversaries are able to fingerprint an SDN in a more fine-grained way, exposing the target network to a more severe threat.

* Corresponding author.
*E-mail address:* wenchang@ruc.edu.cn (W. Shi).

https://doi.org/10.1016/j.future.2020.01.046
0167-739X/© 2020 Elsevier B.V. All rights reserved.

In this paper, we explore a new way to fingerprint match fields of flow rules in an SDN, which is fine-grained and highly sensitive information of an SDN. Adversaries can infer controller's type, security policies and even control logic of an SDN from the match field information. We implement our design in different environments with diverse controllers to demonstrate the generality of such threat to SDNs. Furthermore, most existing work evaluates their methods either in a simulated or a physical environment, while we carry out our experiments in both two environments and illustrate the difference in implementing the same design. Moreover, we present a lightweight mitigation mechanism to defend SDN networks against the fingerprinting attacks without additional modifications in the data plane. The experimental result demonstrates that the mitigation mechanism can considerably increase the difficulty for adversaries to fingerprint SDN networks, which can play a role in mitigating the fingerprinting threats with minor overheads. The main contributions of this work can be summarized as follows:

- We reveal the risk of the control information disclosure rooted in the SDN architecture and raise the concern of the possibility for an attacker to obtain fine-grained knowledge about an SDN network.
- We present a fine-grained fingerprinting method that it can learn the match fields of flow rules by distinguishing the transmission delays of different packets. We implement our method in both simulated and physical testbeds with four popular controllers to evaluate the feasibility and generality of the method. The differences in design and effects between the simulated and physical environments are also analyzed.
- We analyze the common characteristics of time-based fingerprinting attacks in SDN networks and propose a lightweight mitigation mechanism without additional modifications on the switches, trying to hide the time-based features of the SDN network. Experimental results show that our countermeasure can efficiently mitigate the risk of leakage of control information with minor impact on network performance.

The rest of the paper is organized as follows. We present our research background in Section 2. In Section 3, we bring out the design principles of our fingerprinting method for the match fields of flow rules. We implement our method in simulated and physical environments respectively. The results of the experiments are presented and analyzed in Sections 4 and 5. In Section 6, we explore a possible solution to mitigate the risk of SDN control information disclosure. We evaluate the effectiveness of our mitigation strategies as well as its impact on network performance. In Section 7, we discuss related work, then we conclude in Section 8.

## 2. Background

### 2.1. Motivation

When an adversary is going to attack an SDN network, what information does he/she most want to know?

*Controller type.* The controller plays a key role in SDN environment with centralized control functions. To grab a stronger control of a network, most attackers tend to invade the control plane. Having knowledge of the type/version of a running controller, adversaries can exploit existing vulnerabilities or leverage the knowledge of control logic to launch more targeted attacks. Our observation shows that different controllers usually use different default match fields in flow rules. If adversaries grasp the information about match fields, they can infer the type of target controller easily.

*Security policy.* In most cases, a security policy will be set to protect a network from attacks. Adversaries who are aware of security policy are more likely to bypass security checks to attack. In an SDN, the security policy is defined by the control layer and applications, then delivered to network devices in the form of flow rules. Network devices process data flows based on the installed flow rules. Therefore, it is possible to infer a policy defined by an upper layer according to flow rules.

*Malicious traffic.* Because a controller is the core of an SDN, network performance would be greatly compromised if a controller suffers from DoS attacks. Attackers are eager to know what kind of data flows would increase the load of a controller. According to the control logic of SDN, mismatched network packets will be forwarded to control plane for decision, and the controller will deliver control instructions to switches to process these packets. With the knowledge of match fields of flow rules, adversaries can craft the packets that mismatch the existing flow rules by randomly forging some or all fields. These table-miss packets will trigger heavy traffic from the switch to the controller, consuming the communication bandwidth. In addition, as the controller optionally installs flow rules on switches to handle the new data flow, the crafted packets may cause plenty of rule installation, which would make flow tables overflow.

In short, flow rules are of great value as they reflect the control logic of an SDN network. As an important part of flow rules, match fields are treated as fine-grained and highly sensitive information in an SDN environment. To our knowledge, our work is the first attempt to fingerprint match fields of flow rules without knowing network configurations (e.g. controller type) in advance. In this paper, we set up simulated and physical environments for evaluation, and use four different types of controllers to verify the feasibility and generality of our method. The results demonstrate the difference between simulated and physical environments and we also optimize the means for analysis.

### 2.2. SDN/OpenFlow

Open Networking Foundation (ONF) separates SDN architecture into three layers: application plane, control plane, and infrastructure plane [10]. Application plane manages the network with open interfaces provided by control plane to implement complex control logic. Infrastructure plane is composed of network devices that are only responsible for executing forwarding/dropping actions. Network devices are deprived of control functions while control plane is responsible for maintaining network status information and configurations. To instruct the actions of network devices, control plane delivers control information to infrastructure plane through southbound interface. Due to such architecture, network management in SDN becomes flexible that network administrators can easily deploy and manage the network by programming an SDN application. In general SDN architecture, there are various means to implement southbound interface. Among all those implementations, OpenFlow is the ONF-recommended and most widely accepted one. In this paper, we discuss the SDN that uses OpenFlow.

OpenFlow was firstly proposed by McKeown et al. [11]. OpenFlow specifies that each OpenFlow switch retains one or more flow tables. Each flow entry in the flow table is associated with an action that instructs how to process a certain flow. The flow entries can be defined through a secure channel remotely and the secure channel connects switches with a remote control process (i.e., controller). Obviously, the flow entries in flow tables become the most important control information because it can reflect the control logic.

**Table 1**
Fields used to match with flow entries in OpenFlow 1.0.

| Ingress port | Src MAC | Dst MAC | Ether type | VLAN ID | VLAN priority | Src IP | Dst IP | IP proto | IP ToS | Src TCP/UDP port | Dst TCP/UDP port |
|---|---|---|---|---|---|---|---|---|---|---|---|

### 2.3. Flow rules and match fields

In an OpenFlow switch, a flow table contains many flow entries. Each flow entry is a forwarding rule for a certain flow, so a flow entry is also called a flow rule. OpenFlow switches handle flows based on flow rules. One flow rule consists of three parts: header fields, counter and action. Header fields play a fundamental role in matching. As listed in Table 1, OpenFlow 1.0 uses 12-tuple header fields [12]. Follow-on versions of OpenFlow protocols support the extensible header fields, such as the IPv6 extension headers. In this paper, we focus on the basic header fields defined in OpenFlow 1.0, which are used in all versions of OpenFlow. When a packet comes, the switch first sets ingress port and identifies Ethernet type, then adds VLAN ID and other information according to the Ethernet type. After parsing and extracting packet's header, the switch compares the packet's header with the rule set to find the rule that matches the packet. Once matched, the switch executes the action defined in this flow rule. Usually, in a flow rule, not all the fields need exact matching. Some fields may be set to ANY, which means the field can match any value, so values of these fields will not influence the matching results. If different values of a field lead to different matching results, this field is an exact matching field. Different SDN networks will set different header fields as exact matching fields, which may reflect the control logic of the network. In the paper, we call an exact matching field as a match field in short.

Flow rules contain sensitive control information of the network. As a primary part of flow rules, match fields should be well protected. The disclosure of match fields will expose the SDN to great danger.

## 3. Fingerprinting scheme design

Before introducing how to fingerprint the match fields, we first illustrate the packet processing procedure in SDN. Based on the analysis of the essential characteristics of SDN, we then present our method of obtaining match field information in detail.

### 3.1. Packet processing in SDN

According to the definition in [11], each flow entry is associated with a simple action to instruct the OpenFlow switch to handle the matched packets. There are three basic types of actions (as shown in Fig. 1):

- Forward directly: the switch forward the packet to a specified port. (Action-1)
- Forward to the controller: the switch packs the packet and sends it to the controller. The controller analyzes the packet and installs a flow rule on the switch optionally. (Action-2)
- Drop: for security or other reasons, the switch drops the packet. (Action-3)

Packets of the same flow have the same value in every header fields. When the switch takes different actions to deal with the same flow's packets (the value of the header fields are the same), this field is a match field. (It indicates that the first packet triggers an interaction between the controller and the switch while the following packets are directly forwarded by the switch). When the value of the packet's header field changes and the switch take different actions to the packet, this field is also a match field. (It indicates that different values of this field result in different matching results).

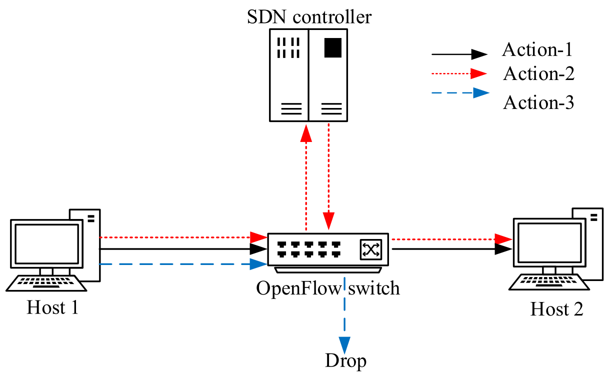

**Fig. 1.** Three types of actions for packet processing in SDN.

### 3.2. Fingerprinting scheme

Based on the analysis presented in 3.1, we consider the following scenarios. We suppose that there are two hosts in the same SDN network and one host acts as Sender to send packets to the other (as Receiver).

(1) Receiver can receive the packet

- Case-1: The switch has already installed the corresponding flow rule, therefore, the packet matches the rule successfully and the switch forwards this packet directly. In this case, the switch performs the Action-1.
- Case-2: The switch has not found a matched entry, so it will send this packet to the controller for a decision. The controller analyzes the packet and installs a corresponding rule on the switch to handle this flow. In this case, the switch performs the Action-2.
- Case-3: The switch has not found the corresponding flow rule, so it interacts with the controller. The controller instructs the switch to forward the packet instead of issuing the relevant rule. The switch performs the Action-2 in this case.

The three cases are shown in Fig. 2. In Case-2 and Case-3, the time for a packet to travel from Sender to Receiver is significantly longer than Case-1, as the packet triggers an interaction between the controller and the switch.

(2) Receiver cannot receive the packet

- Case-a: There is no matched entry in the switch's flow table. The switch will ask the controller for a decision to handle the packet. If the controller does not deliver forwarding instructions to the switch, or delivers a Drop rule to the switch, the packet will be dropped. Here, the switch performs the Action-2.
- Case-b: The switch drops the packet according to the corresponding rule installed on the switch. The switch performs the Action-3 in this case.
- Case-c: The packet gets lost due to network congestion or other occasional reasons.

The dataflow of these three cases when Receiver cannot receive the packet are shown in Fig. 3.

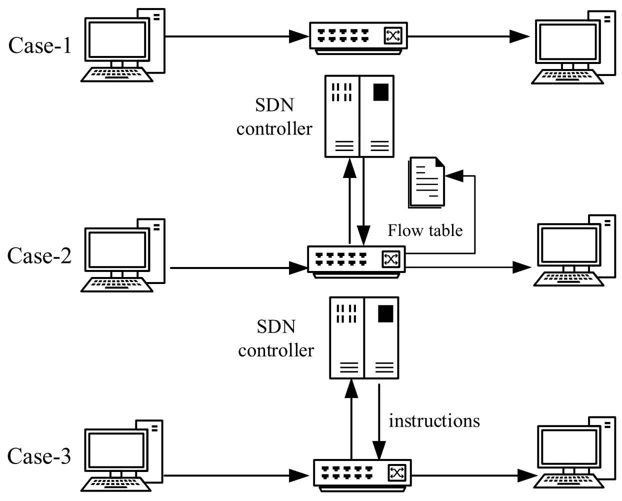

**Fig. 2.** Three cases of packet processing when Receiver can receive the packet.

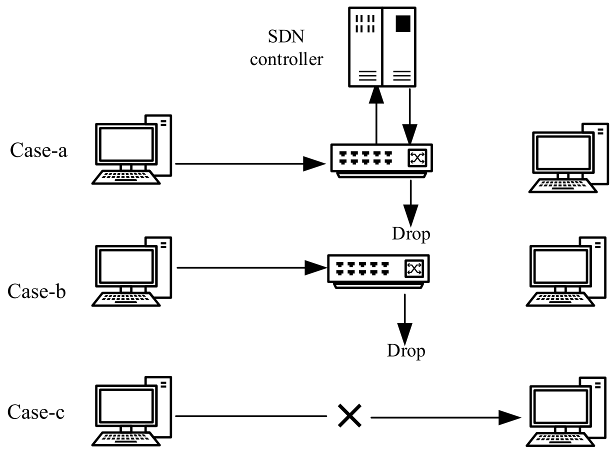

**Fig. 3.** Three cases of packet processing when Receiver cannot receive the packet.

We design a scheme to fingerprint the match fields in an SDN network based on the above analysis. Supposing that Sender and Receiver are in the same network (LAN) and are able to communicate with each other normally ignoring the occasional Case-c, we try to determine whether a header field is a match field.

For a given header field we send a number of packets with the same header field value at a certain interval.

- On condition that Receiver receives all the packets, we can determine whether this header field is a match field by analyzing the length of time between sending and receiving. If there is a measurable difference between the time duration of Case-1 and the time duration of Case-2 and Case-3, we can infer the switch has taken different actions to the packets of the same flow. So this header field is a match field.
- On condition that all the packets fail to reach Receiver, we can deduce that the switch has taken different actions to the different flows with different values in the given field. This header field is a match field, because the pair of hosts can communicate with each other normally before sending this set of test packets.
- On condition that Receiver receives some of the packets, we can infer that the switch handles the same flow with different actions. It may happen due to flow rule expiration. This rarely happens in theory if we do not consider the occasional factors. But this situation also shows that the change of the field's values will lead to different matching results, so that we can infer it is a match field.

Multi-group repeated experiments will help us to rule out occasional cases and get a more accurate result. Based on the principle introduced above, we have implemented the method of fingerprinting match fields, and have conducted experiments in both simulated and physical environments.

## 4. Implementation and evaluation in simulated testbed

### *4.1. Implementation*

Based on the analysis in Section 3, we choose two features as measurements of our fingerprinting method: (1) whether the target host has received the packets successfully, and (2) the latency of packet transmission from Sender to Receiver. To implement the method, it should be ensured that there are two hosts in an SDN network, and they are able to communicate with each other as Sender and Receiver, respectively. We present the fingerprinting process as follows.

- Step-1: Select a header field, *MF*, to be tested, and define $N$ to represent the number of samples.
- Step-2: Sender crafts the forged packet $P$ with a legal value of *MF* randomly, and keeps values of other fields the same as before. (In practice, we exclude some special values when choosing the value of *MF*, e.g., the broadcast IP address and frequently used ports.) Sender transmits the forged packet $P$ to Receiver and records the sent time. Receiver records the receipt time of $P$ when receiving it. We use $t_1$ to represent the time lag between sending and receiving. After a one-second delay, Sender sends the packet $P$ to Receiver again, and we use $t_2$ to represent the time lag between sending and receiving (if Receiver does not receive both packets, we will not record any time information).
- Step-3: Repeat Step-2 $N$ times, after which we have $T_1$, a sample set of the values of $t_1$, and $T_2$ of $t_2$.
- Step-4: Process and analyze $T_1$ and $T_2$ to infer whether the tested header field *MF* is a match field based on features (1) and (2).

In our design, Sender sends UDP-based probe packets and utilize Scapy [13] to modify the value of the header field and craft the packet. We record the sent time and the receipt time using the timestamps provided by TCPDUMP.

### *4.2. Testbed*

Using Mininet [14] to simulate the underlying network, we deploy the controller and the underlying network on a single physical machine. We choose four different controllers: Ryu [15], Floodlight [16] and two versions of OpenDaylight [17]. All the simulated SDN environments use the OpenFlow 1.0 protocol. The header fields of OpenFlow 1.0 protocol are the basic fields that are supported by all follow-on versions. The subsequent protocols have begun to support the extension of header field defined by the third-party since OpenFlow 1.2 protocol. Therefore, in view of the generality of the fingerprinting method, we focus on the header fields defined in OpenFlow 1.0, using OpenFlow 1.0 protocol in our tests. The details of the testbed are shown in Table 2. There are two OpenFlow switches connecting with each other in the testbed. Host $h_1$ connects to switch $s_1$. Host $h_2$ and $h_3$ connect to switch $s_2$. Note that $h_1$ and $h_2$ are malicious hosts controlled by the adversary. Two hosts are in the same LAN and can communicate normally.

**Table 2**
Configurations of simulated and physical testbeds.

| | Simulated testbed 1 | Simulated testbed 2 | Physical testbed |
|---|---|---|---|
| CPU | Intel(R) Core(TM) 2 Duo CPU E4400 @ 2.00 GHz Duo core CPU | Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz 4-core CPU | Intel(R) Xeon(R) CPU E5506 @ 2.13 GHz 4 cores |
| Memory | 5.8 GB | 7.7 GB | 3.8 GB |
| OS | 3.13.0 kernel distribution, Ubuntu 14.04 LTS | 3.13.0 kernel distribution, Ubuntu 14.04 LTS | 3.13.0 kernel distribution, Ubuntu 14.04 LTS |
| Controller | RYU-4.10, OpenDaylight H (0.1.1), Floodlight-1.2 | OpenDaylight Be (0.4.2) | RYU-4.10, Floodlight-1.2, OpenDaylight H (0.1.1), OpenDaylight Be (0.4.2) |
| Other | Mininet-2.3.0, Open vSwitch-2.0.2 | Mininet-2.3.0, Open vSwitch-2.0.2 | Mininet-2.3.0, Open vSwitch-2.0.2 |

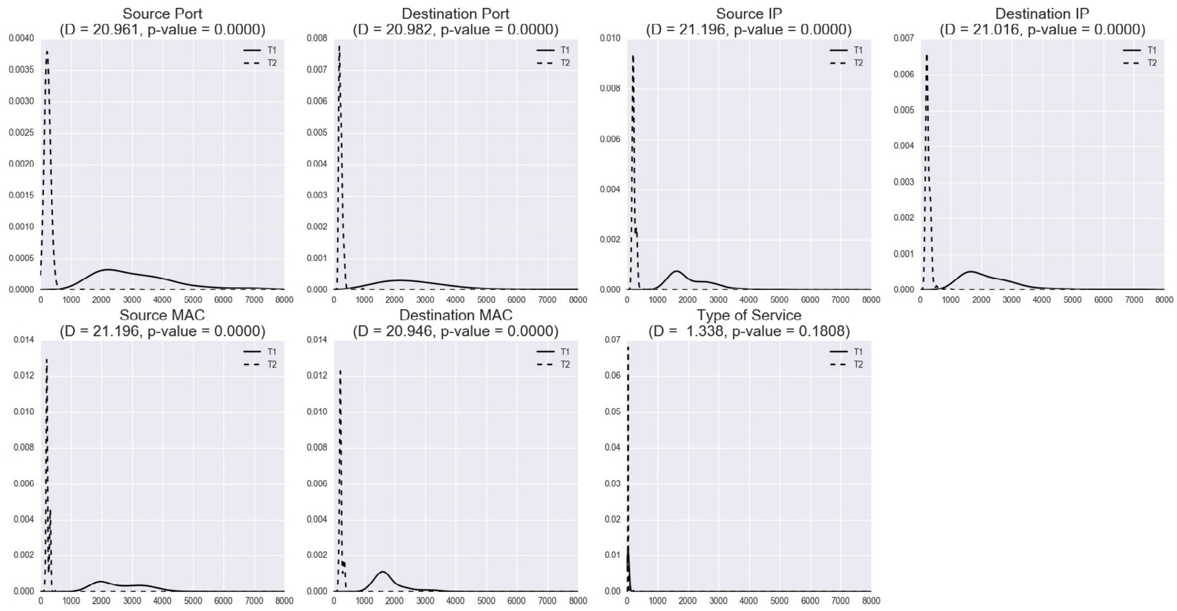

**Fig. 4.** PDFs of $T_1$ (solid line) and $T_2$ (dotted line) for each measured field, in the simulated SDN with Floodlight controller. The unit of the x axis is μs.

## 4.3. Results and evaluation

We have implemented the method described above to fingerprint the match fields of the flow rules on the testbeds with four popular controllers. Considering that some of the match fields (e.g., IP ToS) defined in OpenFlow 1.0 only have several legal values, we mainly focus on the commonly used fields with wide value range, including src/dst port, src/dst IP, src/dst MAC. Besides, we also fingerprint ToS field and analyze the values of this field as the control group. For each field to be tested, we randomly choose 300 different values from its legal value span, that is, we define $N$ as 300.

We compute the probability density function (PDF) of the measured values of $T_1$ and $T_2$. The PDFs of $T_1$ and $T_2$ are shown in Fig. 4 (Floodlight as an example). In Fig. 4, the x axis represents the time duration while the solid line represents the PDF of $T_1$ and the dotted line represents the PDF of $T_2$. The area under the density function above the x axis represents the probability over a specific time range. In statistics, the method of hypothesis testing is usually used to determine whether the two statistics are significantly different. When the distribution of random variables is unknown, Wilcoxon rank-sum test [18] is usually used. Therefore, we apply a rank-sum test to $T_1$ and $T_2$ for every measured field. Note that, if all the test packets of certain measured field fail to reach Receiver, the PDF figure remains blank with no Wilcoxon rank-sum test results.

It is intuitive to show the overlap ratio of $T_1$ and $T_2$ in the PDFs figure. It is hard to distinguish two samples with high overlap ratio, which means that there is no significant difference between the two samples. We show the $P$-value produced from the rank-sum test for each measured field in Table 3.

According to hypothesis testing theory, when the $P$-value is below 0.05, the null hypothesis is rejected, so $T_1$ is significantly different from $T_2$. Combining the results shown in Table 3 with the reception of packets, we can infer whether the test field is a match field. In Table 4, we list the inferred match fields based on our fingerprinting and the actual match fields in use. The underlined fields are overlooked in our fingerprinting, but they are match fields indeed. The main reason for the false negative is that the legal value of these fields is limited and we are not able to randomize the value of these fields.

The field in bold is a false positive. In the experiment with OpenDaylight H, the main cause of misjudging the MAC field as a match field is that there is a little nuance between the sample distributions of $T_1$ and $T_2$. Wilcoxon rank-sum test is demanding, so the result shows $T_1$ and $T_2$ are remarkably different.

The result demonstrates that our fingerprinting method has a high accuracy in inferring match fields. Values of match fields can be randomized to conduct further attacks.

## 5. Implementation and evaluation in physical testbed

### 5.1. Implementation

The basic principle of implementation in a physical environment is roughly the same as that in simulated environment. And

**Table 3**
P-Value from rank-sum test on each field's T1 and T2.

| | Src port | Dst port | Src IP | Dst IP | Src MAC | Dst MAC | ToS |
|---|---|---|---|---|---|---|---|
| RYU-4.10 | 0.9675 | 0.9998 | 0.6378 | 0.7892 | 0.9885 | 0.0000 | 0.4129 |
| Floodlight-1.2 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.1808 |
| OpenDaylight H | 0.6637 | 0.9431 | 0.4755 | – | 0.3143 | 0.0000 | 03657 |
| OpenDaylight Be | 0.3891 | 0.6517 | 0.6931 | 0.9172 | – | – | 0.9870 |

we use the same features (1), (2) mentioned in 4.1 as measurements. But in practical situation, it is hard to ensure the synchronization of different hosts. In spite of using Network Time Protocol (NTP) to synchronize time, the accuracy is limited. And the system time of different machines is prone to bias as time goes on. With the consideration of synchronization, we adjust the implementation described in Section 4.1 as follow.

- Step-1: Select a header field, $MF$, to be tested, and define $N$ to represent the number of samples.
- Step-2: Sender randomly selects a legal value from the range of $MF$, and keeps the values of other fields the same as before. Then it transmits the forged packet $P$ to Receiver and records the sent time. Receiver records the receipt time of $P$ when receiving it. We define $t_1$ as the time lag between two timestamps. Sender repeats sending the same packet P for three times. Then we compute three time lags, $t_2$, $t_3$ and $t_4$ respectively. Every packet is sent at a 1-second interval. After getting $t_1 \sim t_4$, we compute $\Delta t_1 = t_1 – t_2$, $\Delta t_2 = t_3 – t_4$.
- Step-3: Repeat the Step-2 $N$ times, after which we have a set of $\Delta t_1$ values, represented as $TC_1$, and a set of $\Delta t_2$ values as $TC_2$.
- Step-4: Process and analyze $TC_1$ and $TC_2$, and infer whether the tested header field $MF$ is a match field based on features (1) and (2).

Similarly, we use UDP packets in our experiments. Sender modifies header fields and crafts packets with Scapy. Both Sender and Receiver use timestamps from TCPDUMP to represent sent and receipt time.

## 5.2. Testbed

A four-host physical testbed is set up for evaluation. One machine is installed with the SDN controller, and one with Open vSwitch [19] as an OpenFlow switch. The other two machines act as malicious hosts in the SDN, as Sender and Receiver, and they connect to the same OpenFlow switch. In evaluation, we use four different controllers to set up testbed and also use OpenFlow 1.0 protocol in the tests. Detailed information about the four physical machines is listed in Table 2. The same as that in simulated case, $h_1$ and $h_2$ are malicious hosts controlled by the adversary. They are in the same LAN and can communicate with each other.

## 5.3. Results and evaluation

Similar to our evaluation in Section 4, we fingerprint the match fields of flow rules under diverse controllers. The measured fields are: src/dst port, src/dst IP, src/dst MAC. Also, we use the IP ToS field as the control group. For each test field, we randomly choose 500 different values from its legal value span, that is, we define $N$ as 500.

*Statistical analysis.* Because different variables are used in physical testbed evaluation, we adjust the statistical analysis method. We treat $t_1 \sim t_4$ as four different variables. We define another two variables $T_1$ and $T_2$ to represent $t_1$-$t_2$ and $t_3$-$t_4$ respectively. $T_1$ and $T_2$ are used to exclude the effect of time deviation of different systems. The data sets $TC_1$ and $TC_2$ defined in 5.1 are samples of $T_1$ and $T_2$.

**Table 4**
Comparison of inferred match fields with actual match fields in simulated testbed.

| | Inferred match fields | Actual match fields |
|---|---|---|
| RYU-4.10 | Dst MAC | Dst MAC<br>Ingress Port |
| Floodlight-1.2 | Src/Dst Port<br>Src/Dst IP<br>Src/Dst MAC | Src/Dst Port<br>Src/Dst IP<br>Src/Dst MAC<br>Ingress Port<br>EtherType<br>IP Protocol |
| OpenDaylight H | Dst IP<br>**Dst MAC** | Dst IP<br>EtherType |
| OpenDaylight Be | Src/Dst MAC<br>Dst MAC | Src/Dst MAC<br>EtherType |

As we know, in an SDN, if the controller installs rules for a new flow, only the first one of the four test packets will trigger the switch-controller interaction, and the subsequent packets would be forwarded by the switch directly. In this case, $t_1$ is significantly longer than $t_2$, $t_3$ and $t_4$. We record the system deviation time of Sender and Receiver as $t_d$, the transmission delay as $t_t$, and the latency of switch-controller interaction as $t_i$. In theory, $t_1 \sim t_4$ can be presented as follow.

$$t_1 = t_d + t_t + t_i$$
$$t_2 = t_3 = t_4 = t_d + t_t$$

In theory, $T_1 = t_1 - t_2 = t_i$, $T_2 = t_3 - t_4 = 0$. But in practice, every parameter mentioned above is a random variable, the value of which is not fixed but obeys a certain distribution. So samples of $T_1$ and $T_2$ distribute around the theoretical values with a little nuance. However, this little nuance is negligible, compared with the delay caused by switch-controller interaction. Therefore, we can infer whether the switch-controller interaction happens based on the difference between $T_1$ and $T_2$.

Firstly, we apply Gauss kernel density estimation to $TC_1$ and $TC_2$. We compute and draw the PDFs of $T_1$ and $T_2$. It is intuitive to observe the difference and overlap ratio of value distributions. We take the fingerprinting result of Floodlight as an example and show it in Fig. 5. In Fig. 5, the x axis represents the time duration while the solid line represents the PDF of $T_1$ and the dotted line represents the PDF of $T_2$. The area under the density function above the x axis represents the probability over a specific time lag range.

Secondly, we quantify the discrepancy between $T_1$ and $T_2$. In the evaluation of simulated experiments, we use a hypotheses testing method to evaluate the difference between two variables. However, this method is inapplicable to the evaluation of the experiments in physical testbed. $T_1$ and $T_2$ are statistics obtained from the difference between two random variables. When the difference is slight in practice, the value ranges of $T_1$ and $T_2$ are small. So any little nuance of the mean value (e.g. 100 $\mu$s) will influence the rank-sum test results largely. However, in practical environment, little deviation caused by devices or network status is common and unavoidable. This little nuance cannot indicate significant difference between $T_1$ and $T_2$. Considering that this nuance is not a fixed value, we cannot simply minus it from each

**Table 5**
Probability P of each measured field.

| | Src port | Dst port | Src IP | Dst IP | Src MAC | Dst MAC | ToS |
|---|---|---|---|---|---|---|---|
| RYU-4.10 | 0.9980 | 1.0000 | 0.9880 | 1.0000 | 0.9920 | 0.0000 | 1.0000 |
| Floodlight-1.2 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.9980 |
| OpenDaylight H | 1.0000 | 0.9714 | 0.8855 | – | 0.9198 | 0.9953 | 0.9980 |
| OpenDaylight Be | 1.0000 | 1.0000 | 0.9837 | 1.0000 | – | – | 1.0000 |

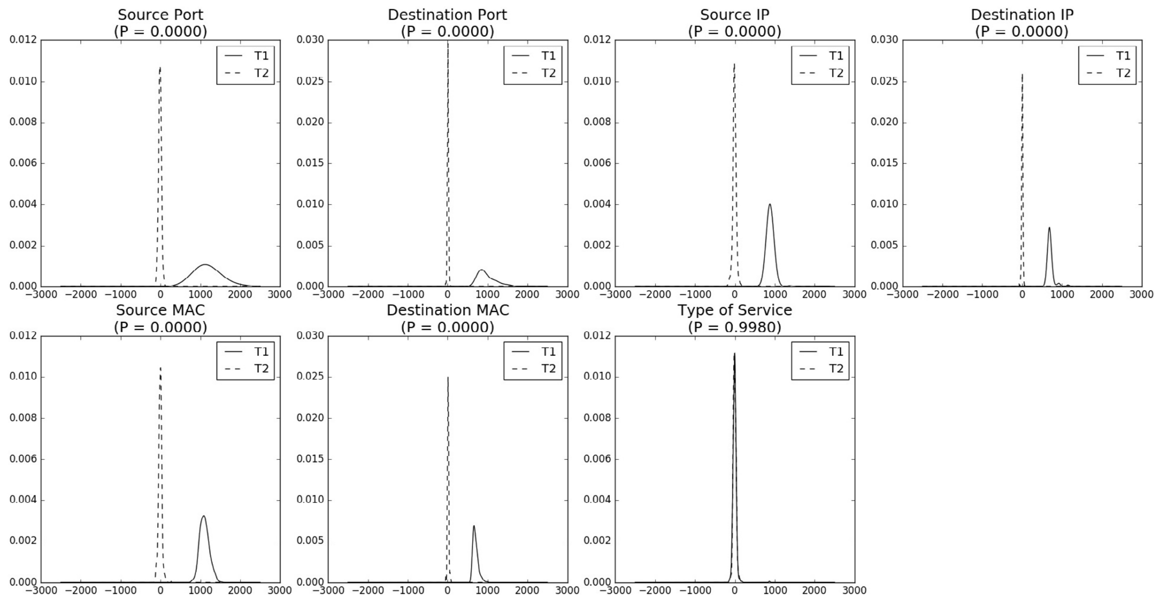

**Fig. 5.** PDFs of $T_1$ (solid line) and $T_2$ (dotted line) for each measured field, in the SDN with Floodlight controller. The unit of the x axis is μs.

measurement. Due to this situation, we adopt another analysis approach instead of traditional hypotheses testing methods.

Before analysis, an acceptable range of deviation $\Delta$x should be defined. Then, we calculate the mean of variable $T_2$ as $\mu$. Finally, we compute the probability P of $T_1$ distributing in range ($\mu - \Delta$x, $\mu + \Delta$x). A large P value means that $T_1$ distributes around the mean value of $T_2$ with high probability. That is, these two variables are not significantly different from each other in an acceptable range. The deviation $\Delta$x is an empirical value that differs according to the experimental environments. We set $\Delta$x as 200 μs in our experiment, as the RTT of pings between Sender and Receiver is around 400 μs. We summarize the P values of different fields in Table 5.

Under the consideration of the environmental effects, when P is close to zero, we conclude that $T_1$ is significantly different from $T_2$. So we infer that this test header field is a match field. In Table 6, we compare the inferred match fields obtained by our fingerprinting method with the actual match fields in use. The underlined fields are overlooked in our fingerprinting, but they are match fields indeed. The main reason for the false negative is same as what we mentioned in simulated environment. That is, the legal value of these fields is limited and we are not able to randomize the value of these fields.

It should be noted that the false positive disappears due to our adjusted analysis approach. Thus, analysis that takes deviation into account is more accordant with practical situations. The result indicates that our fingerprinting method has a high accuracy in the physical environment.

The information obtained from fingerprinting is useful for adversaries to launch a dedicated DoS attacks. Based on the match field information, an adversary can flood the network by sending a large number of table-miss packets. These table-miss packets will trigger interactions between the controller and the switch, consuming the communication bandwidth, CPU computation, and memory in both control and data planes.

## 6. A lightweight countermeasure

To mitigate the risk of SDN control information disclosure, we first analyze the common characteristics of time-based fingerprinting attacks in SDN networks and explore a lightweight countermeasure to defend against the fingerprinting attacks. We evaluate the performance and overhead of our proposal in a simulated testbed.

### 6.1. Analysis of the characteristics of SDN fingerprinting attacks

The threat of SDN control information leakage originates from the centralized control function, which is rooted in SDN architecture. To explore a general countermeasure to mitigate the time-based fingerprinting threats, we first analyze the common characteristics of the fingerprinting attacks in SDN networks. The characteristics are discussed in the following.

(1) Changes in the value of header fields. If a flow rule is installed, the packets of the same flow will be forwarded directly. In this situation, there is no significant difference between transmission time of packets in one flow. Therefore, to collect different time data, the fingerprinting attacks need to craft the packets with different values of header fields.
(2) Some fingerprinting packets must pass by the controller. Adversaries need to collect the transmission time of the packets that are directly forwarded by the switches and the transmission time of the packets that are forwarded to the controller. Therefore, they need to craft some packets that pass by the controller.

Therefore, to defend against fingerprinting attacks, one possible countermeasure is to add transmission delay to all the subsequent packets to reduce the difference of transmission time

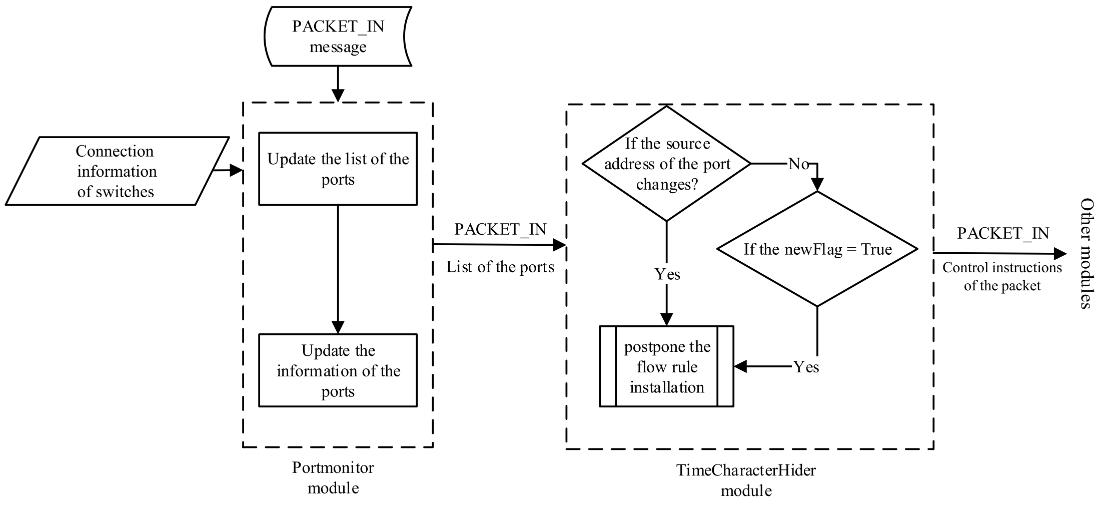
PACKET_IN message
Connection information of switches
Update the list of the ports
Update the information of the ports
Portmonitor module
PACKET_IN
List of the ports
If the source address of the port changes?
No
If the newFlag = True
Yes
postpone the flow rule installation
Yes
TimeCharacterHider module
PACKET_IN
Control instructions of the packet
Other modules

**Fig. 6.** The framework of the mitigation mechanism.

**Table 6**
Comparison of inferred match fields with actual match fields in physical testbed.

| | Inferred match fields | Actual match fields |
|---|---|---|
| RYU-4.10 | Dst MAC | Dst MAC<br>Ingress Port |
| Floodlight-1.2 | Src/Dst Port<br>Src/Dst IP<br>Src/Dst MAC | Src/Dst Port<br>Src/Dst IP<br>Src/Dst MAC<br>Ingress Port<br>EtherType<br>IP Protocol |
| OpenDaylight H | Dst IP<br>Dst MAC | Dst IP<br>EtherType |
| OpenDaylight Be | Src/Dst MAC | Src/Dst MAC<br>EtherType |

between different packets in one flow. However, this method may introduce a long transmission delay, severely affecting the network performance. Hence a more reasonable countermeasure needs to selectively delay some packets to reduce the impact on network performance as much as possible. But this method needs to maintain status information of each new flow to determine whether to add delay to the packets, which may pose great overhead to the switch or controller. Therefore, to hide the difference between the transmission time of different packets in the same flow, a lightweight mitigation method can postpone the installation of flow rules according to appropriate strategies by maintaining as little status information as possible.

## 6.2. Implementation

Based on the analysis of common characteristics of the fingerprinting attack, we propose a lightweight defense mechanism to defend against fingerprinting attacks and mitigate the threat of control information disclosure in SDN networks. The core idea of our proposal is to randomize the difference of transmission time between different packets in one flow and maintain as little status information of the flows as possible.

The defense mechanism mainly includes two modules, i.e., the PortMonitor module and the TimeCharacterHider module. The PortMonitor module monitors the status of the ports, which can be used to infer the behaviors of hosts. The TimeCharacterHider module aims to hide the difference of transmission time between the first packet and the subsequent packets in the same flow based on the ports status information from the PortMonitor module. The framework of our defense mechanism is shown in Fig. 6.

*PortMonitor module.* This module generates and maintains a list of the hosts that are directly connected with the switches in the network. The location information of the host (i.e., the ID and port of the switch connected with the host) can be obtained from the PACKET_IN messages. In SDN networks, OpenFlow switches usually cannot deal with the packets from the new host because there is no corresponding flow rule in their flow tables. Therefore, they send PACKET_IN messages containing the host information to the controller to request control instructions. For each port connected with the host, we record the source MAC and source IP of the latest host, the number of packets of the current flow, and whether the current flow is a new flow. Whether the current flow is a new flow is represent by a variable *newFlag*, we will introduce the role of this variable later. The TimeCharacterHider module can identify potential fingerprinting attack flow based on the above information.

Additionally, adversaries usually launch the fingerprinting attacks at the first step to obtain the configurations of the SDN network, which helps to launch a more targeted and destructive DoS attack. The monitoring information maintained by the PortMonitor module can also be used for detecting and defending the DoS attacks. But the precise detection and defense for the DoS attacks are outside the scope of this paper.

*TimeCharacterHider module.* This module aims to hide the difference of transmission time between the first and the subsequent packets of a new flow by selectively postponing the installation of flow rules. That is, the controller may not install the flow rule on the switch when the first packet of a new flow reaches, it will install the flow rule on the switch selectively in a later time. Therefore, the difference of transmission time between the packets of a new flow may appear randomly, which increases the difficulty in fingerprinting attempts.

The location of the host does not change frequently in a real SDN network. That is, the hosts connected to the port of the switch do not change frequently within the network. And in realistic networks, the MAC address of a physical device is usually unchanged [20]. Therefore, based on these characteristics, we treat the changes of the host's address as potential fingerprinting behaviors. So the TimeCharacterHider module only delays the rule installation when the host's address changes, which can reduce the impact on network performance.

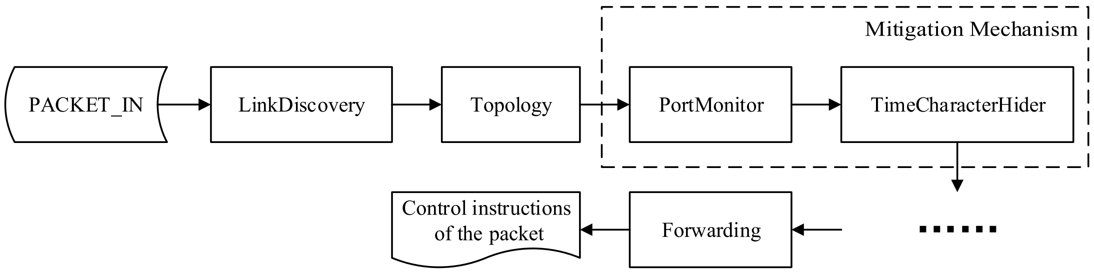

**Fig. 7.** The processing chain of PACKET_IN message.

When a PACKET_IN message arrives at the controller, the TimeCharacterHider module will extract the source IP and the source MAC of the host and compare the two addresses with the latest record in the list of the hosts obtained from the PortMonitor module. It is a detection process to identify potential fingerprinting attack packets. If both or either of these two addresses of the host is different with the latest record, the TimeCharacterHider module will postpone the installation of a flow rule randomly when the packets of this flow arrive at the controller. If both addresses of the packet are the same as the latest record, the TimeCharacterHider module needs to distinguish two cases.

- Case_1: the packet is from the same host as the last packet, the source addresses corresponding to the port keep unchanged.
- Case_2: the packet is from a different host, the source addresses corresponding to the port actually change. This packet is the subsequent packet of a new flow that triggers the delay of the flow rule installation. Therefore, both addresses of this packet are the same as the packets of this new flow that arrives before the installation of the flow rule.

We set a variable *newFlag* for each port to distinguish these two cases. *newFlag* is initially be set as False. When the source addresses corresponding to the port change, which means that a new flow appears, *newFlag* will be set as True. Therefore, the TimeCharacterHider module will also postpone the installation of the flow rule when both addresses of the packet are the same as the latest record and *newFlag* is True. Additionally, there is a predefined threshold of delay PACT_MAX, that is when the packets of one flow continuously reach the controller for PACT_MAX times, the rule will be installed at once. Therefore, most packets can still be forwarded efficiently under the mitigation mechanism, which will only introduce a negligible performance overhead to the network performance.

### 6.3. Testbed

Using Mininet to simulate the underlying network, we implement the prototype of our mitigation method on a open source SDN controller Floodlight without additional modifications in the data plane. Two modules of the mitigation mechanism are mounted on the processing chain of PACKET_IN message, as shown in Fig. 7. We use the same experimental setup as testbed 2 shown in Table 2. The simulated SDN network uses the OpenFlow 1.0 protocol. We adopt the Fat-tree topology, one of the common topologies for data center networks [21], with pod = 2 and host density = 2 (that is, each edge switch connects with 2 terminal hosts). The topology of our experiment is shown in Fig. 8. Note that we choose two hosts located in different pods as malicious hosts in the experiments.

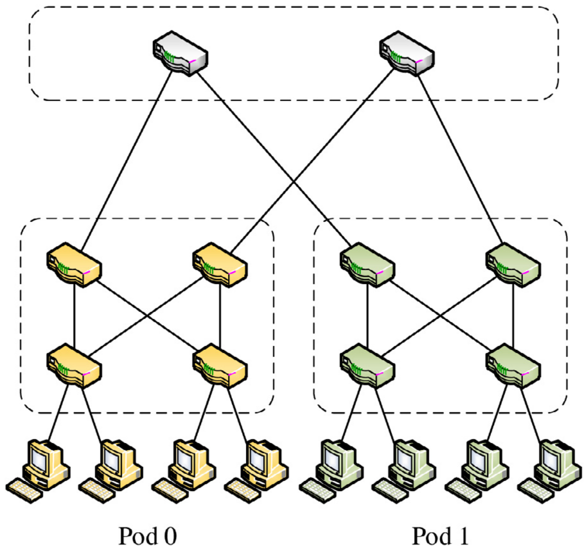

**Fig. 8.** Experimental topology.

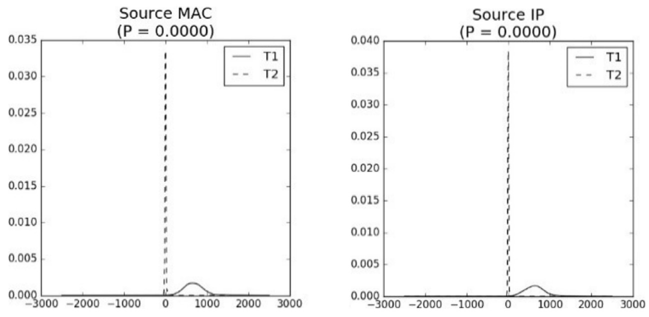

**Fig. 9.** Attack effect in the environment with the original controller. The unit of the x axis is µs.

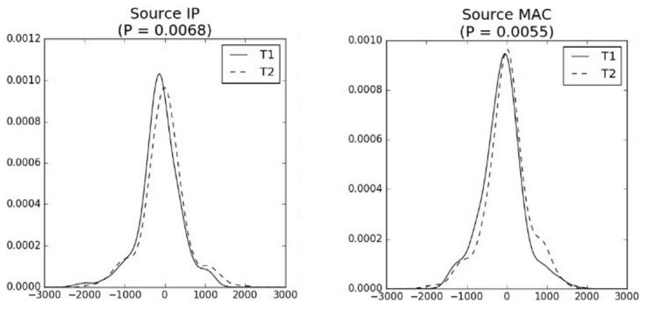

**Fig. 10.** Attack effect in the environment with the mitigation mechanism. The unit of the x axis is µs.

**Table 7**
Test results of network throughputs.

| Type of controller | TCP_STREAM | UDP_STREAM | | Packet loss ratio |
|---|---|---|---|---|
| | | Send | Receive | |
| Original controller | 81.07 Mbit/s | 76.27 Mbit/s | 75.21 Mbit/s | 1.39% |
| Controller with mitigation module | 78.75 Mbit/s | 75.93 Mbit/s | 74.85 Mbit/s | 1.41% |

**Table 8**
Communication delays of migrated hosts.

| | Average response time of 10 ping packets (ms) |
|---|---|
| Original controller | 0.891 |
| Controller with mitigation module | 0.991 |

**Table 9**
Average download time.

| | Download time of wget | Download time of TFTP |
|---|---|---|
| Original controller | 4.69 s | 3.37 s |
| Controller with mitigation module | 4.81 s | 3.31 s |

## 6.4. Results and evaluation

### 6.4.1. Defense effects

To evaluate the effectiveness of the countermeasure, we have launched our fingerprinting attacks in two network environments with: (i) an original Floodlight controller (ii) a Floodlight controller with the mitigation mechanism.

Because source addresses (i.e., src IP and src MAC) are the most common fields to be forged [22], we have mainly analyzed the experimental results of these match fields. Our fingerprinting method is still effective in the Fat-topology. The result of the fingerprinting attack in the environment (i) is shown in Fig. 9. It can be seen from Fig. 9 that $T_1$ and $T_2$ have significant difference and both P values obtained by the rank sum test are approaching 0, which demonstrates that src IP and src MAC are exactly the match fields.

Next, we set the threshold of delay PACKET_MAX $= 3$ and implement the same fingerprinting attack in environment (ii). The result is shown in Fig. 10. The result shows that the distributions of $T_1$ and $T_2$ highly overlap and both P values obtained by the rank sum test are greater than 0.05. Therefore, it is almost impossible to quantitatively judge that the two variables have significant difference.

The effectiveness of our mitigation mechanism can be demonstrated by comparing the results of the two experiments. The mitigation mechanism can considerably reduce the possibility for a remote adversary to distinguish the transmission time of different packets of the same flow by hiding the time-based features of the SDN network. Therefore, the mitigation mechanism provides effective defense against the fingerprinting attacks and prevents the SDN network from the threat of information disclosure to some extent.

### 6.4.2. Impact on network performance

We have evaluated the overhead on network performance introduced by our proposal and carried out the subsequent experiments in the environments (i), (ii), respectively.

*Network throughput.* Firstly, We use Netperf [23] to measure the network throughput between two hosts in the network. In the experiment, we set PACKET_MAX $= 3$ when the mitigation prototype runs. The results are shown in Table 7.

In the experiment of TCP traffic, compared with the result of the environment (i), the network throughput in the environment (ii) falls about 2.86% while both the send and receive throughputs in the experiment of UDP traffic fall about 0.5%. The drop rate of UDP packets increases slightly. It demonstrates that the impact of the mitigation method on the network throughput is minor.

*Migrated hosts communication.* To evaluate the impact of the mitigation mechanism on migrated hosts communication, we first select two hosts in the network. Host_1 sends 10 ping packets (10 is greater than PACKET_MAX) to Host_2, and we calculate the average response time of the 10 packets. Then we replace Host_1 by a new host Host_3 (Src MAC and Src IP changed) to send 10 ping packets to Host_2 and measure the average response time of the 10 packets. The experiment was repeated for 20 times and we calculate the average response time of the 20 experiments.

In theory, because the source address of the host connected to the port changes, the mitigation mechanism will postpone the installation of the flow rule, which may have a greater impact on the first few packets of a new flow. The subsequent packets will be forwarded directly after the flow rule is installed, which are no longer affected by the mitigation mechanism. The results are shown in Table 8.

It can be seen from Table 8 that for the first ten packets, the average communication delay is increased by 11.22%. However, this percentage is closely related to the interaction times. The fewer times of the interactions are, the more significant increase in communication delay is. Additionally, in practical networks, host migration seldom appears and the delay of the flow rule installation will be triggered only when the migrated host generates traffic for the first time.

*Normal network communication.* We evaluate the impact of our proposal on normal network communication. We select a host as a server and another host as a client. The client downloads files from the server using wget command and TFTP protocol, respectively. We repeat the above process for 20 times and calculate the average time. The wget command is based on HTTP protocol and TCP protocol, while the TFTP protocol is based on UDP protocol. The average download times of wget and TFTP are shown in Table 9.

It can be seen from Table 9 that there is no significant difference in normal network communication between environments (i) and (ii).

### 6.4.3. Load analysis of the controller

Theoretically, the mitigation method will increase the number of packets arriving at the controller because it may postpone the installation of flow rules, which may introduce the overhead to the controller. The number of packets arriving at the controller is closely related to the threshold of delay, PACKET_MAX. Assuming that there are $N$ new flows generated in the network, $N$ packets will arrive at the controller in the original environment. In the environment with the mitigation mechanism, the installation of flow rules will be postponed for $N$ times in the worst case. In this case, the number of packets arriving at the controller is:

$$n * \frac{PACKET\_MAX}{2} * (1 + \frac{1}{2^{PACK\_MAX-1}})$$
$$\approx n * \frac{PACKET\_MAX}{2} (\text{PACKET\_MAX} \to \infty)$$

Therefore, when the value of PACKET_MAX increases, the number of packets arriving at the controller increases linearly in the worst case. But in most cases, because the host connected to the port usually remains unchanged, so the flow from the host would not trigger the delay of flow rule installation. So the number of packets arriving at the controller is actually less than the theoretical value. When putting the mitigation strategies into practical use, the PACKET_MAX value should be set reasonably according to the actual load of the controller.

On the other hand, there is an additional need for the memory to maintain the status information of the port connecting with hosts in the mitigation mechanism. Assuming that there are $n$ hosts in an SDN network, the space complexity of maintaining additional port information is O(n), while the space complexity of monitoring flows' status information is $O(n^2)$ in [7,24]. Compared with the existing mitigation methods, the additional memory introduced by our proposal increases linearly with the increase number of hosts, which has a lower overhead on the controller.

## 7. Related work

There have been efforts in demonstrating the threats of information disclosure in SDNs. Zeitlin [25] fingerprints the type of controller used in SDNs, based on the characteristics of controller-switch communication. There is also work that infers controller's type by measuring the expiration time of flow rules [26]. Owing to the separation of control and data plane, SDN exhibits special time features when processing new flows. Some researchers use time-based features to fingerprint SDN, whose work is more similar to ours. To our best knowledge, there are mainly four previous efforts in this area.

Shin et al. [6] first proposed a time-based fingerprinting approach to figure out whether the target network is an SDN. At almost the same time, Klöti et al. [8] proposed an attack to reveal some information of an SDN network based on timing analysis. They measure the time for establishing sequential TCP connections to determine whether the existing rule is an aggregated rule. They tested this approach in a simulated environment. However, their approach can only obtain little information of the network.

Sonchack et al. [9] develop a novel inference attack also based on timing analysis, which is similar to a side channel attack. Their timing approach is based on the principle that the load of the control plane may affect the time the control plane takes to process a packet. To determine whether a certain test flow is traveling through the control plane, two types of data flows are injected into the network, i.e., timing probes and test packet streams. If the test packet stream is handled by control plane, the RTTs of timing probes will increase. It should be noted that the time probes are specially crafted Pings that will be forwarded to the controller, so adversaries must have prior knowledge of the controller logic. Their work mainly concerns about whether certain packets reach the controller and whether the controller has installed the corresponding flow rules. They infer some network configurations based on the analysis of timing data. In this paper, we aim at capturing more fine-grained information about the SDN, with which we can infer abundant configuration information. In their further work, a countermeasure is designed [27]. They introduced a timeout proxy between forwarding plane and control plane to equalize the response time between probe streams. That is, when the switch sends packets to the controller, the time proxy acts as a cache to average the response time of the controller. This countermeasure is not applicable to hide the difference between the processing delay of the controller and the forwarding delay of the switch. Additionally, caching all the packets to be processed by the controller in the proxy will pose challenges to proxy's load, network performance, and the design of default rules and time threshold.

Cui et al. [7] set up a testbed similar to the practical datacenter network. Choosing the RTT and packet-pair dispersion of exchanged packets as measurements, they collected a great amount of data during a long time. Their conclusion demonstrates a high accuracy in fingerprinting the SDN using these two time-based features. However, their approach requires a flow table clearing before sending each test flow, which is hard for an adversary in practice. Additionally, they designed a countermeasure that optionally added transmission delay to some packets to confuse attackers. This countermeasure was implemented on switches. The switch recorded the last appearance time of all existing flows. They considered a flow to be inactive if no packets of such flow were received by the switch in a time threshold. Instead of directly forwarding the packets based on the flow table, the switch would delay some packets of an inactive flow by a few milliseconds. By increasing transmission delay of the inactive stream, this proposal made it difficult for adversaries to determine whether the specific packet triggered the installation of the flow rule. However, this countermeasure introduce additional modifications on the switch to record status information and make decisions, which may limit the universality of the method.

On the other hand, recently some research efforts [28–30] start to explore stateful data planes by means of P4 or P4-like switches. They aim to offload some stateful traffic processing and control tasks inside the switch, so as to reduce the overhead introduced by switch-to-controller interaction. The decreases of interaction times between the controller and switches can mitigate the time-based fingerprinting threats to some extent.

## 8. Conclusion

The separation of control and data planes exposes SDNs to new threats. Adversaries can leverage this characteristic to fingerprint network configurations. Existing work has explored methods to fingerprint SDNs, but in relatively coarse-grained ways. To reveal more severe threats to SDNs, in this paper, we propose a method to capture network sensitive information by time-based analysis. Compared with previous work, the information we collect is fine-grained with high sensitivity, and our fingerprinting approach is of high adaptation and independent of controller-related information, which reveals a more dangerous information disclosure threat to SDNs. By evaluating our method in different environments with different controllers, we demonstrate that the threat exists in general SDNs. Furthermore, the experimental results from different testbeds demonstrate the difference between simulated and physical environments. Experiments indicate that an SDN attack scheme that works well in a simulated environment may not always applicable to a physical one, while many existing schemes are only evaluated with the simulated testbed. In one word, our work reveals that it is possible for adversaries to launch more dangerous attacks to SDNs by way of fine-grained fingerprinting that may capture high sensitive information.

Based on our findings, we present a lightweight countermeasure that can hide time-based features to defend against time-based fingerprinting attacks without introducing additional modifications on switches. The evaluation results of the countermeasure demonstrate its effectiveness in defense of fingerprinting attacks in SDN networks with minor overheads, preventing the network from control information leakage.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under grant No. 61472429, Beijing Natural Science Foundation, P.R. China under grant No. 4122041, and National High-Tech Research Development Program of China under grant No. 2007AA01Z414.

## References

[1] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, A. Vahdat, B4: experience with a globally-deployed software defined wan, in: ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12–16, 2013, pp. 3–14.
[2] G. Nencioni, R.G. Garroppo, A.J. Gonzalez, B.E. Helvik, G. Procissi, Orchestration and control in software-defined 5G networks: Research challenges, Wirel. Commun. Mobile Comput. (2018).
[3] I. Ahmad, S. Namal, M. Ylianttila, A.V. Gurtov, Security in software defined networks: A survey, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2317–2346.
[4] I. Alsmadi, D. Xu, Security of software defined networks: A survey, Comput. Secur. 53 (2015) 79–108.
[5] D. Kreutz, F.M.V. Ramos, P.J.E. Veríssimo, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proc. IEEE 103 (1) (2015) 14–76.
[6] S. Shin, G. Gu, Attacking software-defined networks: a first feasibility study, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN 2013, Chinese University of Hong Kong, Hong Kong, China, 2013, pp. 165–166, Friday, August 16, 2013.
[7] H. Cui, G.O. Karame, F. Klaedtke, R. Bifulco, On the fingerprinting of software-defined networks, IEEE Trans. Inf. Forensics Secur. 11 (10) (2016) 2160–2173.
[8] R. Klöti, V. Kotronis, P. Smith, OpenFlow: A security analysis, in: 2013 21st IEEE International Conference on Network Protocols, ICNP 2013, GÖttingen, Germany, October 7–10, 2013, pp. 1–6.
[9] J. Sonchack, A.J. Aviv, E. Keller, Timing SDN control planes to infer network configurations, in: Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, SDN-NFV@CODASPY 2016, New Orleans, la, USA, March 11, 2016, pp. 19–22.
[10] ONF, Software-Defined Networking (SDN) Definition, https://www.opennetworking.org/sdn-resources/sdn-definition.
[11] N. McKeown, T. Anderson, H. Balakrishnan, G.M. Parulkar, L.L. Peterson, J. Rexford, S. Shenker, J.S. Turner, OpenFlow: enabling innovation in campus networks, Comput. Commun. Rev. 38 (2) (2008) 69–74.
[12] OFN, OpenFlow Switch Specification (Version 1.0.0), https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.
[13] Scapy, http://www.secdev.org/projects/scapy/.
[14] mininet, http://mininet.org/.
[15] Ryu, http://osrg.github.io/ryu/.
[16] Floodlight, http://www.projectfloodlight.org/floodlight/.
[17] OpenDayLight, https://www.opendaylight.org/.
[18] Wikipedia, Mann–Whitney U test, https://en.wikipedia.org/wiki/Mann--Whitney_U_test.
[19] Open vSwitch, http://openvswitch.org/.
[20] S. Deng, X. Gao, Z. Lu, X. Gao, Packet injection attack and its defense in software-defined networks, IEEE Trans. Inf. Forensics Secur. 13 (3) (2018) 695–705.
[21] M.F. Bari, R. Boutaba, R.P. Esteves, L.Z. Granville, M. Podlesny, M.G. Rabbani, Q. Zhang, M.F. Zhani, Data center network virtualization: A survey, IEEE Commun. Surv. Tutor. 15 (2) (2013) 909–928.
[22] S. Deng, X. Gao, Z. Lu, Z. Li, X. Gao, DoS vulnerabilities and mitigation strategies in software-defined networks, J. Netw. Comput. Appl. 125 (2019) 209–219.
[23] Netperf, https://hewlettpackard.github.io/netperf/.
[24] M. Dhawan, R. Poddar, K. Mahajan, V. Mann, SPHINX: Detecting security attacks in software-defined networks, in: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8–11, 2015.
[25] Z.J. Zeitlin, Fingerprinting Software Defined Networks and Controllers, Tech. Rep., Air Force Institute of Technology Wright-Patterson AFB School, 2015.
[26] A. Azzouni, O. Braham, T.M.T. Nguyen, G. Pujolle, R. Boutaba, Fingerprinting openflow controllers: The first step to attack an SDN Control Plane, in: 2016 IEEE Global Communications Conference, GLOBECOM 2016, Washington, DC, USA, December 4–8, 2016, pp. 1–6.
[27] J. Sonchack, A. Dubey, A.J. Aviv, J.M. Smith, E. Keller, Timing-based reconnaissance and defense in software-defined networks, in: Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5–9, 2016, pp. 89–100.
[28] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, M. Conti, A survey on the security of stateful SDN data planes, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1701–1725.
[29] P. Vörös, A. Kiss, Security middleware programming using P4, in: Human Aspects of Information Security, Privacy, and Trust - 4th International Conference, HAS 2016, Held As Part of HCI International 2016, Toronto, on, Canada, July 17–22, 2016, Proceedings, 2016, pp. 277–287.
[30] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, P. Castoldi, P4 edge node enabling stateful traffic engineering and cyber security, J. Opt. Commun. Netw. 11 (1) (2019) A84–A95.

**Jianwei Hou** received the B.S. degree in information security from Harbin Engineering University, Harbin, P.R. China, in 2016. She is currently pursuing the Ph.D. degree at the School of Information, Renmin University of China, Beijing, P.R. China. Her research interests include system security, software-defined networking, and IoT security.

**Minjian Zhang** received the B.S. degree in information security and the M.S. degree in computer science and technology from Renmin University of China, Beijing, P.R. China, in 2015 and 2018. She researched on SDN security for her M.S. degree.

**Ziqi Zhang** received the B.S. degree in Information Technology from Wuhan University of Technology, Wuhan, P.R. China. in 2016. He is currently pursuing the M.S degree at the School of Information, Renmin University of China, Beijing, P.R. China. His research interests include system security, network security, and software-defined networking.

**Wenchang Shi** received his B.S. degree in Computer Science from the Department of Computer Science and Technology, Peking University, Beijing, P.R.China, and his Ph.D. degree in Computer Science from the Institute of Software, Chinese Academy of Sciences, Beijing, P.R. China. Currently, he is a Professor at School of Information, Renmin University of China, Beijing, P.R. China. He is a member of the Steering Committee of Cybersecurity Education, Ministry of Education, China, the vice president of the China Cyber and Information Law Society, and the Vice Chair of the Academic Committee, China Cloud Security Alliance. His research interests include System Security, Trusted Computing and Digital Forensics.

**Bo Qin** received her Ph.D. degree in Cryptography from Xidian University in 2008 in China. Since then, she has been with Xi'an University of Technology (China) as a lecturer and with Universitat Rovira i Virgili (Catalonia) as a postdoctoral researcher. She is currently a lecturer in the Renmin University in China. Her research interests include pairing-based cryptography, data security and privacy, and VANET security. She has been a holder/co-holder of 5 China/Spain funded projects. She has authored over 60 publications and served in the program committee of several international conferences in information security.

**Bin Liang** received the Ph.D. degree in Computer Science from Institute of Software, Chinese Academy of Sciences. He is currently a professor at School of Information, Renmin University of China. His research interests focus on program analysis, vulnerability detection and Web security.