Fine-Grained Fingerprinting Threats to Software-Defined Networks

Minjian Zhang, Jianwei Hou, Ziqi Zhang, Wenchang Shi*, Bo Qin, Bin Liang
School of Information, Renmin University of China, Beijing 100872, P.R.China
** wenchang@ruc.edu.cn*

***Abstract*—Thanks to its flexibility and programmable features, Software-Defined Networking (SDN) has been attracting more and more attention from the academia and the industry. Unfortunately, the fundamental characteristic of SDN that decouples control plane from data plane 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 and reveals the much more severe threats to SDN Security. By analyzing network packets, the approach digs out match fields of SDN flow rules innovatively. Being sensitive and control-related information in SDN, the match fields of flow rules can be used to infer the type of an SDN controller and the security policy of the network. With these sensitive configuration information, adversaries can launch more targeted and destructive attacks against an SDN. We implement our approach in both simulative and physical environments. Furthermore, we conduct experiments with different kinds of SDN controllers to verify the effectiveness of our concept. Experiment results demonstrate the feasibility to obtain highly sensitive, fine-grained information in SDN, and hence reveal the high risk of information disclosure in SDN and severe threats of attacks against SDN.**

Keywords—SDN; fingerprinting; timing attacks; information disclosure

I. INTRODUCTION

Software-Defined Networking (SDN), an emerging network architecture, consists of application layer, control layer and infrastructure layer. It decouples the network controlling and forwarding functions, enabling centralized management with software. It has attracted more and more research organizations and commercial enterprises [1,2] due to its flexible and programmable features. 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. To ensure centralized network control, in SDN, the network status and control information are managed and issued by the control plane, which opens a door for attackers. The key issue that 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 confirm whether a target network is an SDN remotely, which uses a time-based method for fingerprinting [6]. Based on the fingerprinting results, a targeted DoS attack can be launched. With 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 an installation of a flow rule. Parallel research efforts assume that the adversary is inside the network, inferring whether the certain flow triggers the interaction between the data and control plane [8,9]. On account of 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 characteristics of centralized control. However, most of their work is limited to determining whether a certain packet triggers a flow rule installation, and information collected in this way is coarse-grained. What we want to discuss further is that whether adversaries are able to fingerprint an SDN in a more fine-grained way, and expose the target network to a more severe threat for which new countermeasure need to be taken.

In this paper, we explore a new way to obtain fine-grained and highly sensitive information of an SDN. We aim at match fields of flow rules in an SDN, and implement our design in different environments with diverse controllers to illustrate the generality of such threat to SDNs. Furthermore, while most existing work evaluate their methods either in a simulative or a physical environment, we carried out our experiments in both kinds of environments to demonstrate the difference in implementing the same design.

In the remainder of this paper, we present our research background in Section Ⅱ. In Section Ⅲ, we bring out the design principles of our fingerprinting method for the match fields of flow rules. We implement our method in simulative and physical environments respectively. The results of our experiments are presented and analyzed in Section Ⅳ and Section Ⅴ. In Section Ⅵ, we discuss related work, then we conclude in Section Ⅶ.

II. BACKGROUND

A. Motivation

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

2324-9013/17 $31.00 © 2017 IEEE
DOI 10.1109/Trustcom/BigDataSE/ICESS.2017.229

Controller type. 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 and launch attacks with the knowledge of controller logic. Observations show that different controllers usually use different default match fields in flow rules. If adversaries can capture information about match fields, they may infer controller type 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 according to the installed flow rules. Therefore, it is possible for a policy defined by an upper layer to be inferred with flow rules information.

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 SDN control logic, mismatched network packets will be forwarded to control plane for decision, and the controller will deliver control instructions to switches for processing these packets. Being aware of match fields of flow rules, adversaries can craft packets that mismatch the existing flow rules. By changing values of packet headers, heavy traffic may be generated to travel through the control plane. 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 installations, which would overflow the limited space of flow tables.

In short, flow rules are of great value as they reflect the control logic of an SDN. As an important part of flow rules, match fields are treated as highly sensitive information in an SDN environment. In our knowledge, our work is the first attempt to explore an effective method to fingerprint match fields of flow rules, and investigate the feasibility of obtaining highly sensitive and fine-grained information in SDNs without knowing network configurations (e.g. controller type) in advance. In this paper, we set up simulative and physical environments for evaluation, and use four different types of controllers to verify the effectiveness of our method. Experiment results reveal the difference between simulative and physical experiments, and ask for optimization of analysis method.

B. 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 using open interface provided by control plane to implement complex logic. Infrastructure plane is composed of network devices that are only responsible for executing forward/drop actions. Network devices are deprived of control functions, since control plane maintains network status information and configurations. To guide the action of network devices, control plane delivers control messages to infrastructure plane through southbound interface. Network management becomes agile thanks to such architecture, as network administrators can easily deploy and manage network by programming an SDN application. In the generalized SDN architecture, there are various means to implement southbound interface. Among all those implementations, OpenFlow is the ONF-recommended and the most widely accepted one. In this paper, we discuss the SDN that uses OpenFlow.

OpenFlow was firstly proposed by McKeown e*t al.* [11]. It 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. According to OpenFlow, flow entries can be defined through secure channel remotely, and the secure channel connects switches to the remote control process (controller). In this way, control plane dominates network devices by OpenFlow protocol. Obviously, the flow entries in flow tables become the most important control information, which reflect the control logic to some extent.

C. 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 only depending on flow rules. One flow rule consists of three parts: header fields, counter and action. Header fields are the basis for matching. As listed in Table I, OpenFlow 1.0 uses 12-tuple header fields [12]. When a data packet comes, the switch firstly sets ingress port and identifies Ethernet type, then adds VLAN ID and other information according to the Ethernet type. After parsing and extracting packet header, the switch matches this information with the values of header fields defined in flow rules. Once matched, the switch executes the action defined in this flow rule. Usually, in a flow rule, not all the fields require an exact match. Some fields may be set to ANY that means it can match any value, so values of these fields won't influence the match results. If different values

TABLE I. FIELDS USED TO MATCH AGAINST 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

of certain field cause different match results, we will call this field a match field.

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.

III. Fingerprinting Scheme Design

Before introducing how to fingerprint the match fields, we first investigate the packet processing procedure in SDN. According to the essential characteristics of SDN, we propose our method of obtaining match fields information.

A. 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:

1) forward directly: The switch forward this flow's packets to the specified port/ports.

2) forward to the controller: The switch packs the packet and send it to the controller. The controller analyzes the packet and installs a flow rule on the switch optionally.

3) drop: For security or other reasons, switches drop the packet.

Packets of the same flow have same value in every header fields. When the value of a field is fixed, the switch takes different actions to handle the same flow's packets, this field is a match field. (It indicates that the first packet triggers an interaction between controller and the switch). When the value of a field is changed and the switch handles different packets with different actions, this field is also a match field (It indicates that the different values of this field result in different matches).

B. Fingerprinting Scheme

Based on the analysis presented in III-A, we consider the following scenario. We suppose that there are two hosts in the same SDN network and one acts as Sender sending a packet to the other.

1) Receiver receives the packet

a) 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 action (A-1).

b) The switch has not found a match entry, so it will send this packet to the controller for decision. The controller analyzes the packet and installs a corresponding rule on the switch to handle this flow. In this case, the switch performs action (A-2).

c) 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 action (A-2) in this case.

In (1-b) and (1-c), the time of a packet travelling from Sender to Receiver is significantly longer than case (1-a), as the packet triggers an interaction between the controller and the switch.

2) Receiver doesn't receive the packet

a) The switch will ask controller for decision to handle the packet if there is no corresponding rule in flow tables. If the controller doesn't deliver forwarding instructions or flow rules to the switch, the packet will get lost. Here, the switch performs action (A-2).

b) The switch drops the packet according to the corresponding rule. The switch performs action (A-3) in this case.

c) The packet gets lost due to network congestion or other occasional reasons.

We design a scheme to fingerprint match fields on the basis of the above analysis. Supposing that Sender and Receiver are on the same network (LAN) and able to communicate with each other normally, ignoring the occasional case (2-c), we will determine whether a header field is a match field.

For a given header field, we send a number of packets with the same header value at a certain time 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 difference between the time duration helps us distinguish (1-a) from (1-b)(1-c), we can infer that 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 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 by different actions. It may happen due to flow rule expiration. However, it rarely happens in theory if we do not consider the occasional factors.

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

The number of switches that a packet passes through may influence the analysis results. Some controllers will install flow rules to all switches in the forwarding path, once

receives a new flow packet. However, the controller may install a flow rule only to the switch that sends PACKET_IN message to it. As the size of our probe packets is small, we assume that the transmission delay of packets is negligible. Based on the consideration above, we conduct our experiments using one or two OpenFlow switches. Experiments in a more complex environment will be conducted in our future work.

IV. Implementation and Evaluation in Simulative Testbed

A. Implementation

Based on the analysis in Section III, 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 our 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.

- Select a header field ,MF, to be tested, and define N to represent the number of samples.
- 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. Sender sends the packet P to Receiver again, and we use t_2 to represent the time lag between sending and receiving (if Receiver doesn't receive both packets, we will not record any time information).
- Repeat the previous step for N times, after which we have T_1, a sample set of the values of t_1, and T_2 of t_2.
- Process and analyze the sets of t_1 and t_2, and infer whether the tested header field MF is a match field based on feature (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 via the timestamps provided by TCPDUMP.

B. 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 simulative SDN environments use the OpenFlow1.0 protocol. The details of the testbed are shown in Table II. There are two OpenFlow switches connecting with each other in the testbed. Host h1 connects to switch s1. Host h2 and h3 connect to switch s2. Note that h1 and h2 are malicious hosts controlled by the adversary. They are in the same LAN and can communicate normally.

TABLE II. Configurations of Simulative Testbed

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

C. Results and Evaluation

We have implemented the method described above to fingerprint the match fields of the flow rules on the testbeds with different controllers. Considering some of the match fields (e.g., IP ToS) defined in OpenFlow1.0 only have several legal values, we mainly focus on the commonly used fields with wide value range: src/dst port, src/dst IP, src/dst MAC. Besides, we also fingerprint ToS field and analyze its values as the control group. For each field to be tested, we randomly choose 300 different values from its legal value span, namely, 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 Figure 1 (Floodlight as an example). In Figure 1, the solid line represents the PDF of T_1, and the dotted line represents the PDF of T_2. 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 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 between T_1 and T_2 in the PDFs figure. Two samples are hard to be differentiated with high overlap ratio, which means that there is no significant difference between two samples. We show the P-value produced from rank-sum test for each measured field in Table Ⅲ.

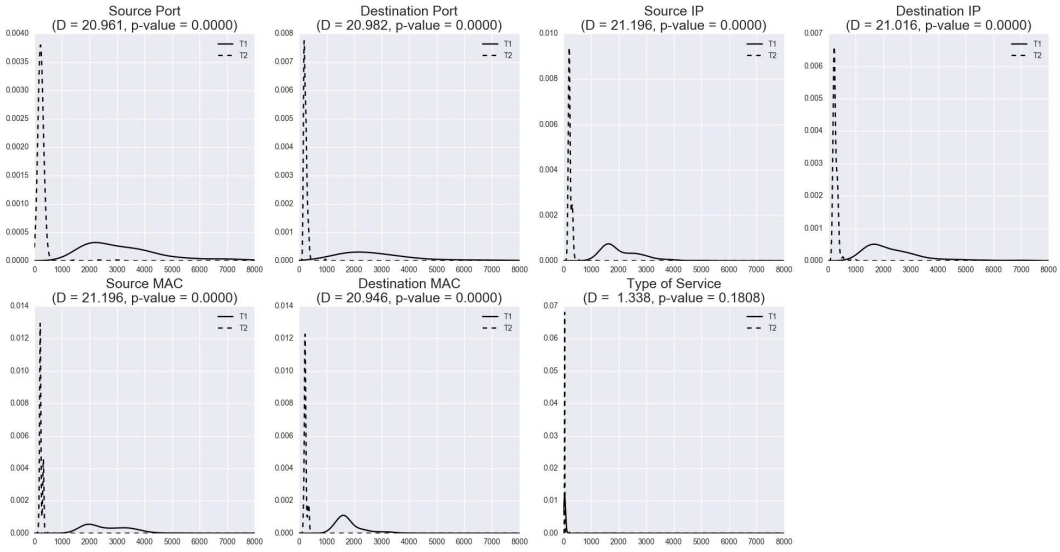

Fig. 1. PDFs of T_1 (solid line) and T_2 (dotted line) for each measured field, in the simulative SDN with Floodlight controller. The unit of the x axis is

TABLE III. 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

According to hypothesis testing theory, when the P-value is below 0.05, the null hypothesis will be rejected, so T_1 is significantly different from T_2. Combining the results shown in Table III with the packets received, we can infer that whether the measured field is a match field. In Table IV, 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. Because of the limited legal value range, we are not able to randomize the value of these fields. The field in bold indicates a false positive.

In the experiment using 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 strict, 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.

TABLE IV. COMPARISON OF INFERRED MATCH FIELDS WITH ACTUAL MATCH FIELDS IN SIMULATIVE TESTBED

	Inferred Match Fields	Actual Match Fields
RYU-4.10	Dst MAC	Dst MAC, *Ingress Port*
Floodlight-1.2	Src/Dst Port, Src/Dst IP, Src/Dst MAC	Src/Dst Port, Src/Dst IP, Src/Dst MAC *Ingress Port, Ether Type, IP Protocol*
OpenDaylight H	Dst IP, **Dst MAC**	Dst IP, *Ether Type*
OpenDaylight Be	Src/Dst MAC	Src/Dst MAC

TABLE V. CONFIGURATIONS OF PHYSICAL TESTBED

CPU	Intel(R) Xeon(R) CPU E5506 @ 2.13GHz 4 cores
Memory	3.8GB
OS	3.13.0 kernel distribution，Ubuntu 14.04 LTS

V. IMPLEMENTATION AND EVALUATION IN PHYSICAL TESTBED

A. Implementation

The basic principle of implementation in a physical environment is roughly the same as that in simulative environment. And we use the same features (1), (2) mentioned in IV-A as measurements. But in practical situation, it's 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 times of different machines are prone to bias as time goes on. With the consideration of synchronization, we adjust the implementation described in Section IV-A as follow.

- Select a header field ,MF, to be tested, and define N to represent the number of samples.

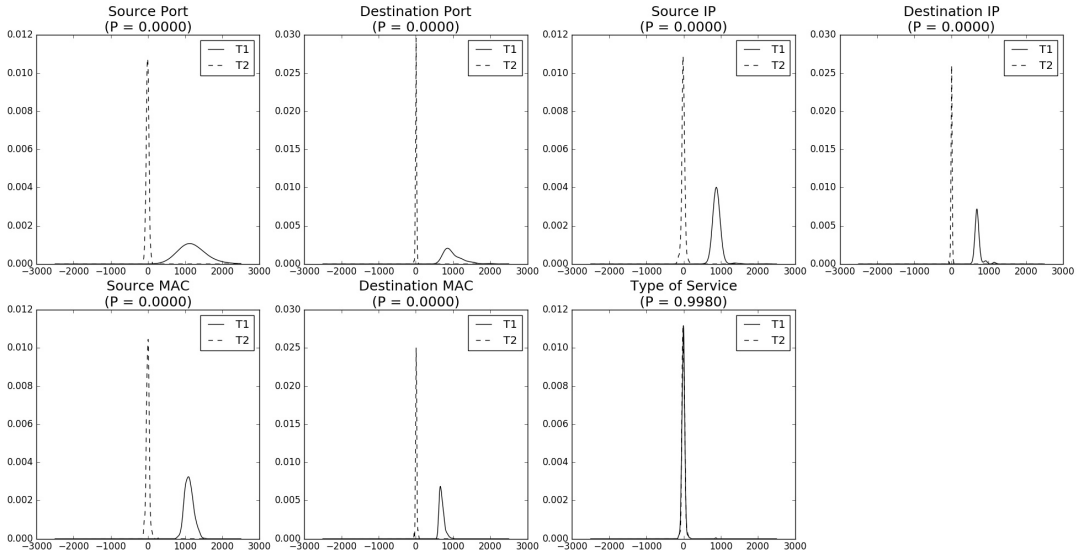

Fig. 2. PDFs of T_1 (solid line) and T_2 (dotted line) for each measured field, in the SDN with Floodlight controller. The unit of the x axis is μs.

TABLE VI. 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

- Sender randomly selects a value from the legal value range of MF, and keeps values of other fields the same as before. Then 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 lag between two timestamps. Sender repeats sending the same packet P for three times. Then we compute three lags, t_2, t_3 and t_4, respectively. Every packet is sent at a 1-second interval. After getting t_1~t_4, we compute $\Delta t_1=t_1-t_2$, $\Delta t_2=t_3-t_4$. A group of measurements is done as above.
- Repeat the previous step for N times, after which we have a set of Δt_1 values, represented as TC_1, and a set of Δt_2 values as TC_2.
- Process and analyze TC_1 and TC_2, and infer whether the tested header field MF is a match field based on feature (1) and (2).

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

B. Testbed

A four-host physical testbed is setup 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 setup testbed, and OpenFlow1.0 protocol is used. Detailed information about the four physical machines is listed in Table V. The same as that in simulative case, h1 and h2 are malicious hosts controlled by the adversary. They are in the same LAN and can communicate with each other.

C. Results and Evaluation

Similar to our evaluation in Section IV, 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 IP ToS field as the control group. For each field being tested, we randomly choose 500 different values from its legal value span, namely, we define N as 500.

Statistical analysis. Because different variables are used in physical evaluation, we adjust the statistical analysis method. We treat t_1~t_4 as four different variables. We define another two variables T_1 and T_2, representing 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 set TC_1 and TC_2 defined in V-B are samples of T_1 and T_2.

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 packets that follow 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 from switch-controller interaction. Therefore, we can infer whether the switch-controller interaction happens depending on the degree of difference between T_1 and T_2.

Firstly, we apply Gauss kernel density estimation to TC_1 and TC_2, computing and drawing the PDFs of T_1 and T_2. It's intuitive to observe the difference and overlap of value distributions. We take the fingerprinting result of Floodlight as an example and show it in Figure 2.

Secondly, we quantify the discrepancy between T_1 and T_2. In simulative evaluation, we use a hypotheses testing method to evaluate the difference between two variables. However, this method is inapplicable to our physical evaluation. T_1 and T_2 are both deviation values which will be very slender in practice, which means that the value ranges of T_1 and T_2 are small. So any little nuance of the mean value (e.g. 100μs) 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 this nuance is not a fixed value, we cannot simply minus it from each measurement. So we adopt another analyzing approach with deviation in consideration instead of traditional hypotheses testing methods.

Before analyzation, an acceptable margin of deviation Δx should be defined. Then, we calculate the mean of variable T_2 as μ. Finally, we compute the probability P of T_1 distributing in range $(\mu-\Delta x, \mu+\Delta x)$. A large P value means 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 limit. The deviation Δx is an empirical value that differs depending on experimental environments. We set Δx as 200μs in our experiment, as the RTT of pings between Sender and Receiver is around 400μs. We summarize the P values of different fields in Table VI.

When P is close to zero, we conclude that T_1 is significantly different from T_2, even having taken environment effects into consideration. So we infer that this measured header field is a match field. In Table Ⅶ, we compare the inferred match fields based on our fingerprinting with the actual match fields in use. The underlined fields are overlooked in our fingerprinting, but they are match fields indeed.

It should be noted that, the false positive disappears due to our adjusted analyzing approach. Thus, analysis that takes deviation into account is more accordant with practical situation. 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 planning attacks.

TABLE VII. 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, *Ingress Port*
Floodlight-1.2	Src/Dst Port, Src/Dst IP, Src/Dst MAC	Src/Dst Port, Src/Dst IP, Src/Dst MAC *Ingress Port, Ether Type, IP Protocol*
OpenDaylight H	Dst IP	Dst IP, *Ether Type*
OpenDaylight Be	Src/Dst MAC	Src/Dst MAC

VI. RELATED WORK

There have been efforts in demonstrating the threats of information disclosure in SDNs. Zeitlin [20] fingerprints the type of controller used in SDNs on the basis of the characteristics of controller-switch communication. There is also work that infers controller type by measuring the duration of flow rules [21]. Owing to the separation of control and data plane, SDN exhibits special time features when processing new flows. On the basis of these fundamental characteristics, more adaptive and effective approaches can be designed, which reveal an unavoidable threat that the SDN is faced with. Accordingly, some researchers use time-based features to fingerprint SDN, whose work is closer to ours. To our best knowledge, there are mainly four previous efforts in this area.

Shin *et al.* [6], for the first time, proposes 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 a timing attack aiming at revealing information. They measure the different time for establishing sequential TCP connections to determine whether the existing rule is an aggregated rule. They assess this approach in simulative environment. However, little amount of information can be collected using their approach.

Sonchack *et al.* [9] develop a novel timing attack similar to the side channel attack that times execution. In their further work, a countermeasure is designed [22]. Their timing approach is based on the principle that control plane load affects how long the control plane takes to process a packet. To determine whether a certain test flow is travelling through control plane, two types of data flows are injected into the network, timing probes and test packet streams. If the test packet stream is handled by the 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, for which adversaries must have prior knowledge on the controller logic. Their work mainly concerns about whether certain packets reach the controller and whether the controller install flow rules. They infer some network configurations based on the analysis of timing data. From another perspective, we aim at capturing more fine-grained information about the SDN, with which we can also infer abundant configuration information.

Cui *et al.* [7] setup a testbed similar to practical datacenter's 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 features. However, their approach requires a flow table clearing before each test flow, which is hard for an adversary in practice. So their work tends to be a kind of validation.

VII. Conclusion

The separation of control and data planes exposes SDNs to new threats. Adversaries are able to use this characteristics to fingerprint network configurations. Existing work has discussed methods to fingerprint SDNs, but in relatively coarse-grained ways. In order to reveal more severe threats to SDNs, in this paper, we innovatively 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 exhibits more dangerous information disclosure threats to SDNs. By evaluating our method in different environments with different controllers, we illustrate that the threats may exist in general SDNs. Furthermore, the test results from different testbeds demonstrate the difference between simulative and physical environments, which indicates that an SDN attack scheme that works well in a simulative environment may not always applicable to a physical one, while many existing schemes are only evaluated with simulative tests. 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.

In our future work, we will discuss whether such serious threats may appear in more complex network environments. We are going to explore a possible way to fingerprint more kinds of match fields, which may expose SDNs to a greater danger. To defense the revealed threats, corresponding measures need to be taken.

Acknowledgment

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

References

[1] S. Jain et al., "B4: Experience with a Globally-Deployed Software DefinedWA," Proc. ACM SIGCOMM 2013 Conf. SIGCOMM - SIGCOMM '13, p. 3, 2013.

[2] A. Andreyev, "Introducing data center fabric, the next-generation facebook data center network." [Online]. Available: https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/.

[3] I. Ahmad, S. Namal, M. Ylianttila, and A. Gurtov, "Security in Software Defined Networks: A Survey," IEEE Commun. Surv. Tutorials, vol. 17, no. 4, pp. 1–1, 2015.

[4] I. Alsmadi and D. Xu, "Security of Software Defined Networks: A Survey," Comput. Secur., vol. 53, pp. 79–108, 2015.

[5] D. Kreutz, F. M. V Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proc. IEEE, vol. 103, no. 1, pp. 14–76, 2015.

[6] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," Proc. Second ACM SIGCOMM Work. Hot Top. Softw. Defin. Netw., pp. 165–166, 2013.

[7] H. Cui, G. O. Karame, F. Klaedtke, and R. Bifulco, "On the Fingerprinting of Software-Defined Networks," IEEE Trans. Inf. Forensics Secur., vol. 11, no. 10, pp. 2160–2173, 2016.

[8] R. Klöti, V. Kotronis, and P. Smith, "OpenFlow: A security analysis," in Proceedings - International Conference on Network Protocols, ICNP, 2013.

[9] J. Sonchack, A. J. Aviv, and E. Keller, "Timing SDN Control Planes to Infer Network Configurations," Proc. 2016 ACM Int. Work. Secur. Softw. Defin. Networks Netw. Funct. Virtualization, pp. 19–22, 2016.

[10] ONF, "Software-Defined Networking (SDN) Definition." [Online]. Available: https://www.opennetworking.org/sdn-resources/sdn-definition.

[11] N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, p. 69, 2008.

[12] ONF, "OpenFlow Switch Specification (Version 1.0.0)." [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf.

[13] "Scapy." [Online]. Available: http://www.secdev.org/projects/scapy/.

[14] "Mininet." [Online]. Available: http://mininet.org/.

[15] "Ryu." [Online]. Available: http://osrg.github.io/ryu/.

[16] "Floodlight." [Online]. Available: http://www.projectfloodlight.org/floodlight/.

[17] "OpenDaylight." [Online]. Available: https://www.opendaylight.org/.

[18] "Mann–Whitney U test," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Mann–Whitney_U_test.

[19] "Open vSwitch." [Online]. Available: http://openvswitch.org/.

[20] Zachary J. Zeitlin, "Fingerprinting Software Defined Networks and Controllers," AIR FORCE INSTITUTE OF TECHNOLOGY, 2015.

[21] A. Azzouni, O. Braham, N. Thi, M. Trang, G. Pujolle, and R. Boutaba, "Fingerprinting OpenFlow controllers: The first step to attack an SDN control plane," 2016.

[22] J. Sonchack, A. Dubey, A. J. Aviv, J. M. Smith, and E. Keller, "Timing-based reconnaissance and defense in software-defined networks," Proc. 32nd Annu. Conf. Comput. Secur. Appl. - ACSAC '16, pp. 89–100, 2016.