Unveiling SDN Controller Identity through Timing Side Channel

Sukwha Kyung, Jaejong Baek, Gail-Joon Ahn
Arizona State University
Tempe, AZ, USA
{sukwha.kyung, jaejong, gahn}@asu.edu

Abstract—**Software-defined networking (SDN) has revolutionized the landscape of network management by decoupling control and data planes and becoming the backbone of many IT infrastructures including data centers, cloud computing, and enterprise networks. At the same time, however, the control plane has become a prime target for adversaries due to its critical role in network operations and centralized control functions. In this paper, we demonstrate how to discover the identity of different SDN controllers, which could be leveraged for more sophisticated attacks by adversaries. Our approach adopts a timing-based side channel and deep neural networks (DNN). To achieve this, we analyze real-world SDN traffic in a research computing center and accurately identify the controllers, minimizing the impact of random noise. Despite various factors that influence controller behaviors, our fingerprinting approach achieves an average accuracy of more than 90%. Lastly, the mitigation strategies are also discussed.**

1. Introduction

Previous work on SDN fingerprinting has demonstrated the feasibility of fingerprinting whether or not the target network leverages SDN by the aforementioned timing-based side channel [1], [2], [3], [4], [5], [6]. Nevertheless, these attempts on SDN fingerprinting did not deal with *the identities of SDN controllers* such as the name and version of a controller, and thus is not sufficient for adversaries to devise an exploit against a target control plane. For example, controller-specific exploits such as directory traversal and data storage vulnerabilities heavily rely on prior knowledge about the exact identity of the target controller [7], [8], [9].

In this paper, we explore the question of *how to fingerprint a controller identity as well as estimate the size of flow paths in a real-world SDN environment*. To tackle this challenge, we observed behaviors of diverse SDN controllers and noticed that there exist variations in round-trip time (RTT) between probing packets, exhibiting a distinct distribution pattern. This distribution pattern is specific to different SDN controllers, taking into account their versions and the scale of their flow paths. In other words, any incoming flows without corresponding flow rules must go through additional procedures to be handled on the control plane, causing a delay in setting up a new flow rule, which eventually affects the response time from servers to clients.

Challenges. However, relying solely on the timing-based side channel to fingerprint SDN controller identity has two major challenges. (C1) *Architectural restrictions of SDN:* One of the challenges in fingerprinting the SDN controller is inherited from its architecture design, which separates the control and data planes. This strict separation prevents direct packet injections from the data plane to the control plane. Such restriction poses a challenge for fingerprinting efforts since traditional methods with direct communication may not be feasible. (C2) *Potential performance fluctuation of the controller:* The performance of SDN controllers can be affected by several factors including random noise in the network, the physical hardware on which the controller is running, and the controller applications. These factors may create randomness in controller performance measurements, resulting in inaccuracy in fingerprinting results.

Approach. Previous work on SDN fingerprinting also explored a similar methods [4], [6], [10] but are limited to fingerprinting only the logic of the control plane. Our approach demonstrates how to further fingerprint the *identity of the controllers* (i.e., the name and version of a running controller). To address C1, we employ deep learning models to analyze the temporal patterns of RTT rather than just individual values, thereby enhancing accuracy and reliability in the non-deterministic nature of networks. In addition to fingerprinting the identity of controllers, we also estimate the size of flow paths, even under different network configurations, by leveraging deep learning models. Also, we account for variations in network sizes, SDN applications running on the controller, and traffic volume to address C2. In summary, our fingerprinting approach surpasses previous side-channel attacks against SDN by not only identifying controllers but also efficiently distinguishing between different flow paths. This capability enhances the granularity of security analysis and improves the comprehensive understanding of network behavior. Also, we found that the size of the data plane is a factor that has the significant impact on response time within the network.

We evaluate the effectiveness of our approach using major commercial-grade open source SDN controllers used in real-world data centers and enterprise networks—OpenDaylight (ODL) [11], Open Network Operating System (ONOS) [12], and Ryu [13], [14]. Our experimental results show that the flow setup delay of each controller is significantly different from each other and can be used as a distinctive feature to identify SDN controllers and their versions. Moreover, our approach can be employed to identify the specific flow rules that are assigned to individual flow paths by utilizing the flow setup delays from multiple flow paths. This implies that by analyzing the timing of the flow setup, our method enables the mapping of flow rules to their corresponding flow paths.

In summary, the contributions of this paper are as follows:

- We present a fingerprinting approach for identifying SDN controllers. Our approach takes advantage of the timing side channel in SDN and employs deep learning to minimize the impact of various factors that affect the performance of SDN controllers.
- Our approach enables the identification of different flow paths within the network. By distinguishing and understanding the various paths through which network traffic flows, an attacker can assess the presence of conflicts in network policies. These conflicts can then be exploited to craft more sophisticated and targeted attacks.
- We evaluate our approach on major commercial-grade open source controllers—ODL, ONOS, and Ryu—with different network configurations in terms of the size of the network and SDN applications running on the controller. Our experiments demonstrate that our fingerprinting is efficient in a real-world SDN-based computing environment.

2. Background

An OpenFlow-enabled switch in a data plane contains *Match-Action* flow tables, in which match-fields of each flow entry are paired with corresponding actions (*directly forward to destination*, *forward to controller*, or *drop*). For example, OpenFlow 1.3 supports 40 match fields [15], supporting various L2/L3 field values such as TCP/UDP ports, MAC and IP src/dst addresses, etc. Simply by matching the flow rule entries stored in flow tables to the incoming packets, the data plane forwards the packets to the next hop according to network policies defined by the controller, and the intelligence of various network functions (including routing, load-balancing, IDS/IPS, etc.) are offloaded to centralized network operating systems, i.e., SDN controllers. When a switch receives a new packet that does not match any flow rule entries stored in its flow tables, the switch forwards the packet to the controller encapsulated in a *Packet-In* message, requesting appropriate action regarding the packet. The controller decodes the received *Packet-In* message, retrieves policy and network status information from the data storage to analyze and generate corresponding rules regarding the packet, generates flow rules, and finally pushes the flow rules into the data plane by sending out *Flow-Mod* message. Therefore, the process of installing flow rules involves sequences of network elements (switch, controller, and links between switch and controller) and modules inside the controller to process the new flow. Due to the extra packet processing or an initial packet, the delays caused by flow rule installation are detected to fingerprint whether the given network is SDN or not [1].

Data stores and OpenFlow APIs are the major factors leading to different delays between SDN controllers. The data storage in ODL is built based on Model-Driven Service Adaptation Layer (MD-SAL), which is an abstracted SDN data storage component implemented using Java and YANG. The main purpose of MD-SAL in ODL is to provide a common API to define data storage access and definition of data, along with the ease of flexible extension of the control plane. ONOS also implements its data storage in a similar manner, which is based on YANG management system to implement RESTCONF and NETCONF. The design of the data storage in ONOS, however, specifically focuses on the seamless scaling of the network and optimization of synchronization between distributed data storages. This design for data storage in ONOS is implemented as a separate Dynamic Configuration Manager module along with the YANG management system [16]. For Ryu, the data storage is also implemented simply for centralized management of the OpenFlow switch. As a result, an API for OpenFlow configuration protocol (i.e., NETCONF) is implemented in Python.

In addition, the implementation of OpenFlow API is different from controller to controller. When ODL receives a *Packet-In* message, an OpenFlow protocol plugin called *openflowjava* translates the message and passes it to the OpenFlow configuration API in MD-SAL. MD-SAL generates flow rules containing actions to be performed on the flow through communication with appropriate network service implementations (e.g., router, Access Control List (ACL), IDS/IPS, etc.), and registers the router information in the configuration data storage. After generating the flow rules, the OpenFlow plugin is called again to translate the message into *Flow-Mod* message, and then, the message is sent and installed in the flow tables in switches. Similarly, the flow processing in the other two controllers involves different implementations of OpenFlow APIs and data storages to generate appropriate flow rules. In ONOS, *FlowRule-Service API* is involved in the generation and modification of flow rules, along with *DynamicConfigStore* modules to support configuration in the data storage. Lastly, the listener implemented in Ryu detects the *Packet-In* event and utilizes multiple libraries in OpenFlow API implementations. These libraries include *Packet, OF-Config*, and *OVSDB* libraries.

3. Fingerprinting SDN Controllers

We first present the threat model followed by the attack scenarios. We then discuss our approach to collecting and analyzing flow setup delays.

3.1. Threat Model

Our threat model assumes that the attacker performs reconnaissance in an SDN-based network by injecting and sniffing the traffic to collect information in the data plane. Conceptually, the attacker achieves this goal by actively

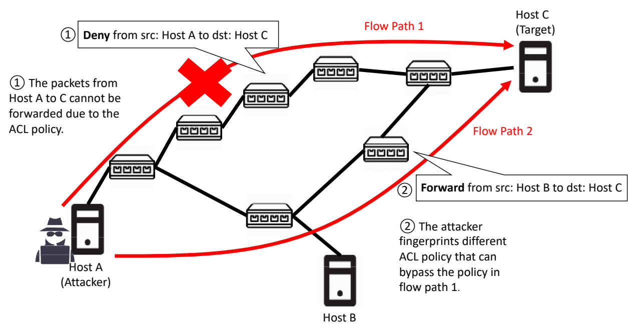

① Deny from src: Host A to dst: Host C
Flow Path 1
Host C (Target)
① The packets from Host A to C cannot be forwarded due to the ACL policy.
Flow Path 2
② Forward from src: Host B to dst: Host C
② The attacker fingerprints different ACL policy that can bypass the policy in flow path 1.
Host A (Attacker)
Host B

Figure 1. Advanced Attack Scenario.

sending probing flows with various match fields for which corresponding flow entries do not exist in the data plane. Subsequently, the SDN controller is invoked for processing the new flows while disclosing the information regarding the control plane through its processing delay. The attack takes place only at end hosts without compromising any component in the data plane.

3.2. Attack Scenarios

We define three attack scenarios where our approach can be used to obtain the identity of controllers (Scenario 1) and sizes of flow paths (Scenario 2). Then we present an advanced attack case in which the information fingerprinted from the first two scenarios is leveraged to find conflicts in network policies (Scenario 3).

Scenario 1. Fingerprinting controller identity (S1). To fingerprint the identity of the controller, an attacker injects probing packets into the network and calculates flow setup delays. By comparing the obtained pattern of delays in different controllers, the attacker can determine the distinctiveness of the target controller. To that end, we design our fingerprinting approach by leveraging DNN models to extract features and specific patterns related to each controller, and precisely classify the obtained flow setup delays.

Scenario 2. Revealing the sizes of flow paths (S2). From our observation, we find that the delay caused by flow rule installation is mainly affected by the number of switches in the flow path: The more switches that exist in the flow path, the more distinctive patterns emerge from a set of flow setup delays of an SDN controller. Thus, the attacker can estimate the size of flow paths (i.e., the number of switches in flow paths) by analyzing the patterns of flow setup delays from multiple flow paths. In other words, the attacker can differentiate one flow path from another based on the different sizes of the flow path.

Scenario 3. Advanced attack (S3). Finally, we show that our SDN fingerprinting method can also discover possible policy misconfiguration in the network and leverage it to perform a more advanced attack. The ability to estimate the size of a flow path from S2 enables the attacker to differentiate each flow path and can fingerprint match fields that are specific to each flow path. This method provides a way to bypass flow rules (e.g., firewall rules) that are installed to prevent the establishment of a communication channel between two hosts. The attacker can abuse this method to establish the communication channel from a compromised host (at which the attacker is positioned) to the target system and send over the malicious traffic.

In our scenario illustrated in Figure 1, the attacker is located at an end host (Host A), and any packets from Host A are disallowed to be forwarded to Host C, which is the target host. However, the attacker can discover a flow rule between Host B and Host C by crafting the match fields in the probing packets that contain the network address of Host B, which is not captured by the network administrator. As a result, the attacker can leverage this uncaptured exception in network policies and send the malicious packets that flow through from Host B to Host C. Once the attacker obtains the relevant information, the attacker can initiate attacks targeting both Host B and Host C based on security postures placed in each host.

3.3. Analysis of Delay with Deep Learning

We first collect two types of RTT from probing packets sent into the data plane: *Base RTT* and *RTT with Flow Rule Installation* denoted as T_1 and T_2, respectively. T_1 is the RTT of probing flows without having any involvement of the control plane. It is used to calculate a flow setup delay after T_2 is collected. To that end, it is necessary for an attacker to send probing packets that should be responded to by some hosts.

We collect T_1 by simply sending probing packets to a host or network service (e.g., web server or data node) and measuring RTT for each packet. To collect T_2, probing packets are crafted with random or spoofed values in match fields that force the control plane to install flow rules in the data plane. After receiving the responses, we determine whether a flow rule installation has been processed by applying a t-test on the collected RTTs. In our t-test, if the p-value is less than 0.05, then we conclude that T_1 and T_2 are significantly different from each other.

After collecting the RTTs, one of the three possible outcomes is observed as follows, where R_1 and R_2 denote a set of flow rules in the match fields of probing packets and a set of flow entries in the flow tables of switches, respectively. Also, RTT$_{change}$ implies that there exists a significant delay in RTT.

(i) If $R_1 \cap R_2$ = $True$ && *Forward Packets*
Then RTT$_{change}$ = $False$
As a result, there is no change in the collected RTT value.

(ii) If $R_1 \cap R_2$ = $True$ && *Drop Packets*
Then RTT$_{change}$ = $False$
Obviously, no responses do not contribute to the RTT value.

(iii) If $R_1 \cap R_2$ = $False$ && *Install Flow Rules*
Then RTT$_{change}$= $True$
Therefore, the process of flow rule installation causes a delay that will be added to the RTT value.

After obtaining T_1 and T_2, we calculate the flow setup delay denoted as ΔT, which means ΔT = T_2 - T_1.

Even if the ΔT of SDN controllers induced by flow installation significantly varies, simply comparing the distribution pattern of flow setup delay lacks the granularity and accuracy required for determining the identity of controllers, particularly between different versions of the same controller. This is because there exist potential overlaps in the distribution of flow setup delays between different controllers and versions. In addition, a flow setup delay is not strictly static for all network environments, and thus, simple pattern matching may have biases in the final results.

To overcome this challenge, we utilize DNN models to classify the given dataset of ΔT based on the patterns of specific controllers. Specifically, we utilize three different DNN models and compare their performance—Feedforward Neural Network (FNN) [17], Convolutional Neural Network (CNN) [18], and Recurrent Neural Network (RNN) [19]. All three models can extract additional abstract features from our dataset of control plane flows, enabling robust classification capabilities that extend beyond relying solely on ΔT. In our classification, RNN is utilized to consider the temporal dynamics and historical context of the flow setup delay values (ΔT). RNN is particularly suitable for achieving accurate classification when the order and historical behaviors of the ΔT sequence are crucial.

Identifying Controllers Given the potential for multiple distributions of ΔT to exhibit similar patterns, relying solely on comparing ΔT as a single feature is insufficient. Therefore, we employ DNN models to classify the dataset, allowing us to identify distinctive features associated with the controllers. To that end, we label categorical variables such as field values, protocol, IP addresses, etc, along with ΔT. This labeling enables the DNN models to extract relevant features. We collect training data not only from a controlled lab environment but also from a campus computing infrastructure (Section 4). We split the captured dataset into training and test sets, with 30% of the data allocated for training and the remaining 70% for testing. Finally, we attempt to optimize the hyperparameters of the DNN models through random search. Unlike grid search, which exhaustively evaluates all possible combinations of hyperparameters, random search selects a random subset of parameter values. This approach not only saves computational resources but also facilitates effortless parallelization, as randomly selected samples can be trained separately.

Estimating Size of Flow Paths Once the controller is identified, our analysis extends to the input dataset from multiple flow paths. Estimating the size of flow paths follows a similar approach to identifying controllers. First, we match the distribution pattern of the collected ΔT to that of the corresponding flow paths associated with the identified controller utilizing the DNN.

Nevertheless, accurately determining the *exact* size of flow paths in terms of the number of switches poses challenges, even with the assistance of DNN models. This difficulty arises from various factors that can influence the measurement of ΔT, such as random network noise, link capacity between nodes, and data plane traffic volume. Thus, in the case where the input set of ΔT is classified into more than one flow path, we estimate the size of flow paths in the form of a numeric range. The accuracy of estimation can be improved as the number of switches in flow paths varies. This variance allows for distinguishing between different flow paths based on the increasing pattern observed in ΔT.

4. Evaluation

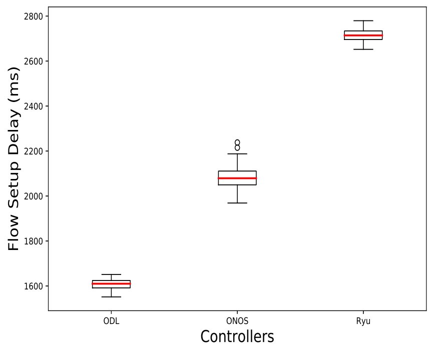
We performed our analysis in both lab and real-world SDN-based networks that are used as the research computing infrastructure [20]. The infrastructures consist of more than 500 nodes with Intel Broadwell and CascadeLake CPUs, each of which has at least 64GB RAM. All hosts in our experiment ran desktop and server version of Ubuntu 16.04 LTS. Also, it has two supercomputers with more than 34,000 CPU cores and over 580 GPU accelerators as well as 4PB data storage platform. Within this environment, the amount of traffic directed to the SDN-based network is more than 500,000 packets per day with 50 active users.

For different SDN configurations, networks of different sizes were tested using various numbers of Open Vswitch [21] and HP Openflow Switch [22], ranging from 1 to 30. For *applications*, load-balancer and L2 firewall were run on top of each controller, while network traffic was also limited to 100, 500, 1000, and 2000 packets per second for each period. The duration of each period is approximately 15 minutes, depending on the traffic rate. Due to ethical considerations, we did not collect any sensitive data that could identify or track specific individuals in our study.

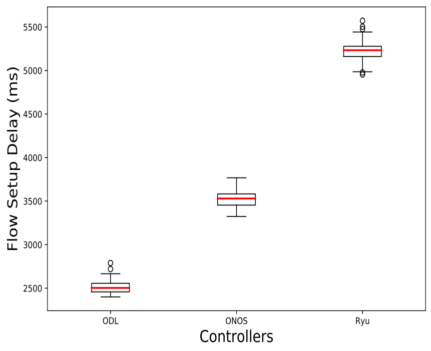
4.1. Analysis of Flow Setup Delay

Probing Control Plane We first conducted an experiment to observe if the flow setup delays incurred by three major SDN controllers were different from each other through statistical analysis. To show that flow setup delays can be leveraged to identify controllers, we first collect the delays by measuring the *Base RTT* values without invoking the control plane. We crafted packets with match fields as existed in the flow entries in flow tables. Then, we sent out 100 packets per second to a target server and measured the RTT. In an actual reconnaissance, the attacker could either send packets in a large amount to accelerate the scanning process or send packets at a low rate (for example, 10 packets per second) to evade detection. Upon receiving responses, we applied a t-test to confirm whether or not the RTT of the first packet was significantly different from the RTT value of the second packet. In our t-test, if the resulting p-value from the test was less than 0.05 then we concluded that T_1 and T_2 were significantly different from each other, meaning that flow rules corresponding to the ones in match fields of the probing packets did not exist and the control plane was invoked. Therefore, we did not use the obtained value as *Base RTT*. In other words, if there exists no significant difference in RTT, then the flow is not sent to the control plane for additional packet processing such as flow rule installation.

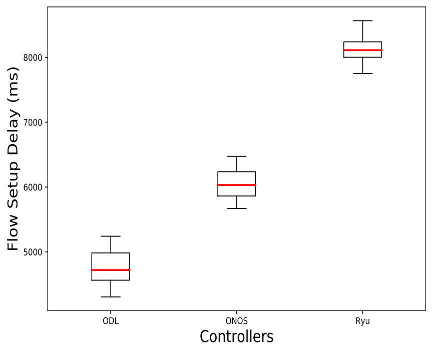
After obtaining the base RTT value, we generated and sent additional probing flows with randomly generated



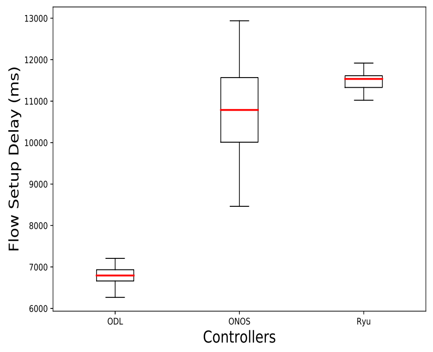
(a) Flow Setup Delay with 5 Switches



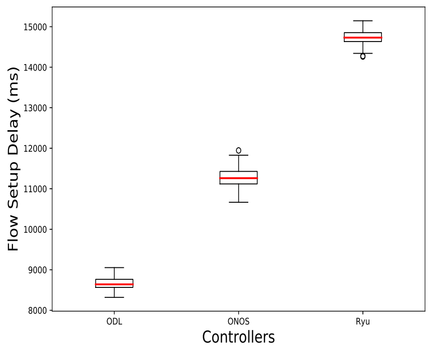
(b) Flow Setup Delay with 10 Switches



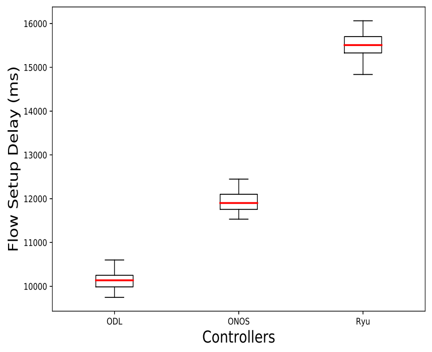
(c) Flow Setup Delay with 15 Switches



(d) Flow Setup Delay with 20 Switches



(e) Flow Setup Delay with 25 Switches



(f) Flow Setup Delay with 30 Switches

Figure 2. Flow setup delay of ODL, ONOS, and Ryu with different numbers of switches in a flow path. The performance overhead caused by the control function is reflected in the flow setup delay. As the number of switches involved in the flow path increases, the flow setup delay also increases.

match fields that triggered flow rule installation in the data plane. When the probing flow caused new flow rule installation, a significant change in RTT values between T_1 and T_2 was observed. We calculated the difference between the two RTT values, which was the flow setup delay, ΔT. After obtaining the ΔT dataset, we applied the t-test again to determine if the ΔT from each controller was significantly different from each other. We repeated the process 50 times for each controller to obtain enough empirical datasets to train the DNN models. For hyperparameters of DNN models, we used *adam* as the optimizer. We also used batch sizes of 32 or 64 and epochs ranging from 5 to 15. For CNN, kernel sizes ranged from 3 to 10.

Delays of different controllers We first measured the flow setup delays of each controller: ODL Oxygen, ONOS 1.15.0, and Ryu 4.30. Figure 2 shows the probabilistic distributions of flow setup delays (ΔT) of each controller. The result also shows the distribution of ΔT with different sizes of the network. It is evident that, as the size of a flow path increases, the flow setup delay also increases as the controller needs to install corresponding flow rules to the switches in parallel. The results show a trend of increasing patterns in ΔT as the configuration size of flow paths increases. The results also show that the distributions of upper and lower quartiles around the median value (i.e., the most frequent range of ΔT) are significantly distant from each other. Overall, we observed that the flow setup delays of ODL were the shortest, while those of Ryu were the longest. The average and median of ΔT from the controllers were distant from each other with a p-value less than 0.01.

Figure 2 also indicates that there exists a slight but distinctive performance overhead in terms of flow setup delay between the three SDN controllers as the size of the flow path increases, which can be leveraged by an attacker to fingerprint each controller. Although all major controllers take less than 16 seconds to install over 6,000 flow rules to 30 switches, the differences in the statistical distribution of flow setup delays for the controllers become more distinctive as the size of flow paths increases. In other words, the delays induced by the flow installation process from the SDN controllers are negligible in terms of *performance* and *end-user experience*, but significant enough for an attacker to fingerprint the controllers.

However, random jitters in the network can affect the collection of ΔT value[1]. Such a case is illustrated in Figure 2(d), in which the distribution of ONOS exhibits dispersion and overlaps with the distribution of RYU. Therefore, relying solely on univariate fingerprinting is not considered reliable in real-world SDNs, where multiple factors influ-

1. During our experiments, we observed that there were dispersed distributions in a few cases while most of our experiments showed distinctive distributions. The root causes for such cases include the host application and other network services running on the host OS.

TABLE 1. Accuracy, Precision, and Recall Values of FNN, CNN, and RNN for ODL, ONOS, and Ryu Controllers with 30 switches. High performance results $> 93\%$ is in bold.

		Accuracy			Precision			Recall		
Controller Name	Version	FNN	CNN	RNN	FNN	CNN	RNN	FNN	CNN	RNN
ONOS	1.15.0	88.1%	91.1%	90.0%	87.9%	89.5%	90.3%	88.6%	92.9%	89.8%
	1.13.1	89.6%	91.1%	87.1%	89.5%	89.8%	91.5%	89.7%	92.6%	82.9%
	1.11.12	89.6%	91.4%	89.8%	89.7%	91.0%	90.3%	89.4%	92.1%	89.3%
OpenDaylight	Oxygen	81.5%	**93.3%**	**93.7%**	85.5%	**93.3%**	90.6%	80.2%	**93.3%**	**93.2%**
	Carbon	83.5%	90.0%	**94.6%**	87.5%	90.3%	**93.5%**	82.0%	90.0%	**96.0%**
	Beryllium	83.6%	89.5%	92.6%	80.1%	88.7%	91.6%	88.7%	90.6%	**98.1%**
RYU	4.30	**94.6%**	**94.7%**	92.1%	92.0%	91.9%	91.8%	**95.0%**	**98.0%**	92.8%
	4.20	92.1%	**93.2%**	91.5%	91.9%	90.0%	92.8%	91.3%	**95.2%**	89.9%
	4.10	89.4%	92.0%	**95.6%**	90.0%	88.3%	90.4%	91.2%	87.9%	**96.1%**

ence the flow setup delay. We incorporate DNN to capture additional temporal relationships among data points.

Analyzing delays of different versions of controllers Upon confirming that ΔT can be leveraged in identifying the SDN controllers, we extended the same experiment to different versions of each controller to determine if flow setup delays can also be leveraged to fingerprint different versions of the controllers. We used three different versions from each controller running on a physical host—ODL (Oxygen, Carbon, Beryllium), ONOS (1.15.0, 1.13.1, 1.11.12), and Ryu (4.30, 4.20, 4.10). We collected ΔT from the three versions of each controller and compared them with the analysis results from FNN, CNN, and RNN. In our experiments, we collected more than $40,000$ packets in total to train the models. Table 1 shows the average accuracy, precision, and recall of fingerprinting SDN controllers with different versions.

For fingerprinting the name and version of an SDN controller, each DNN model generated at least 80% for accuracy, precision, and recall values as shown in Table 1. For example, the accuracy for identifying OpenDaylight Oxygen was 81.5%, 93.3%, and 93.7% for FNN, CNN, and RNN, respectively. The difference between the highest and lowest accuracy is 14.1%. Although all three models performed similarly achieving the accuracy of the lowest 81.5% from FNN, and the highest 95.6% from RNN. The high overall accuracy of RNN may be attributed to how its network functions. RNN is designed to process time-dependent sequences with a recurrent structure that allows them to maintain internal memory. The internal memory also captures long-term dependencies in the data over multiple time steps. In addition, all models showed the highest accuracy for fingerprinting the identity of RYU. Our interpretation of the result is that RYU can be distinguished from other controllers easily because its performance is clearly reflected in ΔT, compared to the other two controllers.

4.2. Analyzing the Impact of SDN Environment

Two major factors affect the performance of SDN controllers and flow setup delay—*the size of network* and *SDN applications running on the controller*. To investigate the impact of these factors on the effectiveness of our fingerprinting, we evaluated our approach in different network environments considering these two factors.

To evaluate our model comprehensively, we initially constructed a network with a scale varying from 1 to 30 switches. Each controller operates either a load-balancer, an L2 firewall, or concurrently running both. Following this setup, we sent probing flows toward multiple destinations and collected ΔT.

Fingerprinting Controller Identity (S1) As we discussed our attack scenarios in Section 3, we investigated the effectiveness of our fingerprinting with different sizes of the network. The size of the network is determined by the number of switches in the data plane. Thus, we increased the size of the network by deploying a different number of switches. Each switch was connected to a fixed number of hosts, creating its own subnet, increasing the total number of hosts at a fixed rate.

Figure 3 shows the average accuracy for different sizes of the network. As the results clearly show, the accuracy of the fingerprinting is proportional to the size of the network. For instance, the accuracy increases significantly from 67.3% to 91.4% for ONOS as the number of switches increased from 5 to 30. This is because the corresponding flow rules that need to be installed in the flow path impose additional performance overhead on the controller, which creates more distinctive temporal patterns in ΔT. The overhead subsequently translates into the processing time, which affects ΔT. Therefore, even seemingly negligible overhead is sufficient to fingerprint the identity of controllers with high accuracy, generating a unique pattern in the dataset.

In addition to different sizes of networks, we also evaluated the effect of SDN applications on the accuracy of fingerprinting. Table 2 shows the average accuracy for fingerprinting the identity of controllers with different combinations of applications. Compared to the impact of the size of the network, applications running on top of the controllers do not significantly impact the accuracy of the fingerprinting. Even though we used two applications, the results comply with findings from previous work [10].

Revealing the Sizes of Flow Paths (S2) If our fingerprinting method could identify the controller accurately, it should also be capable of inferring the size of flow paths given the fact that the size of the network has the most significant

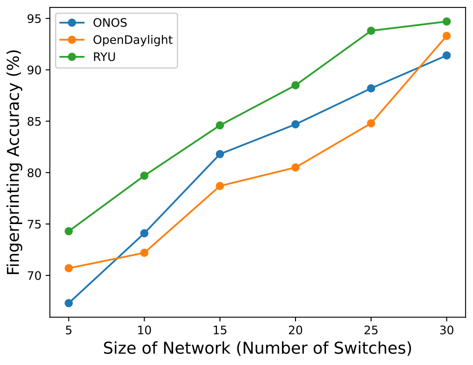


Figure 3. Accuracy of Fingerprinting SDN Controllers in Different Sizes of Network.

TABLE 2. ACCURACY OF FINGERPRINTING SDN CONTROLLERS WITH DIFFERENT APPLICATIONS.

Controller Name	Accuracy			
	No App	Firewall	Load-Balancer	Firewall & Load-balancer
ONOS	91.4%	90.3%	90.6%	88.2%
ODL	93.3%	89.6%	87.3%	88.0%
Ryu	94.7%	93.6%	90.6%	90.5%

impact on flow setup delay.

To assess this hypothesis, we used various sizes of flow paths ranging from 1 to 30 switches and measured how accurately the size of a flow path could be estimated. The results of estimating the size of flow paths are shown in Figure 4. We could distinguish one flow path from another when the difference between two flow paths was less than 5 switches. The accuracy of estimation with an acceptable margin of error was at most 79%. However, as the difference in the size of flow paths increased, the accuracy of estimation also increased up to 97% because there existed differences in both the delay itself and the distribution pattern of delay. Another observation from our experiments was that the flow setup delay to install 100 new flow rules in the data plane takes less than a second for all three controllers. Even so, the delays obtained from our experiments are significant enough to distinguish one flow path from another.

Advanced Attack (S3) Subsequently, we evaluate the advanced attack scenario explained in Section 3.2, in which we fingerprint match fields of ACL policies in each flow path. First, we set up the ACL policy in a flow path from Hosts A to C that blocked packet forwarding from Host A to C. At the same time, we installed another ACL policy in a different flow path from Hosts A to C that allowed Host B to send packets to Host C. As a result, there exists a pitfall in the ACL policy that enables an attacker to bypass the policy installed in the first flow path.

We evaluated if we could find policy conflicts in the campus network by setting up a number of flow paths between Host A and Host C with the real flow rules installed in the flow paths. The sizes of the flow paths ranged from 1 to 30, throughout 50 trials. We deliberately embedded policy conflicts in randomly selected flow paths and measured the

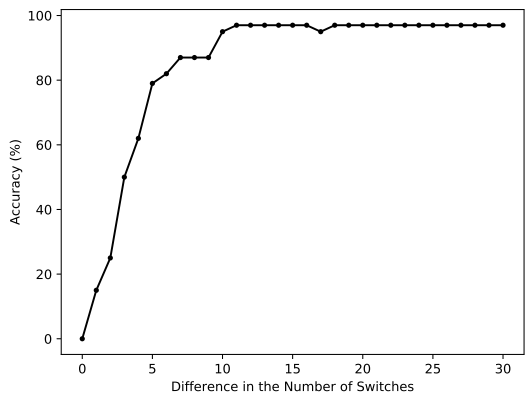


Figure 4. Accuracy of Estimating the Size of Flow Paths.

correctness and accuracy of fingerprinting. Note that we attempted to detect conflicts in a pair of flow paths. The goal of this experiment is to *demonstrate the capability of our fingerprinting method* that enables deducing valuable information such as policy conflicts, instead of checking our ability to detect exact types of policy violations. Implementing such a capability is beyond the scope of tasks in our work.

Throughout the experiment, we observed that we could detect the conflicts in ACL policies more accurately as the difference in sizes of flow paths increased. Specifically, the number of trials in which we detected conflicts significantly increased once the numeric gap of switches between the two flow paths exceeded 10. As a result, we noticed an increase in conflict detection accuracy from 14% to 94% at the line rate when the difference in the number of switches ranged from 1 to 30, respectively. The detection accuracy for policy conflicts across diverse controller versions is presented in Table 3. Furthermore, our findings highlight that the success rate of conflict detection is contingent upon the volume of ΔT data gathered.

In summary, we showed that the flow setup delays generated by three major SDN controllers were significantly different so that the attacker can utilize them to identify controllers. In addition, we observed that the size of the network affects flow setup delay. Leveraging the finding from our experiments, we also found that the attacker can estimate the size of each flow path with the highest accuracy of 97%. Lastly, from our evaluation of the advanced attack scenario, we showed that we could isolate ACL policies installed in each flow path by fingerprinting the size of the flow paths with the highest accuracy of 94% depending on the sizes of flow paths.

5. Discussion

In this section, we discuss the limitations of our fingerprinting method followed by potential mitigation strategies to defeat this attack.

Fingerprinting with a single flow path. In our experiments, we notice that the size of a flow path has a significant impact on flow setup delays. Therefore, the attacker must collect flow setup delays from multiple flow paths to compensate

TABLE 3. ACCURACY OF POLICY CONFLICT DETECTION WITH DIFFERENT SIZES OF FLOW PATH.

Size of flow path	**ODL**			**ONOS**			**Ryu**		
	Oxygen	**Carbon**	**Beryllium**	**1.15.0**	**1.13.1**	**1.11.1**	**4.30**	**4.20**	**4.10**
1	16%	18%	16%	22%	18%	14%	20%	20%	20%
5	24%	20%	22%	44%	32%	40%	50%	32%	32%
10	50%	46%	50%	80%	62%	70%	48%	42%	60%
20	88%	90%	90%	94%	86%	74%	72%	68%	60%
30	92%	90%	94%	94%	90%	88%	78%	60%	66%

for the variations caused by different sizes of flow paths. As a result, our approach will manifest a low accuracy when fingerprinting SDN controllers by collecting flow setup delays from only one flow path.

Estimating flow path sizes. While the size of a flow path greatly impacts flow setup delays, random noises in networks are inevitable, which makes it harder to accurately determine the size of any flow path in a real-world network. Therefore, we can only estimate ranges of flow path sizes instead of accurately determining the sizes. The range varies based on the collected ΔTs. It may be as low as 1 if no obvious overlaps are observed, or as high as 11 when significant overlaps exist. Nevertheless, our approach identifies each controller and its version with high accuracy since the delay measurements over backbone networks, in which SDN resides, are fairly stable without being significantly affected by congestion and traditional quality-of-service [23].

Other influential factors. The flow setup delay can also be influenced by the running instances of applications on the host that runs the controller, the performance of hardware in the network, and network loads. In this work, however, we have not taken into consideration the potential variations in processing time caused by different levels of hardware performance. Recognizing the significance of this factor, we intend to address it in future research endeavors.

Mitigation Strategies We discuss three possible countermeasures in this section — randomizing flow setup delays, randomizing flow paths, and flow setup optimization to mitigate the threat introduced in this paper.

Flow setup delay randomization. SDN administrators may randomize flow setup delays by manually increasing the delay by a few extra milliseconds, which can be performed at controllers or ingress/egress switches. Other researchers have also explored similar strategies, such as introducing random delays into control planes [24], [25], [26], [27]. However, while delay randomization can remove the distinctiveness of the fingerprints of each flow, randomization itself can also depend on random distribution. If a random distribution that follows a certain probability density function is implemented and used by an SDN controller, it can be even easier for an attacker to distinguish the pattern of delay from other traffic.

Flow path randomization. Flow path randomization, a.k.a a network randomization technique, can be employed to obstruct or increase the complexity of our attack when attempting to estimate the sizes of flow paths. It is a variant of the moving target defense, which dynamically and adaptively prevents adversaries from spying on networks [28]. SDN administrators may implement flow path randomization by utilizing an overlay network managed by SDN controllers. An alternative strategy was also proposed by Barrera et al. [29] for software-defined wide area networks (SD-WAN). They advocate for the utilization of multi-path routing to scatter the traffic over numerous network routes, thereby hampering the fingerprinting process.

Flow setup optimization. The importance of data transmission latency is growing, surpassing traditional performance requirements. This shift is crucial not only to protect against fingerprinting attacks but also to meet the increasingly stringent end-to-end latency requirements of network applications and users while ensuring quality of service (QoS). For instance, strategies such as traffic management [30], [31], congestion control [32], [33], and flow table management [34] have been explored to optimize latency. Although these techniques primarily aim to enhance network performance rather than focus on security, they can also contribute to preventing fingerprinting.

6. Related Work

Fingerprinting in SDN mostly focuses on identifying the unique characteristics and behaviors of SDN controllers and devices. This concept is rooted in the understanding that SDN controllers (and their implementations) react distinctively to various network scenarios, thereby emitting identifiable signals or patterns.

Time-based Fingerprinting in SDN The early SDN fingerprinting is explored by Shin et al. [1] and Bifulco et al. [25]. The authors demonstrated that attackers might infer SDN-enabled target networks by measuring latency due to flow table rule mismatches. This method only confirms the target network environment is SDN by assessing RTT differences between new and existing flows. Their emphasis, however, is on the discovery of SDN. In contrast, our research is inclined towards reconnaissance, gathering refined network data such as SDN controller identity and flow path dimensions.

Diverse Information Leak Analyzing timing differences in detail allows attackers to obtain more meaningful information about a target network. Sonchack et al. [3] showed that RTTs can be measured from specific destinations with packet streams, suggesting the control plane's involvement if RTTs are high. Liu et al. [26] proposed a formalized approach by modeling switch flow tables as a Markov model, enabling inference of intricate rules among complex flow rules. Yu et al. [35] also focused on switch parameters such as flow table size, cache replacement policy, and load. Achleitner et al. [36] suggested techniques for flow rule reconstruction using specially designed probing packets that mimic specific header fields to determine if they are used as match fields in flow rules.

Deep Learning-based SDN Fingerprinting Cao et al. [10] showed that attackers can analyze encrypted control traffic patterns with deep learning to infer the SDN apps running on a target controller. Seo et al. [27] inspected the exchange of traffic amongst the controllers. They studied how attackers can leverage deep learning techniques to access confidential details, including the topology and protocols in use within SD-WAN. Their work aligns closest with ours, leveraging flow processing time across different controllers.

In summary, the main difference between our approach and prior research is that we utilize the flow setup delay to fingerprint the *identity of SDN controllers* (i.e., controller name and version), while prior work leverages the time-based side channel information to fingerprint other control plane information, such as network policies and topology. In addition, our approach can also estimate sizes of the flow paths to draw a partial topology of the network.

7. Conclusion

We presented a fingerprinting attack on SDN controllers through DNN-based classification of a timing side channel. In particular, our fingerprinting scheme helped obtain the identity of the controller and different flow paths. By evaluating our fingerprinting method on major commercial-grade SDN controllers, we showed that we could effectively fingerprint the identity of SDN controllers in a real-world SDN-based computing environment with high accuracy. In addition, we discussed potential mitigation strategies to defend the identified side channel on existing SDNs.

Acknowledgments

We would like to express our deepest gratitude to Dr. Tiffany Bao, Dr. Yan Shoshitaishvili, and Dr. Adam Doupé for their invaluable contributions to this research. This work was partially supported by the grants from National Science Foundation (NSF-CICI-2232911) and Institute for Information & Communications Technology Promotion (IITP-MSIT-2017-0-00168).

References

[1] S. Shin and G. Gu, "Attacking Software-Defined Networks: A First Feasibility Study," in *2nd ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 165–166.

[2] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A Robust, Secure, and High-Performance Network Operating System," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014, pp. 78–89.

[3] J. Sonchack, A. J. Aviv, and E. Keller, "Timing SDN Control Planes to Infer Network Configurations," in *2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM, 2016, pp. 19–22.

[4] H. Cui, G. O. Karame, F. Klaedtke, and R. Bifulco, "On the fingerprinting of software-defined networks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 10, pp. 2160–2173, 2016.

[5] A. Azzouni, O. Braham, N. T. M. Trang, G. Pujolle, and R. Boutaba, "Fingerprinting OpenFlow Controllers: The First Step to Attack an SDN Control Plane," *arXiv preprint arXiv:1611.02370*, 2016.

[6] M. Zhang, J. Hou, Z. Zhang, W. Shi, B. Qin, and B. Liang, "Fine-Grained Fingerprinting Threats to Software-Defined Networks," in *Trustcom/BigDataSE/ICESS, 2017 IEEE*. IEEE, 2017, pp. 128–135.

[7] "CVE-2017-1000411," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-1000411.

[8] "CVE-2018-1999020," https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1999020.

[9] "CVE-2019-13624," https://www.cvedetails.com/cve/CVE-2019-13624/.

[10] J. Cao, Z. Yang, K. Sun, Q. Li, M. Xu, and P. Han, "Fingerprinting SDN Applications via Encrypted Control Traffic." in *RAID*, 2019, pp. 501–515.

[11] "OpenDaylight," https://www.opendaylight.org/.

[12] "Open Network Operating System," https://onosproject.org/.

[13] "Ryu SDN Controller," https://ryu-sdn.org/.

[14] "What is a ryu controller?" https://www.sdxcentral.com/networking/sdn/definitions/what-the-definition-of-software-defined-\\networking-sdn/what-is-sdn-controller/openflow-controller/what-is-ryu-controller/.

[15] "Openflow match fields," http://flowgrammable.org/sdn/openflow/message-layer/match/.

[16] "Dynamic Configuration Subsystem in ONOS," https://wiki.onosproject.org/display/ONOS/Dynamic+Configuration+Subsystem.

[17] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *Ieee Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[18] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[19] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, pp. 64–67, 2001.

[20] E. Dart, L. Rotman, B. Tierney, M. Hester, and J. Zurawski, "The science dmz: A network design pattern for data-intensive science," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–10.

[21] "Open Vswitch," https://www.openvswitch.org/.

[22] "Introduction to HP Openflow Switch," https://techhub.hpe.com/eginfolib/networking/docs/switches/K-KA-KB/15-18/5998_8148_ssw_admin_guide/content/c_Introduction.html.

[23] A. Markopoulou, F. Tobagi, and M. Karam, "Loss and delay measurements of internet backbones," *Computer communications*, vol. 29, no. 10, pp. 1590–1604, 2006.

[24] J. Sonchack, A. Dubey, A. J. Aviv, J. M. Smith, and E. Keller, "Timing-based Reconnaissance and Defense in Software-Defined Networks," in *32nd Annual Conference on Computer Security Applications*. ACM, 2016, pp. 89–100.

[25] R. Bifulco, H. Cui, G. O. Karame, and F. Klaedtke, "Fingerprinting software-defined networks," in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*. IEEE, 2015, pp. 453–459.

[26] S. Liu, M. K. Reiter, and V. Sekar, "Flow reconnaissance via timing attacks on SDN switches," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017, pp. 196–206.

[27] M. Seo, J. Kim, E. Marin, M. You, T. Park, S. Lee, S. Shin, and J. Kim, "Heimdallr: Fingerprinting SD-WAN Control-Plane Architecture via Encrypted Control Traffic," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 949–963.

[28] A. R. Chavez, W. M. Stout, and S. Peisert, "Techniques for the dynamic randomization of network attributes," in *2015 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2015, pp. 1–6.

[29] D. Barrera, L. Chuat, A. Perrig, R. M. Reischuk, and P. Szalachowski, "The SCION Internet architecture," *Communications of the ACM*, vol. 60, no. 6, pp. 56–65, 2017.

[30] B. Li, T. Wang, P. Yang, M. Chen, and M. Hamdi, "Rethinking data center networks: Machine learning enables network intelligence," *Journal of Communications and Information Networks*, vol. 7, no. 2, pp. 157–169, 2022.

[31] M. Hayes, B. Ng, A. Pekar, and W. K. Seah, "Scalable architecture for sdn traffic classification," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3203–3214, 2017.

[32] K. Lei, Y. Liang, and W. Li, "Congestion control in sdn-based networks via multi-task deep reinforcement learning," *IEEE Network*, vol. 34, no. 4, pp. 28–34, 2020.

[33] J. Hwang, J. Yoo, S.-H. Lee, and H.-W. Jin, "Scalable congestion control protocol based on sdn in data center networks," in *2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[34] Y.-C. Wang, Y.-D. Lin, and G.-Y. Chang, "Sdn-based dynamic multipath forwarding for inter–data center networking," *International Journal of Communication Systems*, vol. 32, no. 1, p. e3843, 2019.

[35] M. Yu, T. Xie, T. He, P. McDaniel, and Q. K. Burke, "Flow table security in SDN: Adversarial reconnaissance and intelligent attacks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 6, pp. 2793–2806, 2021.

[36] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial network forensics in software defined networking," in *Proceedings of the Symposium on SDN Research*, 2017, pp. 8–20.