ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

EFFICIENT MEASURE STRENGTH SDN NETWORKS BY FINGER PRINTING

P.SATHYA[1] ,P.MOHANAVALLI 2
1.PG Student, Department of Computer Science And Engineering, Idhaya Engineering College For Women,Chinnasalem
2.Assistant Professor, Department of Computer Science And Engineering, Idhaya Engineering College For Women,Chinnasalem
[1]sathyaponnan@gmail.com,[2]mohanap05@gmail.com

***Abstract*— Software-defined networking (SDN) eases network management by centralizing the control plane and separating it from the data plane. The separation of planes in SDN, however, introduces new vulnerabilities in SDN networks, since the difference in processing packets at each plane allows an adversary to fingerprint the network's packet-forwarding logic. In this paper, we study the feasibility of fingerprinting the controller-switch interactions by a remote adversary, whose aim is to acquire knowledge about specific flow rules that are installed at the switches. This knowledge empowers the adversary with a better understanding of the network's packet-forwarding logic and exposes the network to a number of threats. In this paper, we collect measurements from hosts located across the globe using a realistic SDN network comprising of OpenFlow hardware and software switches. We show that, by leveraging information from the RTT and packet-pair dispersion of the exchanged packets, fingerprinting attacks on SDN networks succeed with overwhelm-ing probability. We additionally show that these attacks are not restricted to active adversaries, but can also be mounted by passive adversaries that only monitor traffic exchanged with the SDN network. Finally, we discuss the implications of these attacks on the security of SDN networks, and we present and evaluate an efficient countermeasure to strengthen SDN networks against fingerprinting. Our results demonstrate the effectiveness of our countermeasure in deterring fingerprinting attacks on SDN networks.**

***Index Terms*— Software-defined networking, OpenFlow, fingerprinting, security.**

I. INTRODUCTION

SOFTWARE-DEFINED networking (SDN) [14], [27] eases the development and deployment of network applications by defining a standard interface between the control plane and the data plane. In SDN, the control plane is imple-mented by a logically centralized controller, which interacts over a bi-directional communication channel with the data plane's network devices. The controller can query devices for their state, e.g., to acquire traffic statistics or information about the status of the switches' ports, and modify their forwarding behavior, by installing and deleting flow rules. Network devices can also notify the controller about network events (e.g., the reception of certain packets) and device's state changes. For example, a number of advanced reactive control plane logic implementations [11], [17], [35], [46] configure network devices to send notification to the controller according to some installed policy (e.g., when a received packet does not match any of the installed flow rules). This notification triggers the controller to perform a series of operations, such as installing the appropriate forwarding rules at the switches, reserve network resources on a given network's path, etc.

The separation of the control and data plane in SDN opens the doors for a remote adversary to fingerprint the network. In particular, whenever packet forwarding is performed in hardware, then packets at the data plane are processed several orders of magnitude faster than at the software-based control plane. This discrepancy acts as a distinguisher for a remote adversary to learn whether a given probe packet is handled just at the data plane or triggers an interaction between the data plane and the control plane. An interaction provides evidence that the probe packet does not have any matching flow rule stored at the switch's flow table (or it requires special attention from the controller). This knowledge empowers an adversary with a better understanding of the network's packet-forwarding logic and, as we outline in this work, exposes the network to a number of threats. In spite of the plethora of SDN security solutions in the literature [5], [15], [33], [39]–[41], no contri-bution analyzes the feasibility and realization of fingerprint-ing attacks on practical SDN deployments. Moreover, there are no proposed solutions to alleviate fingerprinting attacks on SDN.

In this paper, we address this problem and study the finger-printing of controller-switch interactions by a remote adversary with respect to various network parameters, such as the number of hops in the communication path, and the data link bandwidth. For that purpose, we collect measurements from 20 different hosts located across the globe (Australia, Asia, Europe, and North America) using an SDN network compris-ing of several OpenFlow hardware and software switches. Our results show that, by leveraging information from the packet-pair dispersion of the exchanged packets, fingerprinting attacks on SDN networks succeed with overwhelming probability. For instance, an adversary can correctly identify, with an accuracy of almost 99%, whether a probe packet triggers the installation of forwarding rules at three hardware switches in our SDN network.

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

We also show that fingerprinting attacks can be mounted by passive adversaries that, e.g., capture a snapshot of the traffic exchanged with the SDN network. Although existing traffic might not contain packet pair traces, our findings show that a passive adversary can leverage the RTT of packets (that are exchanged within a short time interval) to fingerprint the SDN network with an accuracy up to almost 99%. The fingerprinting accuracy due to the RTT of packets is, however, largely affected by the SDN network size, and significantly deteriorates with time. [3] discussed about Reconstruction of Objects with VSN. By this object reconstruction with feature distribution scheme, efficient processing has to be done on the images received from nodes to reconstruct the image and respond to user query.

This work extends our prior work in [2]. For instance, an adversary that knows which packets cause an interaction with the controller can, e.g., acquire evidence about the occurrence of a particular communication event. This enables the adversary to understand the logic adopted by the controller in managing the SDN network, or to launch Denial-of-Service (DoS) attacks by overloading the switches with bogus flow-table updates [22]. In light of our findings, we present and evaluate an efficient countermeasure to strengthen SDN networks against fingerprinting. Our evaluation shows that our countermeasure considerably reduces the ability of an adversary to mount fingerprinting attacks on SDN networks.

The remainder of this paper is organized as follows. In Section II, we define our problem statement. In Section III, we describe our setup, the performed experiments, and summarize the collected data. In Section IV, we present and detail our results. In Section V, we analyze the implications of our findings on the security of SDN networks. In Section VI, we present and evaluate an efficient countermeasure to deter fingerprinting in SDN networks. In Section VII, we discuss related work, and we conclude the paper in Section VIII.

II. Problem Statement

Before describing the focus of our work, we give a brief refresher on OpenFlow, a widely deployed realization of SDN.

A. Background

SDN separates the control and data planes by defining a switch's programming interface and a protocol to access such interface, i.e., the OpenFlow protocol [31]. The controller leverages the OpenFlow protocol to access the switch's programming interface and configure the forwarding behavior of the switch's data plane. The communication between the controller and switches is established using an out-of-band control channel.

The core entities exposed by the OpenFlow switch's programming interface are flow tables and flow rules. A flow table of a switch is just a container for its flow rules, which define the switch's forwarding behavior. The controller can add, delete, or modify flow rules of a switch's flow table by sending an `OFPT_FLOW_MOD` OpenFlow message to the switch. The parameters of an `OFPT_FLOW_MOD` message specify how the flow table of the switch should be modified. A flow rule, for instance, provides a semantic like "if a network packet's IP destination address is 1.2.3.4, then forward the packet to port 2." In general, a flow rule contains a *match set* that defines the network packets to which the rule applies. It further contains an *action set* that defines the actions that should be applied to such packets, for example, forward to port 2. Whenever a packet is received by a switch, the packet's header is used as a search key to retrieve the rule that applies to the packet, by performing a lookup in the flow table. The lookup operation compares the packet's header with the rules' match set to find the rule that *matches* the packet. Rules are prioritized in case multiple rules match. For the cases in which the controller needs to inspect a network packet, before performing a forwarding decision and installing the corresponding forwarding rules, OpenFlow defines a special "forward to controller" action. When this action is applied to a packet, the switch generates an `OFPT_PACKET_IN` message that is sent to the controller. This message contains the original packet and some additional information, such as the switch and the port ID onto which the packet was received.

The `OFPT_PACKET_IN` feature is used in basic network control logic implementations, such as the one of an Ethernet learning switch. It is also used in more complex dynamic con-trol plane implementations. In both cases, the network operates as follows: a packet received by the switch generates an `OFPT_PACKET_IN` message; the controller receives and analyzes the message to take a forwarding decision; the decision is finally implemented by sending `OFPT_FLOW_MOD` messages, which install rules at the relevant switches. This ensures that all similar packets, i.e., those that belong to the same network flow, are forwarded directly by the switches with no further interactions with the controller. Note that the controller can use barrier messages to ensure that message dependencies are met, e.g., when multiple switches are reconfigured by the controller for handling a new network flow. When a switch receives a barrier request (`OFPT_BARRIER_REQUEST`), the switch must finish all previously received messages before processing new messages. After processing all messages, it notifies the controller by sending a barrier reply message (`OFPT_BARRIER_REPLY`).

B. Problem Statement

The main objective of our work is to study the ability of a remote adversary to identify whether an interaction between the controller and the switches (and a subsequent rule installation) has been triggered by a given packet.

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

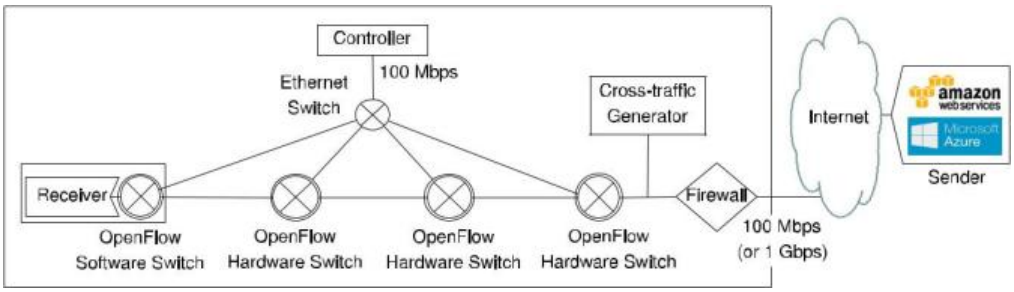

Fig. 1. Sketch of our measurement setup. Our testbed comprises three NEC PF5240 OpenFlow hardware switches and one OpenVSwitch (version 2.3.1).

then this suggests that the received packet requires further examination by the controller, e.g., since it does not have any matching entry stored at the switch's flow table, or because the controller requires additional information before installing a forwarding decision at the switches.

In our study, we consider both active and passive adversaries. We assume that an active adversary can compromise a remote client, inject probe packets of her choice, and capture the timing of the corresponding responses issued by a server. In contrast, a passive adversary cannot inject packets in the network but only monitors the exchanged traffic between the server and the client. Notice that passive adversaries are hard to detect by standard intrusion detection systems since they do not generate any extra network traffic.

Our study focuses on answering the following questions:

- Is it possible to remotely identify whether the installation of flow rules has been triggered by a given packet?
- What is the accuracy of fingerprinting attacks in SDN networks?
- What is the impact of the number of switches that need to be configured on the fingerprinting accuracy?
- What is the impact of the data link bandwidth on the fingerprinting accuracy?
- Is the fingerprinting accuracy affected by the presence of software switches in the SDN network?
- How and to which extent can such fingerprinting attacks be efficiently mitigated?

III. Experimental Setup

In this section, we detail our experimental setup. This includes a description of our testbed, the used features, the conducted experiments, and the collected datasets.

A. Testbed

Our measurement setup is summarized in Figure 1. The testbed comprises three OpenFlow hardware switches (three NEC PF5240 switches [29]) and one OpenFlow software switch (an OpenVSwitch, version 2.3.1 [32]). The switches are connected to the data plane over a 100 Mbps data channel. We also consider the case where the data channel bandwidth increases to 1 Gbps. Note that, although our testbed only comprises three hardware switches, it can emulate the processing of packets in many realistic datacenters. Recall that a conven-tional datacenter's network typically consists of three layers of switches: top-of-rack, aggregation, and core [1]. Packets are usually processed by at most one switch in each of these layers, that is, each packet traverses (at most) three hops in the datacenter's network. The testbed's switches interface with a Floodlight v0.9 controller [12], which runs on a computer with a 6-core Intel Xeon L5640 2.26 GHz CPU and 24 GB of RAM. A legacy Ethernet switch bridges the connections between the OpenFlow switches and the controller. To emulate realistic network load on the control channel, we limit the control interface of the switches to 100 Mbps.

The controller is configured to minimize the processing delay for an incoming *packet-in* event, i.e., we only require the controller to perform a table lookup and retrieve pre-computed forwarding rules in response to *packet-in* events. Furthermore, the controller always performs bi-directional flow installation; that is, the handling of a *packet-in* event triggers the installa-tion of a pair of rules, one per flow direction, at each involved switch. We ensure that the controller's CPU is not overloaded during our measurements.

We deploy a cross-traffic generator on an AMD dual core processor running at 2.5 GHz to emulate realistic WAN traffic load on the switches' ports that were used in our study. The generated cross traffic follows a Pareto distribution with 20 ms mean and 4 ms variance [7].

To analyze the effect of the data link bandwidth on the fingerprinting accuracy, we bridge our SDN network to the Internet using 100 Mbps and 1 Gbps links (respectively), by means of a firewall running on an AMD Athlon dual core processor 3800+ machine. For the purpose of our experiments, we collect measurement traces between an Intel Xeon E3-1230 3.20 GHz CPU server with 16 GB RAM and 20 remote clients deployed across the globe. Table I details the specifications and locations of the clients used in our experiments. In our testbed, the server and the software switch were co-located on the same machine.

Note that, by reducing the time required for rule installation to a minimum, our testbed emulates a scenario that is particularly hard for fingerprinting. In Section IV-C, we discuss the implications of our setup on our findings.

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

TABLE I

REMOTE CLIENTS USED IN OUR EXPERIMENTS. BANDWIDTHS ARE BASED ON ESTIMATES FROM THE CLOUD PROVIDERS

	Location	Profile Details
Amazon	Ireland, Europe	1-core Intel Xeon 2.5 GHz CPU, 100-250 Mbps
		1-core Intel Xeon 2.5 GHz CPU, 100-250 Mbps
		1-core Intel Xeon 2.5 GHz CPU, 250 Mbps
		2-core Intel Xeon 2.5 GHz CPU, 250 Mbps
		4-core Intel Xeon 2.5 GHz CPU, 1 Gbps
	Sydney, Australia	1-core Intel Xeon 2.5 GHz CPU, 100-250 Mbps
		2-core Intel Xeon 2.5 GHz CPU, 250 Mbps
	Oregon, USA	1-core Intel Xeon 2.5 GHz CPU, 100-250 Mbps
		1-core Intel Xeon 2.5 GHz CPU, 250 Mbps
		4-core Intel Xeon 2.5 GHz CPU, 1 Gbps
Azure	The Netherlands, Europe	1-core AMD Opteron 2.1 GHz CPU, 30–40 Mbps†
		2-core Intel Xeon 2.2 GHz CPU, 200 Mbps
		8-core AMD Opteron 2.1 GHz CPU, 800 Mbps
		1-core Intel Xeon 2.2 GHz CPU, 100 Mbps
		4-core Intel Xeon 2.2 GHz CPU, 400 Mbps
	Singapore, Asia	1-core Intel Xeon 2.2 GHz CPU, 100 Mbps
		4-core Intel Xeon 2.2 GHz CPU, 400 Mbps
	California, USA	1-core AMD Opteron 2.1 GHz CPU, 30–40 Mbps†
		2-core Intel Xeon 2.2 GHz CPU, 200 Mbps
		8-core AMD Opteron 2.1 GHz CPU, 800 Mbps

† These bandwidths were obtained using measurements.

hop i; that is, $\tau_{L^i} = \frac{S}{B_{L^i}}$, where S is the size of the packet and B_{L^i} is the capacity of L^i. Furthermore, let $d^j_{L^i}$ refer to the additional delay that is experienced by packet j when traversing L^i. The delay $d^j_{L^i}$ generally results from additional queuing exhibited by packet j due to cross-traffic at hop i.

To identify a communication event between the switches and the controller, we rely on two time-based features: packet-pair dispersion and RTT.

1) *Packet-Pair Dispersion:* The *dispersion* between two packets sent by the client after a link L^i_{cs} refers to the time interval between the complete transmission of these packets on L^i_{cs}. When measuring the dispersion of a packet pair traversing an SDN network, two cases emerge.

a) Case 1—Packets do Not Trigger Rule Installation: Assuming that two probe packets (labeled in the sequel by "1" and "2") are sent by the client with an initial dispersion $_0$, and that they do not trigger any interaction on the control plane, then the resulting dispersion measured after a link L^i_{cs} is given by [8], [18]:

$$_i = (_{i-1}) + (d^2_{L^i_{cs}} - d^1_{L^i_{cs}}) \quad \text{otherwise}. \quad \tau_{L^i_{cs}} + d^2_{L^i_{cs}} \quad \text{if } \tau_{L^i_{cs}} + d^1_{L^i_{cs}} \geq _{(i-1)} \qquad (1)$$

In our setup, the client sends *large* packet pairs back-to-back in time with an initial dispersion $_0 = \frac{S}{B_{L^0_{cs}}}$. These packets are then highly likely to queue at the bottleneck link (the link for which B^{cs}_i is minimal). Let *min* be the index of the bottleneck link on the internet path P_{cs}. Following from Equation 1, $_n$ (measured by the server) is then given by:

$$_n = \frac{S}{B^{cs}_{min}} + d^2_{L^{min}_{cs}} + \sum_{i=min+1}^{n} (d^2_{L^i_{cs}} - d^1_{L^i_{cs}}),$$

where $d^2_{L^{min}_{cs}}$ refers to the additional queuing delay that is experienced by packet 2 at the bottleneck link. If the server immediately issues small replies to the packets sent by the client (e.g., by issuing ACKs), then these packets are unlikely to queue on the reverse path P_{sc}, and the measured dispersion between the reply packets will approximately correspond to $_n$ [8]. As shown in [8] and [18], the packet-pair dispersion is a feature that is relatively stable over time (since it depends on the bottleneck bandwidth of the path)—assuming moderate noise traffic crossing the link.

b) Case 2—Packets Trigger Rule Installation: $_n$ is typically in the order of tens of microseconds in current Internet paths [19]. However, we expect $_n$ to increase (e.g., to the order of few milliseconds) if the probe packet pair triggered an interaction on the control plane. This is mainly due to the (relatively slow) handling of a notification by the controller.[1] Namely, when the packet pair triggers an interaction on the control plane, then:

$$_n = \frac{S}{B^{cs}_{min}} + d^2_{L^{min}_{cs}} + \sum_{i=min+1}^{n} (d^2_{L^i_{cs}} - d^1_{L^i_{cs}}) + \max_k \delta^l_k.$$

Here, δ^l_k refers to the delay introduced by a possible communication between the controller and OpenFlow switch k on the path between the sender and receiver. Since we assume that the controller installs bi-directional rules on all switches at once, only the *maximum* installation delay is accounted (and is only witnessed when packets traverse P_{cs}).

2) *Round Trip Times (RTT):* The RTT witnessed by a packet i sent from the client to the server is:

$$RTT_i = \sum^{n} (\tau_{L^j_{cs}} + d^l_{L^j_{cs}}) + \sum^{m} (\tau_{L^j_{sc}} + d^l_{L^j_{sc}}) + \max_k \delta^l_k. \qquad (2)$$

[1] Before any packet is forwarded by the switch, it undergoes the following steps: *(i)* the packet (or just its header) is transmitted to the controller; *(ii)* the controller performs a table-lookup in order to invoke the corresponding forwarding rule for the packet; *(iii)* the decision is transmitted to the involved

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

rienced by the second packet on the bottleneck link. Notice that in the absence of cross-traffic, $n \approx \frac{S}{B_{min}^{cs}}$. If the server

Clearly, if no communication between switch k and the controller occurs (e.g., forwarding rules are already installed), then $\delta_k^i = 0$. Since there might be more than one OpenFlow switch on P_{cs}, RTT_i depends on the maximum latency incurred by a switch-controller interaction across all the OpenFlow switches included in P_{cs}.

Since the RTT exhibited by packets largely depends on the geographical location of hosts, and on the underlying network condition, we measure in our experiments the difference, δ_{RTT}, between the RTT of two probe packets issued by the same sender, i.e., $\delta_{RTT} = RTT_1 - RTT_2$. This feature does not depend on the location of hosts, but is mainly dominated by rule installation overhead and network jitter. Namely, following from Equation 2:

Otherwise, if one of the packets triggers a rule installation, then $|\delta_{RTT}| 0$, since $\max_{\forall k} \delta_k^1 0$ or $\max_{\forall k} \delta_k^2 0$.

C. Data Collection

To collect timing information based on our features, we deployed 20 remote clients across the globe (cf. Table I) that exchange UDP-based probe packet trains with the local server. Notice that we rely on UDP for transmitting packets since Internet gateways may filter TCP SYN or ICMP packets.

Each probe train consists of:

- A `CLEAR` packet signaling the start of the measurements. Upon reception of this packet, the controller deletes all the entries stored within the flow tables of the OpenFlow switches in P_{cs}.
- After one second[2] since the transmission of the `CLEAR` packet, the client transmits four MTU-sized packet pairs. Here, different packet pairs are sent with an additional second of separation.
- After one second since the transmission of the last packet pair, another `CLEAR` packet is sent to clear all flow tables.
- Two packets separated by one second finally close the probe train.

We point out that all of our probe packets belong to the same network flow, i.e., they are crafted with the same packet header. For each received packet of every train, the local server issues a short reply (e.g., a 64 bytes ACK). We maintain a detailed log of the timing information relevant to the sending and reception of the exchanged probe packets. When measuring dispersion, we account for out-of-order packets; this explains negative dispersion values.

For each of our 20 clients, we exchange 450 probe trains on the paths P_{cs} and P_{sc} to the server. Half of these probe trains are exchanged before noon, while the remaining half is exchanged in the evening. In our measurements, we vary the number of OpenFlow switches that need to be configured in reaction to the exchanged probe packets. Namely, we consider the following four cases where a probe packets triggers the reconfiguration of some of the OpenFlow switches: (1) one switch(es) in the form of a flow table entry; *(iv)* the switch installs the entry,

and, finally, the packet is forwarded by the switch.

hardware switch, (2) two hardware switches, (3) three hardware switches, and (4) the software switch. We remark that the choice of the configured hardware switches in our testbed (cf. Figure 1) has no impact on the measured features since we ensure that the remaining hardware switches have already matching rules installed. Furthermore, we remark that packets of a probe train only traverse the software switch in case (4), i.e., when it is configured. In total, our data collection phase lapsed from April 27, 2015 until October 27,2015, in which 869,201 probe packets were exchanged with our local server using all clients/configurations, amounting to almost 0.66 GB of data.

D. Evaluation Metric

We evaluate two hypotheses based on our features: *(i)* the first hypothesis states that no rule installation was triggered by our probe packets and *(ii)* the second hypothesis corresponds to the conjecture that a rule was installed in reaction to our probes. Here, there are two possible errors: *false match* and *false non-match*. In our case, the former is equivalent to a decision that no rule was installed, while in reality our probes triggered the installation of a rule. The latter is equivalent to a decision that a rule was installed, while in reality no rule was installed. The *False Match Rate* (FMR) and *False Non-match Rate* (FNR) represent the frequencies at which these errors occur. The *Equal Error Rate* (EER), which is used as a single metric for the accuracy of an identification system [13], is the rate at which FMR and FNR are equal. In the sequel, we use the EER to evaluate the effectiveness of our features.

We compute the EER as follows. We compute the *Probability Distribution Function* (PDF) of the measured values of our features (across all configurations and clients location) as described in Sections III-A and III-B. We then separate the PDFs in two categories: *(i)* PDF_N that contains all measurements obtained when our probes did not trigger a rule installation, and *(ii)* PDF_Y that contains those measurements obtained when the probe packets caused a rule installation at k OpenFlow switches (with k = 1, 2, 3 hardware switches or k = 1 software switch). We then compute the rate of falsely accepted and falsely rejected hypotheses given a threshold. The measurements from PDF_N that are above this threshold indicate the number of false rejects (FNR), and measurements from PDF_Y that are below the threshold indicate the number of false accepts (FMR). Recall that the EER is the error rate where FNR and FMR are equal. The value of the EER-based threshold is our reference for an accept/reject decision. If the value of a measurement is smaller than the threshold, then we conjecture it belongs to PDF_N; otherwise, we conjecture that it belongs to PDF_Y.

Note that EER values are between 0% and 100%. An EER value for a feature close to 50% indicates that our hypotheses cannot be distinguished from each other for the given feature. In particular, the value 50% means that PDF_N and PDF_Y for the given feature completely overlap, and, based on the

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com
International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)

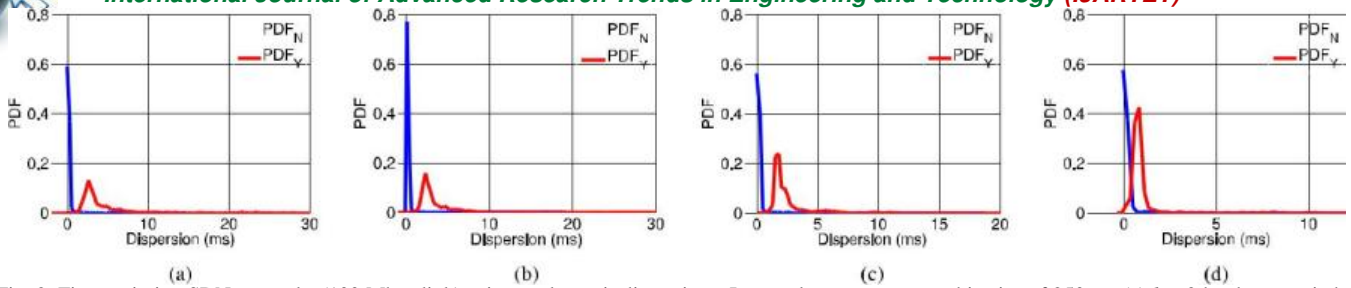

Fig. 2. Fingerprinting SDN networks (100 Mbps link) using packet-pair dispersions. In our plots, we assume a bin size of 250 μs. (a) k = 3 hardware switches. (b) k = 2 hardware switches. (c) k = 1 hardware switch. (d) k = 1 software switch.

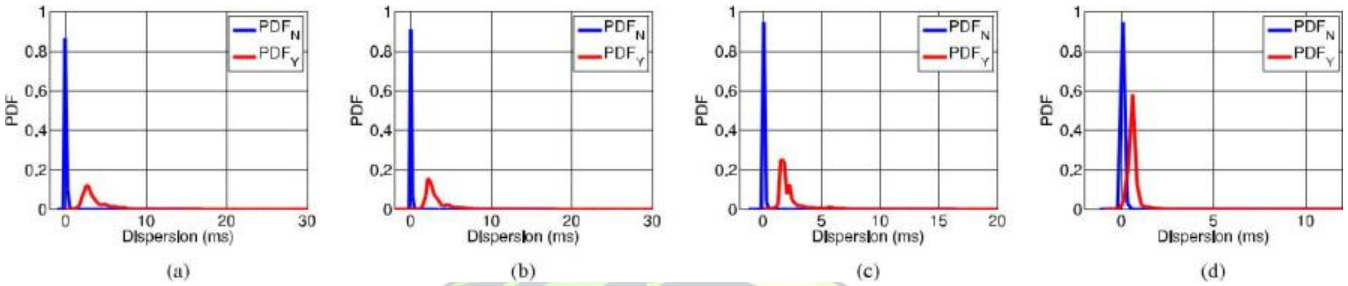

Fig. 3. Fingerprinting SDN networks (1 Gbps link) using packet-pair dispersions. In our plots, we assume a bin size of 250 μs. (a) k = 3 hardware switches. (b) k = 2 hardware switches. (c) k = 1 hardware switch. (d) k = 1 software switch.

feature, an adversary cannot distinguish at all whether a packet triggered a rule installation. Conversely, EER values close to 0% and 100% indicate that our hypotheses are distinguishable based on our features, i.e., the fingerprinting accuracy is high.

increases by almost 0.2% for k = 2, 3 hardware switches when the bandwidth of the data link increases from 100 Mbps to 1 Gbps. In this setting, the EER decreases by 0.5% when k = 1 hardware or software switch.

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

IV. Evaluation Results

In this section, we present and analyze our experimental results using each of our proposed time-based features.

A. Packet-Pair Dispersion Feature

In Figure 2, we show *(i)* the PDF of dispersion values for which none of the packets of a pair triggered any rule installation at the switches (referred to as PDF_N), and *(ii)* the PDF of dispersion values for which the probes triggered a rule installation (referred to as PDF_Y).

As shown in Figure 3, our results are negligibly affected by the data link bandwidth. Notably, the EER marginally

C. Summary of Results

Our evaluation results in Figures 2, 3, 4, and 5 show that fingerprinting attacks on SDN networks are feasible;
in fact, they are already realizable using simple features such as packet-pair dispersions and RTTs.

More specifically, our findings suggest that, irrespective of the number of OpenFlow switches that need to be configured in reaction to a given probe packet, the delay introduced by rule installation, $\max_{\forall k} \delta_k^l$, provides an effective distinguisher for an adversary to identify whether packets are only processed on the fast data plane, or triggers an interaction with the controller on the relatively slow software-based control plane. This delay is clearly distinguishable using the packet-pair dispersion, which is a stable feature over time, and is little affected by the size of the network (i.e., by the number of OpenFlow switches that need to be configured).

Although packet pairs can be easily crafted by an *active* adversary, packet pairs might not always be extractable from existing traffic by a *passive* adversary. However, a passive adversary can monitor existing traffic for packets that share a similar packet header, and are sent apart within a short time interval (e.g., within 10 minutes).
relative difference between the processing speed of packets at the data plane, and at the control plane is even more pronounced. Recall that our testbed was devised to emulate a scenario that is particularly hard for fingerprinting. That is, the controller's CPU was idle most of the time during the measurements; the controller used pre-computed rules when issuing forwarding decision and was connected to a small number of switches (i.e., three); at the time of writing, the deployed OpenFlow hardware switches are among the fastest in installing new flow rules; furthermore, we ensured that the switches' flow tables were empty when performing the measurements, obtaining a flow rule installation time in the order of milliseconds [25], [28]. Hence, it is clear that the fingerprinting accuracy provided by our features only increases when the controller is under heavy load, the data plane bandwidth is larger (e.g., 10 Gbps), or the OpenFlow switches require longer times to update their flow tables. That is, in these settings, the difference in latencies between the data and the control planes will be even more pronounced—which will further increase the accuracy of fingerprinting.

.

Our results indicate that our countermeasure considerably impacts the fingerprinting accuracy of a remote adversary using the dispersion and δ_{RTT} features. More specifically, our countermeasure increases the EER to almost 40% using the dispersion feature, and to 33% using the δ_{RTT} feature when the network comprises three hardware switches. Our countermeasure, however, increases the EER to almost 84% (using both investigated features) when the network com-prises a software switch. Recall that the worst attainable fingerprinting accuracy in this case is when the EER is 50% which signals that the two distributions $P\ D\ F_Y$ and $P\ D\ F_N$ completely overlap. In the case of a software switch, the EER increases to 84% which means that the adversary has an advantage in distinguishing $P\ D\ F_Y$ from $P\ D\ F_N$, in spite of our countermeasure. We believe that this discrepancy mainly originates from the fact that the estimated Generalized-Pareto distribution does not emulate well delays corresponding to software switches (cf. Figure 12).

Similarly, we also argue that lower fingerprinting accuracies can be obtained with our countermeasure if the delay element is equipped with fine-grained delay distributions with respect to the different number of hardware switches that need to be configured in the network. We validate this hypothesis in a separate experiment. Here, we assume the delay element is equipped with best-fit estimates of the distributions of rule installation delays exhibited by both our features with respect to the number of switches in the network, and we measure the corresponding EER witnessed by a remote adversary in our testbed (cf. Figure 10). Our results in Figure 13 confirm our hypothesis, and show that when the delay element is equipped with fine-grained information about the distributions of rule installation delays in the network, the EER is closer to 50%. For example, in this case, the EER increases to almost 40% using both of our features when the network comprises a software switch, and is almost 47% when two hardware switches need to be configured. This shows that our countermeasure considerably reduces the distinguishing advantage of a remote adversary, when fine-grained delay distributions are available to the delay element.

VIII. Conclusion

In this paper, we studied the fingerprinting of SDN networks by a remote adversary. For that purpose, we collected measurements from a large number of hosts located across the globe using a realistic SDN network. Our evaluation shows that, by leveraging information from the RTT and packet-pair dispersion of the exchanged packets, fingerprinting attacks on SDN networks succeed with overwhelming probability. Our results also suggest that fingerprinting attacks are not restricted to active adversaries, but can also be mounted by passive adversaries that capture a snapshot of the traffic exchanged with the SDN network.

Based on our results, we presented and evaluated a counter-measure that leverages the switches' group tables in order to

ISSN 2394-3777 (Print)
ISSN 2394-3785 (Online)
Available online at www.ijartet.com

International Journal of Advanced Research Trends in Engineering and Technology (IJARTET)
Vol. 4, Special Issue 11, March 2017

delay the first few packets of every flow. Our evaluation results show that our countermeasure considerably reduces the ability of an adversary to mount fingerprinting attacks against SDN networks.

ACKNOWLEDGMENTS

This research was partly performed within the 5G-ENSURE project (www.5GEnsure.eu). The views and opinions expressed in this paper are those of the authors and the European Commission is not responsible for any use that may be made of the information the paper contains.

REFERENCES

[1] M. F. Bari *et al.*, "Data center network virtualization: A survey," *IEEE Commun. Surv. Tuts.*, vol. 15, no. 2, pp. 909–928, 2nd Quart., 2013.
[2] R. Bifulco, H. Cui, G. O. Karame, and F. Klaedtke, "Fingerprinting software-defined networks," in *Proc. 23rd Int. Conf. Netw. Proto-cols (ICNP)*, 2015, pp. 453–459.
[3] Christo Ananth, M.Priscilla, B.Nandhini, S.Manju, S.Shafiqa Shalaysha, "Reconstruction of Objects with VSN", International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST), Vol. 1, Issue 1, April 2015, pp:17-20
[4] D. Croce, T. En-Najjary, G. Urvoy-Keller, and E. W. Biersack, "Capacity estimation of ADSL links," in *Proc. 4th ACM Conf. Emerg. Netw. Experim. Technol. (CoNEXT)*, 2008, Art. no. 13.