Fingerprinting SDN Applications via Encrypted Control Traffic

Jiahao Cao[1,2,3,4], Zijie Yang[1,2,4], Kun Sun[3], Qi Li[2,4],
Mingwei Xu[1,2,4], and Peiyi Han[5]

[1]*Department of Computer Science and Technology, Tsinghua University*
[2]*Institute for Network Sciences and Cyberspace, Tsinghua University*
[3]*Department of Information Sciences and Technology, George Mason University*
[4]*Beijing National Research Center for Information Science and Technology, Tsinghua University*
[5]*School of Computer Science, Beijing University of Posts and Telecommunications*

Abstract

By decoupling control and data planes, Software-Defined Networking (SDN) enriches network functionalities with deploying diversified applications in a logically centralized controller. As the applications reveal the presence or absence of internal network services and functionalities, they appear as black-boxes, which are invisible to network users. In this paper, we show an adversary can infer what applications run on SDN controllers by analyzing low-level and encrypted control traffic. Such information can help an adversary to identify valuable targets, know the possible presence of network defense, and thus schedule a battle plan for a later stage of an attack. We design deep learning based methods to accurately and efficiently fingerprint all SDN applications from mixed control traffic. To evaluate the feasibility of the attack, we collect massive traces of control traffic from a real SDN testbed running various applications. Extensive experiments demonstrate an adversary can accurately identify various SDN applications with a 95.4% accuracy on average.

1 Introduction

As a promising network paradigm, Software-Defined Networking (SDN) has attracted much attention from both industry and academia. It is being widely deployed in real-world environments, such as cloud networks [5], data centers [28], and next-generation mobile networks [7]. SDN separates control and data planes with a logically centralized SDN controller managing the whole network. A wide range of innovative applications are deployed in the controller to enable diversified network functionalities, such as load balancing [10], denial-of-service (DoS) attacks detection [24, 71], and network security forensics [61]. They call *high-level* application programming interfaces (APIs) provided by the controller to build their control logic. The controller translates the API calls into *low-level* control traffic, e.g., OpenFlow [8] traffic, to enforce network policies in SDN switches. To prevent potential attacks, control traffic is usually encrypted with the transport layer security (TLS) protocol [8].

SDN applications provide various network services and functionalities in the network and appear as black-boxes by design to switches. Therefore, network users do not know what applications are running on controllers. It is critical for attackers to be aware of what applications are running on the controller before launching their attacks. Attackers may leverage this information to identify valuable targets, understand the presence of network defense, and develop a battle plan for a future attack. For example, if attackers know there is no TopoGuard [27] security application in SDN, a topology poisoning attack can be directly launched to hijack network flows [27]. In contrast, if attackers detect the presence of TopoGuard, they can customize their attack plan to bypass the defense, e.g., leveraging *Port Amnesia* [56].

In this paper, we show that what applications are running on SDN controllers can be inferred by analyzing low-level control traffic even if the traffic is encrypted. The key insight behind our inference attack is that different SDN applications call APIs with different behaviors, which results in diverse patterns of control traffic. For example, `Anonymous Communication` [42] periodically rewrites action fields of flow rules with `FLOW_MOD` control messages, while `Traffic Monitor` [6] periodically collects flow statistics from flow rules with `STATS_REQUEST` and `STATS_REPLY` control messages. The number of packets, the length of packets, and the ratio of incoming and outgoing packets for the control traffic of the two applications are all significantly dissimilar. Such patterns still exist though the control traffic is encrypted. To our best knowledge, inferring applications running on controllers has not been considered so far as a potential attack vector in SDN. Previous studies [12, 21, 32, 39, 52, 57] focus on fingerprinting SDN networks, host communication patterns, and composition of flow rules by actively sending probing packets. Our work here fingerprints SDN applications by passively analyzing control traffic without sending any packets.

Nevertheless, we face two challenging problems to successfully fingerprint SDN applications as follows:

- How to accurately characterize the pattern of control traffic for an application?
- How to efficiently identify multiple applications with mixed control traffic?

For the first problem, the key challenge is that high-level SDN applications generate massive, encrypted, and low-level network control packets, which results in labor-intensive, time-consuming, and difficult manual analysis for characterizing the patterns of control traffic. Particularly, complicated SDN applications call many types of APIs, which results in overlaps of API calls between different SDN applications. For example, `Load Balancer` [4] generates `STATS_REQUEST` and `STATS_REPLY` control messages to calculate the throughput of a flow, and leverages `FLOW_MOD` control messages to determine the port for forwarding the flow. However, the above three types of control messages are also partly generated by `Traffic Monitor` [6] and `Anonymous Communication` [42]. Moreover, there are some identical control packets for different applications, which further increases the difficulty to characterize the patterns of control traffic.

To address the problem, we transform network control packets into a time series and apply deep learning to automatically extract patterns for different applications from it. We try to maintain raw information of control traffic in the time series as much as possible to improve the accuracy of pattern extraction. Specifically, each element in the time series denotes a packet, and the value of an element is the packet length. When a packet is sent from controllers to switches, the corresponding element is multiplied by -1. Besides, the order of elements in the time series is consistent with the order of packets appearing in control traffic. Consequently, most raw information of control traffic is naturally encoded into the time series, such as the lengths of packets, the directions of packets, etc. Thus, the time series can be directly fed into deep neural networks for accurate and automatic feature extraction. Although the contents and delays of packets are missed, they are unhelpful to characterize the patterns of control traffic considering that the packets are encrypted and their delays are usually changeable.

For the second problem, the key challenge is that one single TCP connection between a controller and a switch contains control packets generated by multiple applications concurrently running on the controller. An adversary cannot separate the control traffic of an application from the mixed control traffic to infer what it is. A naive method is to train a deep neural network with all possible combinations of mixed traffic to build a classifier that gives the compositions of applications. However, the number of combinations exponentially grows with more applications. Thus, the deep neural network quickly becomes exceedingly complicated for classifying the exponential combinations of applications. It is extremely time-consuming and not scalable to train such a complicated deep neural network.

Fortunately, we can solve the problem by dividing it into several subproblems. We train multiple classifiers for multiple SDN applications. Each classifier solves a 2-class classification problem, i.e., whether mixed control traffic contains traffic of an application or not. The training samples for each classifier are two types of mixed control traffic that includes or excludes traffic of an application. Thus, the structure of deep neural networks is simplified and each classifier can be trained in parallel, which significantly reduces the training time. By merging the output results of all classifiers, we know what applications run on controllers.

We conduct experiments in a real SDN testbed consisting of commercial hardware switches and a popular open source controller. We deploy 10 SDN applications on the controller, ranging from network performance optimization to network monitor and network security enhancements. We collect about $6,000,000,000$ control packets with different combinations of applications and translate them into many time series of equal lengths. We systematically explore three state-of-the-art deep learning models, i.e., Convolutional Neural Network (CNN) [34], Long Short-Term Memory (LSTM) [26], and Stacked Denoising Autoencoder (SDAE) [59], to train classifiers for fingerprinting SDN applications with time series. The results show that CNN performs the best, which achieves an average accuracy of 95.4% to fingerprint different SDN applications. Besides, we find that the accuracy can be further improved by increasing the length of a time series.

We summarize our key contributions as follows:

- We uncover a new attack vector in SDN, which allows an adversary to infer what applications are running on an SDN controller by analyzing low-level and encrypted control traffic.
- We develop techniques to accurately and efficiently fingerprint SDN applications with mixed control traffic.
- We collect a large dataset of control traffic from a real SDN testbed and systematically evaluate the feasibility of fingerprinting SDN applications with it.

The rest of the paper is organized as follows: Section 2 introduces background on SDN and deep learning. Section 3 provides our techniques to fingerprint SDN applications. Section 4 describes data collection methods and datasets. Section 5 evaluates the effectiveness of fingering SDN applications. Section 6 discusses our current limitations and possible countermeasures against fingerprinting SDN applications. Section 7 reviews related work and Section 8 concludes the paper.

2 Background

In this section, we briefly introduce the necessary background of SDN and deep learning.

2.1 SDN

Software-Defined Networking (SDN) is an emerging programmable network framework that decouples control and data planes. As shown in Figure 1, SDN consists of three main layers: an application layer, a control layer, and a data plane layer. Multiple applications concurrently run in the application layer. They obtain a highly abstracted view of the network and make network policies by calling APIs provided by the control layer. The control layer manages installed network applications and establishes connections with network switches in the data plane layer. It translates API calls of applications into low-level control messages to tell switches on how to forward and process packets.

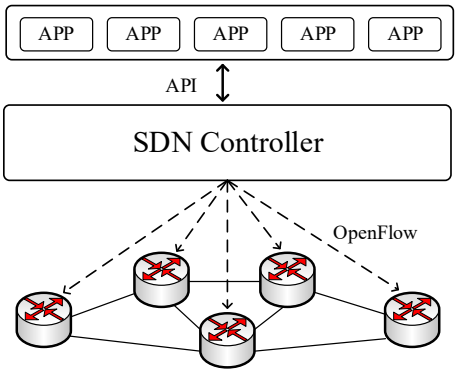


Figure 1: The framework of SDN.

The standardized communication protocol between the control layer and the data plane player is OpenFlow [8]. OpenFlow also specifies functions of SDN switches and enables controllers to manage switches in an open, vendor-neutral, and interoperable way. It defines various control messages to enable diversified functionalities, such as device capabilities advertisement, packet forwarding control, flow statistics reporting, and network events notification. We summarize main control messages and their functionalities in Table 1. Furthermore, as control messages contain sensitive network information and critical network decisions, they are usually encrypted with the TLS protocol.

2.2 Deep Learning

Deep learning has made amazing achievements in many aspects, such as speech recognition, natural language processing, and face recognition. With the support of sufficient data, models with deep structure fit data well and thus can be applied to multiple tasks, such as classification and prediction. Compared to traditional machine learning that requires designing a sophisticated feature extractor with expert experience, deep learning adopts a universal learning method to automatically extract features from massive data, which avoids the heavy workload of manually designing features. Different types of deep neural networks (DNNs) have been designed by researchers for different purposes. Out of all existing types of DNNs, we explore three popular types of DNNs to fingerprint SDN applications.

Table 1: Main Control Messages in OpenFlow.

Category	Message	Functionality
State Modification	`FLOW_MOD`	Modify rules in different tables to control packet forwarding and processing.
	`GROUP_MOD`	
	`METER_MOD`	
Statistics Collection	`FLOW_STAT`* †	Collect various flow statistics measured by rules in different types of tables and ports of switches.
	`GROUP_STAT`*	
	`METER_STAT`*	
	`PORT_STAT`*	
Device Configuration	`SWITCH_CONFIG`	Set and query configuration parameters in switches.
	`TABLE_CONFIG`	
Capability Announcement	`HANDSHAKE`	Identify SDN switches and query their capabilities that different tables support.
	`TABLE_FEATURE`	
	`GROUP_FEATURE`	
	`METER_FEATURE`	
Event Notification	`PACKET_IN`	Notify network events to controllers, e.g., new flows arriving, and send data packets to switches.
	`PORT_STATUS`	
	`FLOW_REMOVED`	
	`PACKET_OUT`	
Liveness Verification	`ECHO_REQUEST`	Verify liveness and conduct customized measurements.
	`ECHO_REPLY`	

* A pair of request and reply messages.
† `FLOW_STAT` can also be used to know all flow rules in switches.

Convolutional Neural Network (CNN). CNN [34] has been widely used in computer vision systems. It contains an input layer, an output layer, and multiple hidden layers that are convolutional layers, pooling layers, and fully-connected layers. Convolutional layers perform a convolution operation to the input and create feature maps that contain abstract features. Pooling layers reduce the dimensions of data by downsampling. CNN typically contains several convolutional and pooling layers to extract more abstract features. Fully-connected layers perform final classification with output feature maps. CNN can well characterize the spatial relationship of data and search for the most important local features. As the positions of SDN control packets in a network flow have strong space relationship due to control logic of applications and may have evident local features, CNN is suitable to characterize patterns of control traffic.

Long Short-Term Memory (LSTM). LSTM [26] is a variant of Recurrent neural network (RNN) that uses feedback connections to store representations of recent input events. LSTM improves RNN for learning long-term dependency in-

formation of sequences and avoiding the problem of vanishing gradient. A common LSTM unit consists of a memory cell, an input gate, an output gate, and a forget gate. The cell is responsible for remembering information over arbitrary time intervals so that it can keep track of the dependencies between the elements in the input sequence. The three gates regulate the flow of information into and out of the cell, deciding whether to let the information in, whether to produce the output, and whether to forget the information. Due to the network structure, it captures temporal dependencies between data. LSTM may be suitable to process control packets of SDN applications since control packets naturally have temporal dependencies.

Stacked Denoising Autoencoder (SDAE). Autoencoder (AE) is a special feedforward neural network, consisting of an input layer, a hidden layer, and an output layer. The input layer and the hidden layer act together as an encoder that compresses data from the input layer into a low-dimensional representation. The hidden layer and the output layer act together as a decoder that reconstructs the data back, i.e., decompressing the representation into something that closely matches the original data. Stacked Denoising Autoencoder (SDAE) [59] stacks multiple AEs together to form a deep network architecture and adds noise to the input data, which makes the network robust. SDAE learns meaningful data representations. Particularly, we may get low-dimensional and highly compressed representations of control traffic to fingerprint SDN applications with SDAE.

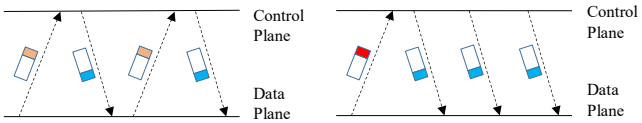
3 Fingerprinting SDN Applications

In this section, we first present the threat model and the key insight for fingerprinting SDN applications. We then introduce practical challenges and design methods to solve them.

3.1 Threat Model

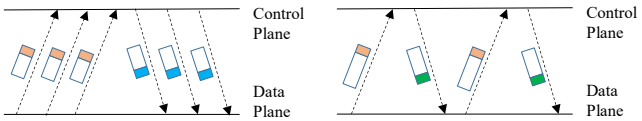
In our threat model, we consider control traffic between an SDN controller and a switch is protected with TLS/SSL. We assume an adversary can eavesdrop control traffic between the controller and a switch. Attackers may eavesdrop SDN control traffic in different ways [13, 15–17, 33, 49, 68], such as conducting ARP poisoning for switches and controllers to make control traffic first pass a listening host [17], placing a device between controllers and switches to intercept control traffic [16] [1], intercepting a forwarding link to eavesdrop control traffic [13], and dumping control traffic through listening mode of switches [15]. Particularly, Yoon et al. [68] demonstrate the feasibility of eavesdropping control traffic with real experiments. We do not require an adversary know payloads of control packets that are usually encrypted.

[1] SDN control traffic may be carried by an inherently adversarial Internet Service Provider (ISP) [16].



(a) `Learning Switch`. (b) `Anonymous Comm.`

Figure 2: Patterns of control packets for `Learning Switch` and `Anonymous Communication`.



(a) Changed patterns for `Learning Switch`. (b) Similar patterns like `Learning Switch` for `ARP Proxy`.

Figure 3: Examples to illustrate that accurately characterizing patterns of control packets for an application is difficult.

Moreover, an adversary may not insert, modify, delay or drop control packets. We do not assume SDN controllers, applications, or switches are compromised by an adversary.

3.2 Key Insight and Challenges

We first give intuitive examples to illustrate our key insight on fingerprinting SDN applications, i.e., different applications generate different patterns of control traffic due to their inherent control logic. Figure 2a and Figure 2b show the patterns of control packets for `Learning Switch` and `Anonymous Communication`. The patterns for the two applications are significantly different. `Learning Switch` receives a `PACKET_IN` message (orange packets) to analyze a packet for a flow and sends back a `FLOW_MOD` message (blue packets) to install flow rules on how to forward the packets for the flow. Consequently, the control traffic of `Learning Switch` consists of multiple pairs of `PACKET_IN` and `FLOW_MOD` messages. However, `Anonymous Communication` periodically inspects all flow rules in a switch with a `FLOW_STAT` message (red packets). After that, it sends multiple `FLOW_MOD` messages to rewrite actions of flow rules to modify packet headers for anonymous communication. Different control logic of the two applications results in different patterns of control traffic in many aspects, i.e., packet lengths, directions of packets, relative orders between packets, etc. The patterns still exist even if controllers encrypt control traffic with TLS/SSL. Therefore, an adversary can fingerprint SDN applications by analyzing patterns of control traffic.

However, there are two key challenges in real SDN environments. The first challenge is how to accurately characterize the pattern of the control traffic for an application. The control traffic is low-level and encrypted, which leads to a hard description of patterns of control traffic for differ-

ent applications. Particularly, patterns of control traffic for some applications are mutable due to different network flows in switches. As shown in Figure 3a, the pattern of control traffic for `Learning Switch` changes if massive new flows come quickly. The reason is that the application is busy processing the burst `PACKET_IN` messages. It takes some time to respond to the messages. Besides, different applications may have similar patterns of control traffic. For example, Figure 3b shows that control traffic for `ARP Proxy` consists of multiple pairs of `PACKET_IN` and `PACKET_OUT` messages (green packets). It looks similar to control traffic in Figure 2a since we cannot know the content of the encrypted packets. We just see there are many pairs of uplink and downlink packets both in Figure 2a and 3b. The only difference here is that `ARP Proxy` has larger uplink and downlink control messages. We need a method that can generalize well to accurately characterize the patterns of control traffic for different applications.

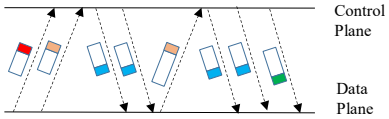


Figure 4: Mixed control packets with `Learning Switch`, `Anonymous Communication`, and `ARP Proxy`.

The second key challenge is that control traffic for multiple SDN applications is mixed in a single TCP connection between a controller and a switch. We cannot easily divide mixed control traffic into multiple types of pure control traffic for identifying each application in turn. Figure 4 shows the mixed control packets with three applications. Actually, the types of packets (colors in packets) are not known from the adversary's view due to encryption. It is difficult to infer which packets belong to an application especially when there are some identical packets, i.e., blue packets in Figure 4. Furthermore, control traffic becomes more complicated with more applications running on controllers. Although we may infer the compositions of applications for one time without dividing mixed control traffic, the number of the compositions exponentially grows with the number of applications. We need a method that can efficiently identify multiple applications with mixed control traffic.

3.3 Methodology

3.3.1 Packet Transformation

To accurately characterize the pattern of control traffic for an application, we apply deep learning since it can automatically extract features and conduct classifications from enough datasets. Moreover, classification models trained by deep learning can achieve a good generalization ability. However, we cannot directly feed SDN control packets into neural networks. The reason is two-fold. First, each bit in control packets is encrypted, which does not maintain the original information. Feeding full packets into neural networks may significantly reduce the accuracy of fingerprinting applications. Second, the size of control packets can be large, e.g., up to 12144 bits in Ethernet-based networks. Training deep neural networks with massive large packets is time-consuming.

In order to efficiently train an accurate classifier, we try to maintain useful information and remove unnecessary information in control packets. We transform control packets into a time series that can be the raw input for deep neural networks to automatically extract features and build classifiers. Formally, consider $\mathbf{a_i}$ is the i-th control packet in a packet series $\mathbf{S} = [\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_m}]$. We transform $\mathbf{S}$ into $S = [f(\mathbf{a_1}), f(\mathbf{a_2}), .., f(\mathbf{a_m})]$. Here, $f(\mathbf{a_i})$ is a transformation function that maps a control packet into a real number. It is defined as follows:

$$f(\mathbf{a_i}) = \begin{cases} |\mathbf{a_i}|, \ if \ \mathbf{a_i} \ is \ sent \ to \ controllers \\ -|\mathbf{a_i}|, \ if \ \mathbf{a_i} \ is \ sent \ to \ switches \end{cases} \quad (1)$$

Here, $1 \leq i \leq m$ and $|\mathbf{a_i}|$ denotes the length of the packet $\mathbf{a_i}$. Although the transformation process is simple and fast, useful information for SDN application classification is naturally encoded into the time series. The lengths of control packets are denoted by the absolute values of the numbers in the time series, the directions of packets are denoted by the signs of the numbers, and the relative orders of packets are denoted by the positions of the numbers. Thus, we can directly feed each time series into deep neural networks to conduct pattern extraction. Although the encrypted payloads of packets and inter-packet delays are lost in the time series, we consider they are little helpful for identifying an SDN application.

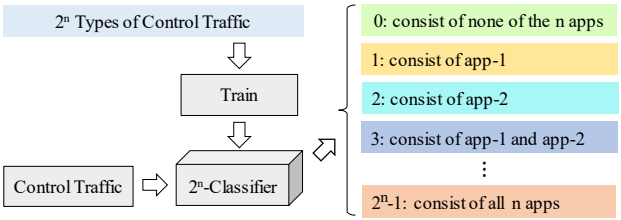
3.3.2 Task Decomposition

Our task is to identify multiple applications that concurrently run on SDN controllers with mixed control traffic. As the mixed control traffic cannot be split, a naive method is to train deep neural networks with all possible combinations of control traffic to build a multi-class classifier that gives the compositions of applications running on SDN controllers. Formally, assume that there are n possible applications running on a controller and control traffic trace is denoted by $\mathbf{t}$, we aim to give a classifier C that assigns a label c to $\mathbf{t}$, where $c \in [0, 1, .., 2^n - 1]$, i.e., possible combinations of n SDN applications. As shown in Figure 5a, if we directly infer the compositions of possible n applications running on a controller, a classifier should output 2^n types. We need to build a deep neural network with a large amount of parameters to classify the types of exponential scales. Therefore, the neural network quickly becomes exceedingly com-

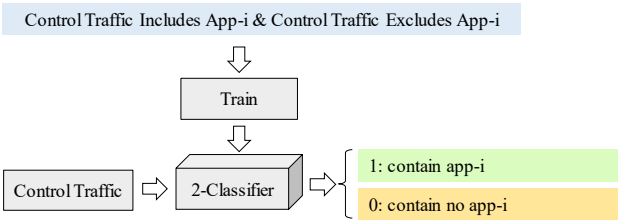
Table 2: SDN Applications in Our Testbed and Corresponding Control Messages.

SDN Applications		Main Control Messages							
		FLOW_MOD	GROUP_MOD	FLOW_STAT	GROUP_STAT	PORT_STAT	PACKET_IN	PACKET_OUT	ECHO
Basic Network Functionalities	Topology Discovery* [11]	×	×	×	×	×	✓	✓	×
	Learning Switch† [3]	✓	×	×	×	×	✓	×	×
	ARP Proxy† [1]	×	×	×	×	×	✓	✓	×
Network Monitor	Traffic Monitor† [6]	×	×	✓	×	✓	×	×	×
	Link Delay Monitor* [31]	×	×	×	×	×	✓	✓	✓
Network Opt.	Load Balancer† [4]	✓	✓	✓	×	✓	✓	×	×
Security and Privacy Enhancement	TopoGuard* [27]	×	×	×	×	×	✓	✓	×
	DoS Detection* [71]	✓	✓	✓	✓	×	✓	✓	×
	Anonymous Comm* [42]	✓	×	✓	×	×	×	×	×
	Scan Detection*‡ [41]	✓	×	×	×	×	✓	✓	×

* For applications without source code, we implement and run them on Floodlight according to papers.
† For applications with source code, we directly run them on floodlight.
‡ We implement Scan Detection with the TRW-CB algorithm in the paper [41].



(a) A model of deep learning for the original task.



(b) A model of deep learning for a subtask.

Figure 5: Task Decomposition.

plicated with many applications. Training such a classifier is time-consuming and not scalable.

Thus, in order to efficiently identify multiple applications, we divide the original task into several subtasks. We train n binary classifiers $[C_1, C_2, ..., C_n]$ for n applications. Each classifier $C_{i\ (1\leq i\leq n)}$ only answers if the mixed control traffic **t** contains control traffic of the i-th application. As shown in Figure 5b, we feed two types of control traffic, i.e., control traffic including and excluding the i-th application, to train the classifier C_i. Thus, the architecture of neural networks is simplified since a classifier only outputs two classes, i.e., containing control traffic of the i-th application or otherwise. Moreover, each classifier is independent and can be trained in parallel to reduce total training time. By testing mixed control traffic in each well-trained classifier, we can know what applications run on controllers.

4 Data Collection

To successfully build an accurate classifier, enough training data is required for deep learning to learn underlying patterns and enable good generalization to unseen samples. As far as we know, there are no public traces of SDN control traffic. In this section, we provide the method of collecting the traces and introduce the dataset.

4.1 Data Collection Methodology

We build a real SDN testbed with five commercial hardware SDN switches, Edgecore AS4610-54T, and a popular open source controller, Floodlight. We deploy the controller on a server with a quad-core Intel Xeon CPU E5504 and 32GB RAM. We attach one host on each switch. Each host has a dual-core Intel i3 CPU and 4GB RAM. All hosts in our experiments run Ubuntu 16.04 LTS. In order to generate real data traffic in our testbed, we inject real traffic traces from CAIDA [2] with TCPReplay.

We deploy 10 SDN applications on the controller, ranging from basic network functionalities, advanced network performance optimization, network monitor to security and privacy enhancements. We list these applications and their main control messages in Table 2, which implement representative network functionalities. Topology Discovery dynamically discovers switches and links between the switches and the controller. Learning Switch learns the mappings be-

tween MAC addresses and switch ports, and forwards packets according to the mappings, which makes SDN switches act as layer-2 switches. `ARP Proxy` provides a MAC address of a host to answer an ARP query for an IP address. `Traffic Monitor` and `Link Delay Monitor` monitor network throughput and link delays to provide necessary information for other applications, respectively. `Load Balancer` optimally schedules the workloads across multiple computing resources. The above six applications are bundled applications in most controllers, including Floodlight, ONOS, and OpenDaylight. Thus, we choose them as typical applications to evaluate the effectiveness of our method on fingerprinting applications. The other four applications are to enhance the security and privacy of SDN [27, 41, 42, 71]. `Topoguard` fixes a vulnerability of topology poisoning that widely exists in SDN controllers. `DoS Detection` applies SDN based methods to detect the DoS attack that is one of the most powerful attacks to disrupt a company or an organization. `Anonymous Communication` provides strong anonymity guarantees for communications in SDN. `Scan Detection` enables prominent traffic anomaly detection algorithms with SDN to effectively identify malicious activities of hosts.

The types of control messages between these applications are overlapped. However, the control traffic still has different underlying patterns, such as packet length, contexts between packets, etc. We consider the applications as suitable tests for deep learning both in the coverage of different applications and diversified control traffic. We write a shell script to automatically combine different applications to run on controllers. We leverage `tcpdump` on the controller's host to capture TCP packets with the 6653 port (OpenFlow port) between the Floodlight controller and a switch. Note that we only leverage the above method to collect control packets for training deep learning models. In the attacking phase, since it is almost impossible for an attacker to run `tcpdump` on the controller to collect control packets, an attacker should collect them using methods mentioned in Section 3.1. We save each captured control traffic for one combination of SDN applications into a text file. We write a `Python` program to automatically label all control packets in each text file according to the combination of the applications. Due to storage constraints, we only save extracted metadata from traces of control traffic. The metadata consists of the capture time of packets, the directions of packets, and the lengths of packets. We discard encrypted payloads of packets since they have little value for an adversary. We remove TCP acknowledgment (ACK) packets containing no control messages.

4.2 Dataset

Our dataset contains 6,000,000 control packets for each combination of the 10 applications. Each type of control traffic for one combination is saved in a separated text file. Totally, there are 6,144,000,000 control packets and 1024 text files. Our current dataset only contains control packets between one switch and the controller. Although collecting more control flows between multiple switches and the controller may help to improve the accuracy of fingerprinting applications, we aim to study the accuracy of fingerprinting in a generalized case since eavesdropping one control flow for an adversary is easy.

5 Evaluation

In this section, we conduct comprehensive experiments to verify the feasibility of fingerprinting SDN applications. We first evaluate the accuracy, precision, and recall rate for 10 applications with three popular deep learning models. Then, we explore how the effectiveness changes with different split lengths of control traffic and different number of datasets. Finally, we evaluate the training time for building a classifier to fingerprint an SDN application.

5.1 Experiment Setup

We implement three models, i.e., SDAE, LSTM, and CNN, for each of the 10 applications with `Keras` in Python. We train each model on a server equipped with one Intel Xeon Silver 4116 CPU (12 cores), 128 GB RAM, 1TB SSD, and NVIDIA Quadro P4000 GPUs. To train a model for an application, we divide all traces of mixed control traffic into two classes: the first contains control traffic of the application and the second contains no control traffic of the application. We randomly select 60% samples from both the classes as the training set, 20% samples as the test set, and 20% samples as the validation set. We initially define a sequence of 150 control packets as one sample. Moreover, we change the length of one sample to explore how different split lengths affect the effectiveness of fingerprinting SDN applications.

In order to accurately fingerprint SDN applications, hyperparameters of each model should be well tuned so that models have the best classification performance and generalize well to unseen traffic traces. Although conducting an exhaustive grid search or other search algorithms is effective, it is computationally expensive. In our experiment, we semi-automatically tune hyperparameters. We first conduct a grid search with a small dataset, i.e., one-tenth of the original dataset, to know the impacts of each hyperparameter. We next manually adjust parameters with the original dataset based on our experience and experimental results. We list our final parameters in Appendix A.

5.2 Effectiveness

Effectiveness with Different Models. We initially set the split length of a sample as 150. Table 3 shows the accuracy, recall rate, and precision of fingerprinting SDN ap-

Table 3: The Effectiveness of Fingerprinting SDN Applications with different DNN Models.

SDN Applications	SDAE			LSTM			CNN		
	Accuracy	**Recall**	**Precision**	**Accuracy**	**Recall**	**Precision**	**Accuracy**	**Recall**	**Precision**
`Topology Discovery`	90.8%	90.6%	94.1%	92.7%	**95.7%**	92.5%	**94.2%**	89.0%	**99.3%**
`Learning Switch`	96.4%	**97.3%**	**99.6%**	**98.3%**	92.7%	88.8%	96.4%	90.3%	99.5%
`ARP Proxy`	87.1%	83.7%	83.9%	92.2%	**95.7%**	88.1%	**94.3%**	92.8%	**90.8%**
`Traffic Monitor`	**94.4%**	**96.5%**	92.5%	92.8%	94.1%	91.7%	90.6%	93.0%	**97.9%**
`Link Delay Monitor`	93.4%	92.9%	94.6%	93.2%	98.5%	84.5%	**96.5%**	**99.3%**	**95.2%**
`TopoGuard`	94.8%	94.8%	97.5%	95.4%	**98.5%**	83.5%	**95.7%**	92.0%	**98.7%**
`Load Balancer`	93.1%	91.9%	87.2%	90.6%	93.1%	85.4%	**97.1%**	**95.3%**	**97.3%**
`DoS Detection`	89.6%	90.2%	88.7%	94.3%	90.0%	90.3%	**97.8%**	**97.9%**	**96.3%**
`Anonymous Comm`	**98.2%**	**97.4%**	**97.5%**	98.1%	94.9%	83.8%	94.7%	89.3%	92.6%
`Scan Detection`	95.6%	94.5%	87.7%	94.2%	**96.6%**	94.2%	**96.8%**	94.0%	**98.7%**
Average Value	93.3%	93.0%	92.3%	94.2%	95.0%	88.3%	95.4%	93.3%	96.6%
Standard Deviation	3.2%	4.0%	5.0%	2.4%	2.5%	3.7%	2.0%	3.3%	2.8%

plications. For an application, different models perform differently. For example, the accuracy for identifying `ARP Proxy` is 87.1%, 92.2%, and 94.3% for SDAE, LSTM, and CNN, respectively. The difference between the highest accuracy and the lowest accuracy is 7.2%. The recall rate and precision also change with different models. Moreover, SDAE performs best for `Anonymous Communication` with a 98.2% accuracy, LSTM performs best for `Learning Switch` with a 98.3% accuracy, and CNN performs best for `DoS Detection` with a 97.8% accuracy. Our interpretation is that different models have different capabilities to characterize underlying patterns of applications. Besides, different applications have unique patterns that may be more suitable for extraction with some deep learning model.

Among the three models, LSTM performs the best for the recall rate with an average value of 95.0%. However, it achieves a low precision, i.e., 88.3% on average. Particularly, there is only an 83.8% precision for `Anonymous Communication`. CNN performs the best both for the accuracy and the precision, which achieves a 95.4% accuracy and a 96.6% precision on average. 7 of the 10 applications have the highest accuracy and 8 of the 10 applications have the highest precision with CNN compared to the other two models. Moreover, CNN achieves an acceptable recall rate of 93.3% on average. SDAE achieves a 93.3% accuracy, a 93.0% recall rate, and a 92.3% precision on average. It performs the worst for the accuracy and the recall rate.

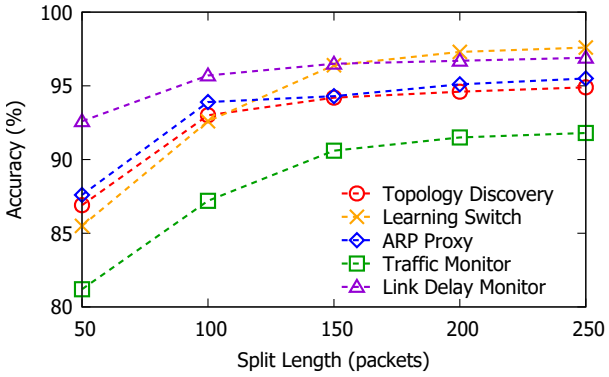
We evaluate the stability of the three models on fingerprinting different applications with the standard deviation. SDAE has the highest standard deviations of accuracy, recall rate, and precision. LSTM has the lowest standard deviation of recall rate and the moderate standard deviations of accuracy and precision. CNN outperforms the other two models both in the standard deviations of accuracy and precision and has a moderate standard deviation of recall rate, which is the most stable deep learning model.

Overall, by comparing the three models with each other, we observe that CNN is the most effective and stable for an adversary to fingerprint different SDN applications, especially in classification accuracy and precision.

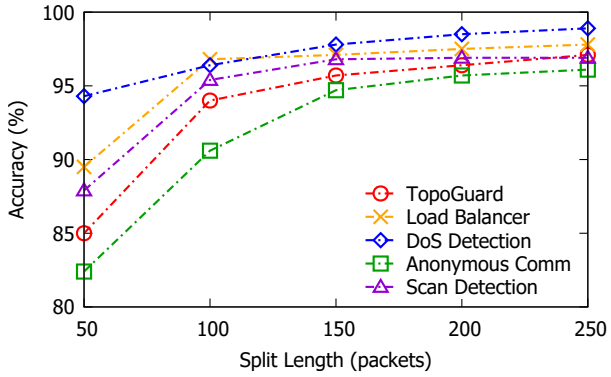
Effectiveness with Different Split Lengths. The effectiveness of fingerprinting SDN applications may change with different lengths of samples. Thus, we divide sequences of control packets into different lengths to train and test deep neural networks. Because CNN performs best, we here explore its accuracy, recall rate, and precision of fingerprinting applications with different lengths of samples.

Figure 6 shows the accuracy for fingerprinting different applications with various split lengths. The results show the accuracy of fingerprinting an application goes up with the split length. When the split length is 50, fingerprinting most applications achieves a low accuracy that is less than 90%. Particularly, fingerprinting `Traffic Monitor` only reaches an accuracy of 81.2%. When the split length is increased to 250, the accuracy reaches more than 95% for 9 of the 10 applications. The accuracy of fingerprinting `Learning Switch, Traffic Monitor, Topoguard, and Anonymous Communication` increases by more than 10%. The reason is that more packets in a sample give more underlying patterns. Although the accuracy goes up with the split length, the growth rate of the accuracy gradually slows down. When we increase the split length from 200 to 250, the accuracy is increased less than 1% for most applications and tends to converge.

Figure 7 shows the recall rate for fingerprinting different applications with various split lengths. Similar to the accuracy, the recall rate goes up with split lengths. When the split length is increased from 50 to 150, the recall rate for fin-

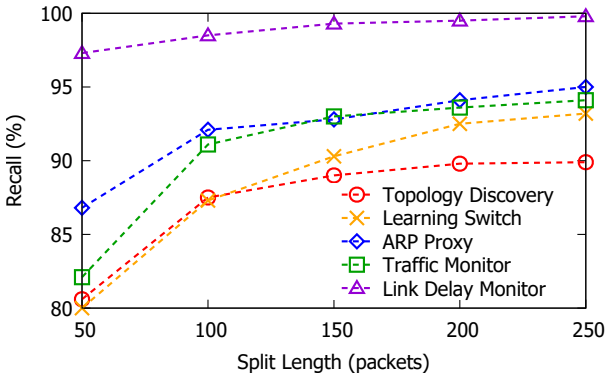


(a) Accuracy of First Five Apps.

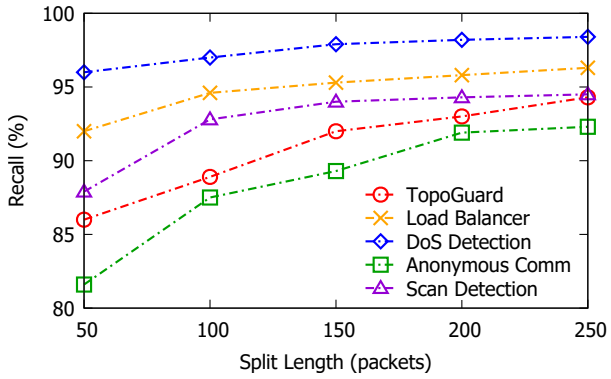


(b) Accuracy of Last Five Apps.

Figure 6: Accuracy of Fingerprinting SDN Applications with Different Split Lengths.



(a) Recall Rate of First Five Apps.



(b) Recall Rate of Last Five Apps.

Figure 7: Recall Rate of Fingerprinting SDN Applications with Different Split Lengths.

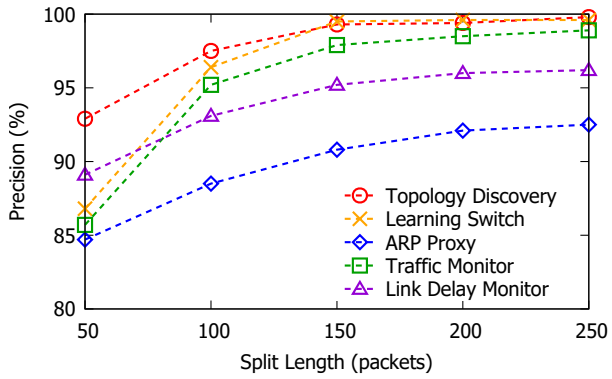
gerprinting most applications increases significantly. For instance, the recall rate for fingerprinting `Learning Switch` increases by 13.2%. There are two exceptions of SDN applications, i.e., `Link Delay Monitor` and `DoS Detection`. The recall rate of fingerprinting the two applications is already more than 90% even with a small part of control traffic, i.e., 50 packets, and improves slightly with more control packets in a sample. When the split length is greater than 150, the recall rate stops significant improvement.

Figure 8 shows the precision for fingerprinting different applications with various split lengths. The precision gradually increases with the split length, following a similar trend like the accuracy and the recall rate. When the split length is increased from 50 to 250, the precision for fingerprinting `Learning Switch` improves the most, i.e., a 12.8% increase, and the precision for fingerprinting `DoS Detection` improves the least, i.e., a 5.3% increase. Moreover, the precision for fingerprinting most applications does not significantly improve when the split length exceeds 150.

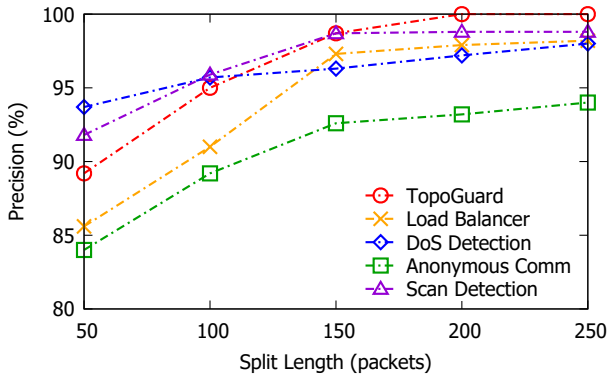
According to the above results, we conclude that an adversary can well fingerprint SDN applications with more than 150 encrypted control packets.

Effectiveness with Different Number of Applications. We explore how the effectiveness of fingerprinting an application changes with different number of applications running on controllers. We train and test CNN models for fingerprinting `ARP Proxy` with five datasets [2]. The five datasets contain control traffic of at most 6, 7, 8, 9, and 10 SDN applications, respectively. As shown in Figure 9, the accuracy, recall rate, and precision slightly decrease with the number of applications. When the number of applications increases from six to ten, the accuracy drops by 1.9%, the recall rate drops by 1.8%, and the precision drops by 2.0%. The results demonstrate that the effectiveness of fingerprinting applications is not significantly affected by the number of applications. Our main conclusion here is that deep learning based classifiers are capable of extracting stable patterns of control

[2] We also test the effectiveness of fingerprinting other applications with different number of applications. The results are similar to those in Figure 9. For simplicity and due to space constraints, we do not present the results.



(a) Precision of First Five Apps.



(b) Precision of Last Five Apps.

Figure 8: Precision of Fingerprinting SDN Applications with Different Split Lengths.

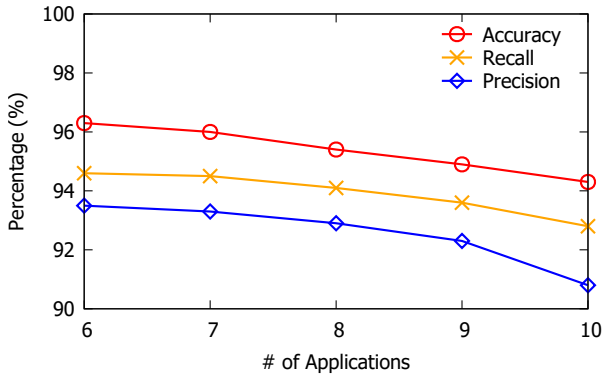


Figure 9: Effectiveness of Fingerprinting an SDN Application with Different Number of Applications.

traffic, which allows an adversary to fingerprint SDN applications with a high success rate.

5.3 Performance

We evaluate the runtime for building a classifier to fingerprint an SDN application. As the runtime of classifiers with different applications changes slightly (less than 1% differences), we list the average runtime for different applications in Table 4. CNN runs fastest among the three models with 2.4 min runtime since it has fewer learnable parameters and most computations in CNN happen in parallel. LSTM performs the slowest due to its recurrent structure where the subsequent processing steps depend on the previous ones.

Table 4: Runtime of Different DNN Models

DNN Models	SDAE	LSTM	CNN
Average Runtime	6.9 min	350.1 min	2.4 min
Loss	0.179	0.192	0.125

6 Discussion

In this section, we discuss the limitations of our current work and possible countermeasures to mitigate the attack.

6.1 Limitations

Model Effectiveness on Traffic and Settings Changes. Our experiments contain more than 10,000 real network flows to train the deep learning models. Results show that enough training data makes the models generalize well for different flows containing different numbers and sizes of packets. However, since deep learning models learn patterns from data, they cannot classify unseen patterns that training data does not contain. Thus, if network traffic or setting of applications changes significantly, the classification accuracy for fingerprinting certain applications may decrease unless we provide more diverse data to train models. According to our analysis, among the tested ten applications, the accuracy of fingerprinting four applications, i.e., `Topology Discovery`, `Traffic Monitor`, `Link Delay Monitor`, and `Anonymous Communication`, is sensitive to settings changes but not to traffic changes since they hardly generate network events based on network traffic. For fingerprinting the other six applications, the accuracy may decrease when either network traffic or settings significantly change if there lacks enough training data to cover the changed patterns.

Classifying SDN Applications with Control Traffic of Multiple Switches. Our threat model currently assumes that an adversary eavesdrops control traffic between one SDN controller and one switch, which is common in practice. Although we demonstrate many applications can be identified with control traffic by deep learning models, we admit a few applications cannot be well classified without further information from control traffic between the controller and other switches. It is because a few applications perform similar

behaviors on one local switch but have different behaviors on multiple switches. Therefore, if we assume a stronger threat model, i.e., an adversary can eavesdrop control traffic of multiple switches, more applications may be classified.

We elaborate this with an example. Considering the two SDN applications: `Learning Switch` and `Reactive Routing`. Any of them running on controllers receives a `PACKET_IN` message to analyze a new flow and then installs a flow rule into the ingress switch with a `FLOW_MOD` message. The `PACKET_IN` messages are same between the two applications for same flows and the `FLOW_MOD` messages between them can also be same if the two applications set same match fields and actions in the flow rules. Thus, the patterns of control traffic for the two applications are same in all aspects, such as packet lengths, relative orders of packets, directions of packets, etc. An adversary cannot classify which application running on the controller only with control traffic between the controller and the ingress switch. However, `Reactive Routing` paves a routing path for a flow in many switches, i.e., installing multiple `FLOW_MOD` messages into each switch along the path once receiving a `PACKET_IN` message from the ingress switch. Instead, `Learning Switch` performs per-hop forwarding, i.e., receiving a `PACKET_IN` message from each switch and installing a `FLOW_MOD` message into each corresponding switch. The patterns of the two applications are different from the view of multiple switches. Thus, it is possible for an adversary to classify the two applications by analyzing control traffic from many switches. However, how to effectively leverage the context between control traffic of multiple switches to fingerprint SDN applications is challengeable. We leave it as future work.

Fingerprinting SDN Applications that generate little control traffic. Although most applications running on controllers continuously generate much control traffic, a few applications only generate little control traffic at some time when network administrators actively change the policy of the applications. For example, `REST Firewall` [9] in the Floodlight controller generates no control messages most of the time. However, if a network administrator updates network security policies by commanding the application with `REST API`, the application will install flow rules with specified match fields and actions into switches to enable new security policies. The control traffic of the application is little most of the time, which is difficult for deep learning models to train effective classifiers to identify the application. Moreover, the patterns of control traffic highly depend on how a network administrator command the application, which is extremely mutable. Deep learning models may not extract universal patterns to generalize well to identify the application. One possible but ineffective solution is to manually summarize useful patterns with enough SDN background knowledge to identify them. Our future work will focus on how to efficiently solve the problem.

6.2 Possible Countermeasures

Reducing Differences for Control Traffic. To the best of our knowledge, there are no public SDN defense systems that can mitigate our attack. However, as an adversary fingerprints SDN applications mainly based on different patterns of control traffic, one straightforward mitigation is to reduce the difference between control traffic of various applications. As we mentioned in Section 3.2, the main difference exists in the packet lengths, the packet directions, the relative orders of packets, and the number of packets. Thus, we may encapsulate control messages to normalize them. To normalize the packet lengths, both controllers and switches can reshape different packets by splitting one big packet into several packets or adding padding in small packets so that the packet lengths are equal. Since different packets cannot be identified without knowing their real lengths, the relative orders of packets are also hided. Moreover, to eliminate the differences in the packet directions and the number of packets, controllers and switches can morph packets into fixed bursts, i.e., breaking each traffic pattern into small bursts of packets consisting of a fixed number of consecutive outgoing packets followed by a fixed number of consecutive incoming packets. By normalizing control packets, deep learning models may thus identify SDN applications with a low accuracy. However, one main disadvantage is that it requires many modifications in switches, controllers, and the OpenFlow protocol. Applying the countermeasure in real SDN environments may take a long time and bring some costs.

Adding Adversarial Examples. Another interesting defense strategy worthy of being further studied is to mislead deep neural networks by deliberately generating adversarial examples. They are specially crafted instances with small and intentional feature perturbations to fool deep learning models into false classifications or predictions. Previous studies [40, 65] have demonstrated that adversarial examples can successfully fool deep learning for computer vision and pattern recognition. We may explore how an SDN application can generate adversarial examples of control packets to mislead fingerprinting SDN applications. For example, `ARP Proxy` may periodically generate control packets that simulate the patterns of another application, such as `Learning Switch`, to mislead the classification of deep learning models. It may effectively decrease the accuracy of fingerprinting SDN applications. This defense requires to modify the SDN applications.

7 Related Work

Fingerprinting and Probing in SDN. There are many previous studies on fingerprinting and probing information in SDN. Shin et al. [52] designed a scanning tool to remotely fingerprint networks that deploy SDN by measuring response delays of probing packets. Klöti et al. [32] provided a prob-

ing technique to fingerprint aggregated flow rules by timing TCP setup. Cui et al. [21] demonstrated that an adversary can acquire knowledge on which flow rules installed on switches by analyzing the packet-pair dispersion of data packets. Achleitner et al. [12] presented SDNMap to reconstruct the detailed composition of flow rules by actively sending probing packets with different network protocols. Liu et al. [39] developed a Markov model to infer if a target flow occurred recently by sending optimized probing packets, in the face of rule expiration and eviction. John et al. [57] presented a sophisticated attack to infer host communication patterns, network access control and network monitoring policies by timing processing delays of controllers. Azzouni et al. [14] fingerprinted SDN controllers by timing timeouts of flow rules as well as processing time of controllers. Although the above studies effectively probe many types of information in SDN, none of them show how to fingerprint SDN applications with control traffic. Our work reveals a new attack vector in SDN.

Security Research in SDN. Recently, many SDN security issues have been studied. They cover attacks and security enhancements in all layers of SDN. In the application layer, studies focus on cross-app poisoning [58], malicious applications abusing [36], and secure system for permission control of SDN applications [46, 63]. A wide range of studies focus on the control layer security. Various attacks are presented, including flooding controllers [53, 60], disrupting control channels [18], attacking information mismanagement in SDN-datastores [23], poisoning network typologies [27] and identifiers of network stack [30], generating harmful race conditions in controllers [66], and subverting SDN controllers [48]. Extensive security enhancement systems [22, 30, 51, 56, 60] are designed to mitigate the attacks. Other studies present attacks on data plane, such as low-rate flow table overflow [19], attacking SDN switches with control plane reflection [69], and security policies violation [45]. To fortify SDN data plane, intrusion detection and abnormal data plane diagnose systems [38, 50] are provided. Moreover, automatic vulnerability discovery and security assessment tools [29, 35] are designed to understand the possible attack surface of SDN. In contrast to existing work, we show a new threat to SDN and existing defense systems cannot defend it.

Encrypted Network Traffic Analysis. Analyzing encrypted network traffic to infer possible information based on packet sizes, timing and other side channel leaks has been extensively studied. Li et al. [37] fingerprinted personas from WI-FI traffic by analyzing meta-data information on interactions through HTTPS connections with machine learning. Chen et al. [20] showed detailed sensitive information can be leaked out from encrypted network traffic of web applications. Wright et al. [64] designed an attack to identify the phrases spoken within a call by analyzing lengths of encrypted VoIP packets. Zhang et al. [70] provided HoMonit to monitor smart home applications from encrypted wireless traffic. Moreover, a series of previous studies [25, 43, 44, 47, 54, 55, 62] focuses on website fingerprinting with the onion router (Tor) that preserves anonymity for Internet users. They reveal which website Tor users are visiting by analyzing Tor traffic with machine learning. However, all these studies make a single page assumption, i.e., the collected traffic always belongs to a single page from a website and contains no mixed traffic from other pages. One study [67] relaxes the assumption and provides a multi-tab website fingerprinting attack on partially mixed traffic. It provides a split algorithm to extract a small initial chunk of packets of the first page, which is not overlapped with the packets of the following pages. Different from these studies, we concentrate on fingerprinting what applications run on SDN controllers via encrypted control traffic. Particularly, control packets of different SDN applications are mixed in a single TCP connection. Since all applications concurrently send control messages to switches, control packets are tightly coupled and totally mixed and thus can not be split using the algorithm in the study [67]. We provide novel techniques to accurately and efficiently fingerprint all applications from mixed control traffic.

8 Conclusion

In this paper, we present a new attack on SDN that fingerprints SDN applications with low-level and encrypted control traffic. It exploits different patterns of control traffic caused by different behaviors of applications to infer what applications run on SDN controllers. In order to characterize the underlying patterns, we transform network packets into the time series and apply deep learning to automatically learn the patterns to fingerprint SDN applications. We divide the task of fingerprinting multiple SDN applications into several subtasks to improve the efficiency of training deep learning models. We collect massive traces of control traffic from a real SDN testbed. Extensive experiments demonstrate that an adversary can effectively fingerprint SDN applications with a high accuracy.

Acknowledgments

The research is partly supported by the National Natural Science Foundation of China under Grant 61625203, 61832013, 61572278, and U1736209, ONR grants N00014-16-1-3214, N00014-18-2893, and ARO grant W911NF-17-1-0447. Mingwei Xu and Qi Li are corresponding authors.

References

[1] ARP Proxy. `https://github.com/mbredel/floodlight-proxyarp/`. [Online].

[2] CAIDA Passive Monitor: Chicago B. http://www.caida.org/data/passive/trace_stats/chicago-B/2015/?monitor=20150219-130000.UTC. [Online].

[3] Learning Switch. https://github.com/floodlight/floodlight/tree/master/src/main/java/net/floodlightcontroller/learningswitch/. [Online].

[4] Load Balancer. https://github.com/floodlight/floodlight/blob/master/src/main/java/net/floodlightcontroller/loadbalancer/. [Online].

[5] Microsoft Azure and Software Defined Networking. https://docs.microsoft.com/en-us/windows-server/networking/sdn/azure_and_sdn/. [Online].

[6] Network Traffic Monitor. https://github.com/floodlight/floodlight/blob/master/src/main/java/net/floodlightcontroller/statistics/. [Online].

[7] NGMN - 5G White Paper. https://ngmn.org/5g-white-paper/5g-white-paper.html. [Online].

[8] OpenFlow Specification v1.5.1. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf. [Online].

[9] REST Firewall. https://github.com/floodlight/floodlight/tree/master/src/main/java/net/floodlightcontroller/firewall/. [Online].

[10] Software Load Balancing (SLB) for SDN. https://docs.microsoft.com/en-us/windows-server/networking/sdn/technologies/network-function-virtualization/software-load-balancing-for-sdn. [Online].

[11] Topology Discovery. https://github.com/floodlight/floodlight/tree/master/src/main/java/net/floodlightcontroller/topology/. [Online].

[12] Stefan Achleitner, Thomas La Porta, Trent Jaeger, and Patrick McDaniel. Adversarial network forensics in software defined networking. In *ACM SOSR'17 (2017)*.

[13] Ahmad Aseeri, Nuttapong Netjinda, and Rattikorn Hewett. Alleviating eavesdropping attacks in software-defined networking data plane. In *ACM CISRC'17 (2017)*.

[14] Abdelhadi Azzouni, Othmen Braham, Thi Mai Trang Nguyen, Guy Pujolle, and Raouf Boutaba. Fingerprinting openflow controllers: The first step to attack an sdn control plane. In *IEEE GLOBECOM'16 (2016)*.

[15] Kevin Benton, L Jean Camp, and Chris Small. Openflow vulnerability assessment. In *ACM HotSDN'13 (2013)*.

[16] Kevin Benton, L Jean Camp, and Chris Small. Openflow vulnerability assessment (extended abstract). https://benton.pub/research/openflow_vulnerability_assessment.pdf.

[17] Michael Brooks and Baijian Yang. A man-in-the-middle attack against opendaylight sdn controller. In *ACM SIGITE/RIIT'15 (2015)*.

[18] Jiahao Cao, Qi Li, Xie Renjie, Kun Sun, Guofei Gu, Mingwei Xu, and Yuan Yang. The crosspath attack: Disrupting the sdn control channel via shared links. In *USENIX Security'19 (2019)*.

[19] Jiahao Cao, Mingwei Xu, Qi Li, Kun Sun, Yuan Yang, and Jing Zheng. Disrupting sdn via the data plane: a low-rate flow table overflow attack. In *SecureComm'17 (2017)*.

[20] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *IEEE S&P'10 (2010)*.

[21] Heng Cui, Ghassan O Karame, Felix Klaedtke, and Roberto Bifulco. On the fingerprinting of software-defined networks. *IEEE Transactions on Information Forensics and Security*, 11(10):2160–2173, 2016.

[22] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *NDSS'15 (2015)*.

[23] Vaibhav Hemant Dixit, Adam Doupé, Yan Shoshitaishvili, Ziming Zhao, and Gail-Joon Ahn. Aim-sdn: Attacking information mismanagement in sdn-datastores. In *ACM CCS'18 (2018)*.

[24] Seyed K Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and elastic ddos defense. In *USENIX Security'15 (2015)*.

[25] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security'16 (2016)*.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *NDSS'15 (2015)*.

[28] Jain, Sushant et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[29] Samuel Jero, Xiangyu Bu, Cristina Nita-Rotaru, Hamed Okhravi, Richard Skowyra, and Sonia Fahmy. Beads: automated attack discovery in openflow-based sdn systems. In *RAID'17 (2017)*.

[30] Samuel Jero, William Koch, Richard Skowyra, Hamed Okhravi, Cristina Nita-Rotaru, and David Bigelow. Identifier binding attacks and defenses in software-defined networks. In *USENIX Security'17 (2017)*.

[31] Seong-Mun Kim, Gyeongsik Yang, Chuck Yoo, and Sung-Gi Min. Bfd-based link latency measurement in software defined networking. In *IEEE CNSM'17 (2017)*.

[32] Rowan Klöti, Vasileios Kotronis, and Paul Smith. Openflow: A security analysis. In *IEEE ICNP'13 (2013)*.

[33] Diego Kreutz, Fernando Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *ACM HotSDN'13 (2013)*.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] Seungsoo Lee, Changhoon Yoon, Chanhee Lee, Seungwon Shin, Vinod Yegneswaran, and Phillip A Porras. Delta: A security assessment framework for software-defined networks. In *NDSS'17 (2017)*.

[36] Seungsoo Lee, Changhoon Yoon, and Seungwon Shin. The smaller, the shrewder: A simple malicious application can kill an entire sdn environment. In *ACM SDN-NFV Security 2016*.

[37] Huaxin Li, Zheyu Xu, Haojin Zhu, Di Ma, Shuai Li, and Kai Xing. Demographics inference through wi-fi network traffic analysis. In *IEEE INFOCOM'16 (2016)*.

[38] Qi Li, Yanyu Chen, Patrick PC Lee, Mingwei Xu, and Kui Ren. Security policy violations in sdn data plane. *IEEE/ACM Transactions on Networking (TON)*, 26(4):1715–1727, 2018.

[39] Liu, Sheng et al. Flow reconnaissance via timing attacks on sdn switches. In *IEEE ICDCS'17 (2017)*.

[40] Jiajun Lu, Theerasit Issaranon, and David Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. In *IEEE ICCV'17 (2017)*.

[41] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In *RAID'11 (2011)*.

[42] Roland Meier, David Gugelmann, and Laurent Vanbever. itap: In-network traffic analysis prevention using software-defined networks. In *ACM SOSR'17 (2017)*.

[43] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *ACM CCS'18 (2018)*.

[44] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS'16 (2016)*.

[45] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for openflow networks. In *ACM HotSDN'12 (2012)*.

[46] Phillip A Porras, Steven Cheung, Martin W Fong, Keith Skinner, and Vinod Yegneswaran. Securing the software defined network control layer. In *NDSS'15 (2015)*.

[47] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *NDSS'18 (2018)*.

[48] Christian Röpke and Thorsten Holz. Sdn rootkits: Subverting network operating systems of software-defined networks. In *RAID'15 (2015)*.

[49] Arash Shaghaghi, Mohamed Ali Kaafar, Rajkumar Buyya, and Sanjay Jha. Software-defined network (sdn) data plane security: Issues, solutions and future directions. *arXiv preprint arXiv:1804.00262*, 2018.

[50] Arash Shaghaghi, Mohamed Ali Kaafar, and Sanjay Jha. Wedgetail: An intrusion prevention system for the data plane of software defined networks. In *ACM AsiaCCS'17 (2017)*.

[51] Gao Shang, Peng Zhe, Xiao Bin, Hu Aiqun, and Ren Kui. Flooddefender: Protecting data and control plane resources under sdn-aimed dos attacks. In *IEEE INFOCOM'17 (2017)*.

[52] Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In *ACM HotSDN'13 (2013)*.

[53] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *ACM CCS'13 (2013)*.

[54] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *ACM CCS'18 (2018)*.

[55] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *ACM CCS'18 (2018)*.

[56] Richard Skowyra, Lei Xu, Guofei Gu, Veer Dedhia, Thomas Hobson, Hamed Okhravi, and James Landry. Effective topology tampering attacks and defenses in software-defined networks. In *IEEE DSN'18 (2018)*.

[57] John Sonchack, Anurag Dubey, Adam J Aviv, Jonathan M Smith, and Eric Keller. Timing-based reconnaissance and defense in software-defined networks. In *ACSAC'16 (2016)*.

[58] Benjamin E Ujcich, Samuel Jero, Anne Edmundson, Qi Wang, Richard Skowyra, James Landry, Adam Bates, William H Sanders, Cristina Nita-Rotaru, and Hamed Okhravi. Cross-app poisoning in software-defined networking. In *ACM CCS'18 (2018)*.

[59] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12):3371–3408, 2010.

[60] Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *IEEE DSN'15 (2015)*.

[61] Haopei Wang, Guangliang Yang, Phakpoom Chinprutthiwong, Lei Xu, Yangyong Zhang, and Guofei Gu. Towards fine-grained network security forensics and diagnosis in the sdn era. In *ACM CCS'18 (2018)*.

[62] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security'14 (2014)*.

[63] Xitao Wen, Bo Yang, Yan Chen, Chengchen Hu, Yi Wang, Bin Liu, and Xiaolin Chen. Sdnshield: Reconciliating configurable application permissions for sdn app markets. In *IEEE DSN'16 (2016)*.

[64] Charles V Wright, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *IEEE S&P'08 (2008)*.

[65] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *IEEE ICCV'17 (2017)*.

[66] Lei Xu, Jeff Huang, Sungmin Hong, Jialong Zhang, and Guofei Gu. Attacking the brain: Races in the sdn control plane. In *USENIX Security'17 (2017)*.

[67] Yixiao Xu, Tao Wang, Qi Li, Qingyuan Gong, Yang Chen, and Yong Jiang. A multi-tab website fingerprinting attack. In *ACM ACSAC'18 (2018)*.

[68] Changhoon Yoon, Seungsoo Lee, Heedo Kang, Taejune Park, Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Flow wars: Systemizing the attack surface and defenses in software-defined networks. *IEEE/ACM Transactions on Networking (TON)*, 25(6):3514–3530, 2017.

[69] Menghao Zhang, Guanyu Li, Lei Xu, Jun Bi, Guofei Gu, and Jiasong Bai. Control plane reflection attacks in sdns: new attacks and countermeasures. In *RAID'18 (2018)*.

[70] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. Homonit: Monitoring smart home apps from encrypted traffic. In *ACM CCS'18 (2018)*.

[71] Jing Zheng, Qi Li, Guofei Gu, Jiahao Cao, David KY Yau, and Jianping Wu. Realtime ddos defense using cots sdn switches via adaptive correlation analysis. *IEEE Transactions on Information Forensics and Security*, 13(7):1838–1853, 2018.

A Hyperparameters

Table 5: Hyperparameters of Different DNN Models

Hyperparameter	DNN Models		
	SDAE	CNN	LSTM
optimizer	RMSProp	Adam	RMSProp
learning rate	0.001	0.001	0.001
decay	0.0	0.0	0.0
batch size	64	64	64
training epoch	5 .. 10	5 .. 10	15 .. 25
number of layers	4	10	4
input units	50 .. 250	50 .. 250	50 .. 250
hidden layer units	100, 50, 20	-	32, 32, 32
dropout	0.1	0.5	-
activation	tanh	tanh	-
pretraining optimizer	RMSProp	-	-
pretraining learning rate	0.001	-	-
kernels	-	100, 200	-
kernel size	-	10	-
pool size	-	2	-