This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2573756, IEEE Transactions on Information Forensics and Security

On the Fingerprinting of Software-defined Networks

Heng Cui, Ghassan O. Karame, *Member, IEEE,* Felix Klaedtke, and Roberto Bifulco

***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 our study, 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 overwhelming 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 implemented 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], [40], [41], no contribution analyzes the feasibility and realization of fingerprinting 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 fingerprinting 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 comprising 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. Our results suggest that this fingerprinting accuracy is only marginally affected by the number of hardware switches that need to be configured on the path, and by the data link bandwidth. More surprisingly, our results also show that the presence of software switches, which process packets at the software layer, does not hinder the ability of a remote adversary in fingerprinting controller-switch interactions.

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,

Copyright (c) 2013 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

Heng Cui, Ghassan O. Karame, Felix Klaedtke, and Roberto Bifulco are with the NEC Laboratories Europe, Heidelberg, Germany. E-mail: firstname.lastname@neclab.eu.

1556-6013 (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

largely affected by the SDN network size, and significantly deteriorates with time.

This work extends our prior work in [2] (see Section VII for a detailed explanation of our extensions). To the best of our knowledge, this is the first complete work which quantifies—by means of well defined metrics—the success of fingerprinting attacks using state-of-the-art OpenFlow hardware switches. We discuss the implications of our findings on the security of SDN networks, and we show that fingerprinting attacks can expose the SDN network to a number of new threats that are not encountered in traditional networks. 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 control 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. The absence of a controller-switch interaction typically provides evidence that the flow rules that handle the received packet are already installed at the switches. Otherwise, if a communication between the controller and the switches is triggered, 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 conventional 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 installation 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.

B. Features

We define the network path P_{cs} between a client c and a server s as a sequence of consecutive links $P_{cs} = \langle L_{cs}^1, \ldots, L_{cs}^n \rangle$, which are connected via network components. Similarly, we denote the reverse path P_{sc} by $P_{sc} = \langle L_{sc}^1, \ldots, L_{sc}^m \rangle$. We denote by τ_{L^i} the transmission delay at 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 $d_{L^i}^j$ refer to the additional delay that is experienced by packet j when traversing L^i. The delay $d_{L^i}^j$ 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_{cs}^i refers to the time interval between the complete transmission of these packets on L_{cs}^i. When measuring the dispersion of a packet pair traversing an SDN network, two cases emerge.

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 Δ_0, and that they do not trigger any interaction on the control plane, then the resulting dispersion measured after a link L_{cs}^i is given by [8], [18]:

$$\Delta_i = \begin{cases} \tau_{L_{cs}^i} + d_{L_{cs}^i}^2 & \text{if } \tau_{L_{cs}^i} + d_{L_{cs}^i}^1 \geq \Delta_{(i-1)} \\ \Delta_{(i-1)} + (d_{L_{cs}^i}^2 - d_{L_{cs}^i}^1) & \text{otherwise.} \end{cases} \tag{1}$$

In our setup, the client sends *large* packet pairs back-to-back in time with an initial dispersion $\Delta_0 = \frac{S}{B_{L_{cs}^0}}$. These packets are then highly likely to queue at the bottleneck link (the link for which B_i^{cs} is minimal). Let min be the index of the bottleneck link on the internet path P_{cs}. Following from Equation 1, Δ_n (measured by the server) is then given by:

$$\Delta_n = \frac{S}{B_{min}^{cs}} + d_{L_{cs}^{min}}^2 + \sum_{i=min+1}^{n} (d_{L_{cs}^i}^2 - d_{L_{cs}^i}^1)\,,$$

where $d_{L_{cs}^{min}}^2$ refers to the additional queuing delay that is experienced by the second packet on the bottleneck link. Notice

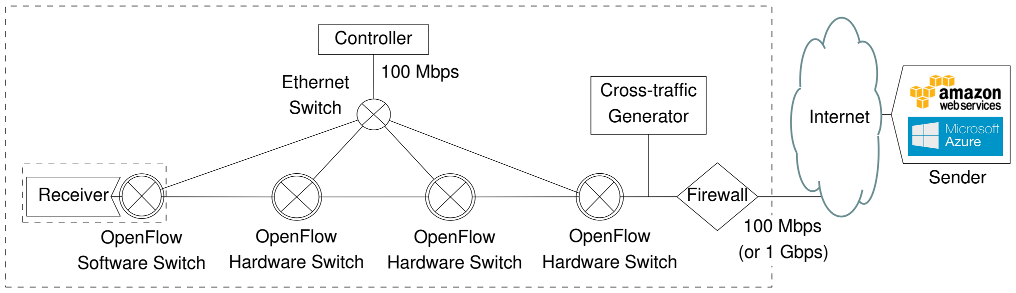


Fig. 1. Sketch of our measurement setup. Our testbed comprises three NEC PF5240 OpenFlow hardware switches and one OpenVSwitch (version 2.3.1).

that in the absence of cross-traffic, $\Delta_n \approx \frac{S}{B^{cs}_{min}}$. 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 Δ_n [8]. As shown in [8], [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.

Case 2—Packets trigger rule installation: Δ_n is typically in the order of tens of microseconds in current Internet paths [19]. However, we expect Δ_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:

$$\Delta_n = \frac{S}{B^{cs}_{min}} + d^2_{L^{min}_{cs}} + \sum_{i=min+1}^{n} (d^2_{L^i_{cs}} - d^1_{L^i_{cs}}) + \max_{\forall k} \delta^i_k .$$

Here, δ^i_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:

$$RTT_i = \sum_{j=1}^{n}(\tau_{L^j_{cs}} + d^i_{L^j_{cs}}) + \sum_{j=1}^{m}(\tau_{L^j_{sc}} + d^i_{L^j_{sc}}) + \max_{\forall k} \delta^i_k . \tag{2}$$

Clearly, if no communication between switch k and the controller occurs (e.g., forwarding rules are already installed), then $\delta^i_k = 0$. 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}.

[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 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.

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:

$$\delta_{RTT} = \max_{\forall k} \delta^1_k - \max_{\forall k} \delta^2_k + \sum_{j=1}^{n} d^i_{L^j_{cs}} + \sum_{j=1}^{m} d^i_{L^j_{sc}} ,$$

where $\sum_{j=1}^{n} d^i_{L^j_{cs}} + \sum_{j=1}^{m} d^i_{L^j_{sc}}$ is typically negligible. If both packets do not result in any rule installation, then $\delta_{RTT} \approx 0$. Otherwise, if one of the packets triggers a rule installation, then $|\delta_{RTT}| \gg 0$, since $\max_{\forall k} \delta^1_k \gg 0$ or $\max_{\forall k} \delta^2_k \gg 0$.

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,

[2]Our experimental results show that one second is enough to account for rule installation on all four OpenFlow switches in our network.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2573756, IEEE Transactions on Information Forensics and Security

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.

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 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 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.

IV. EVALUATION RESULTS

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

1556-6013 (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2573756, IEEE Transactions on Information Forensics and Security

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).

Our results show that the sample mean of PDF_Y is considerably greater than that of PDF_N; PDF_Y and PDF_N are significantly different at 1% according to t-test [45] when the number k of OpenFlow hardware and software switches that need to be configured varies between 1 and 3. This is mainly due to the fact that the delay required for rule installation, $\max_{\forall k} \delta_k^i$, acts as a strong distinguisher when measuring Δ_n. More specifically, our results show that across all locations the obtained EER are approximately 1% for $k = 2, 3$ hardware switches, 1.74% for $k = 1$ hardware switch, and 4.49% for $k = 1$ software switch. For example, when $k = 3$ hardware switches, the EER is calculated using a threshold of 1.43 ms; that is, 98.92% of all measured values in PDF_N are below 1.43 ms, and 98.92% of all measured values above 1.43 ms are contained in PDF_Y.

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

B. RTT Feature

In Figure 4, we plot the PDF of δ_{RTT} values witnessed by probe packets sent within a short time interval (i.e., by the last two probes of our probe train sent 1 second apart) with respect to a varying number of switches. Our results show that, irrespective of the number of switches, the PDF of all δ_{RTT} values collected by our clients for which neither of the two probes triggered any rule installation on the switches (referred to as PDF_N) can be fitted to a normal distribution with mean 0. In contrast, the sample mean of PDF of δ_{RTT} values for which only the first probe (PDF_Y) incurred a rule installation is strictly greater than 0; PDF_Y and PDF_N are significantly different at 1% according to t-test. The EER is approximately 0.43% when $k = 3$ hardware switches, 0.13% when $k = 2$ hardware switches, 1.25% when $k = 1$ hardware switch, and increases to 5.84% when $k = 1$ software switch.

Similar to the dispersion feature, our results are little affected by the speed of the data link (cf. Figure 5). More specifically, the EER was unchanged for $k = 3$ hardware switches and marginally increased by almost 0.7% for $k = 2, 1$ hardware switches when the bandwidth of the data link increases from 100 Mbps to 1 Gbps. Given this change, the EER increased by almost 1.5% when $k = 1$ software switch.

We also compute the EER for δ_{RTT} values measured over a 10 minute span. As shown in Figure 6, our results indicate that δ_{RTT} is not a stable feature over time; for example, when $k = 1$, the EER deteriorates to approximately 5% for a hardware switch, and almost 15% when dealing with a software switch. To further study the stability of δ_{RTT} over a longer period of time (three months), we conducted a separate experiment using four of our nodes located in Europe. Our results (cf. Figure 7) show that PDF_N and PDF_Y are considerably less distinguishable when RTT values are collected over a larger time span; for example, the EER grows to 39.5% for a time span of 3 months when $k = 1$ hardware switch (and to 25.17% and 24.83% when $k = 2$ and $k = 3$, respectively). We believe that this discrepancy is due to changing network conditions, which incur in non-negligible differences in RTT [38]. Namely, our results suggest that the change of the RTT value by a few milliseconds, caused by e.g., a change in the WAN path or traffic conditions, is comparable to the change in the RTT value introduced by an interaction with the controller.

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.

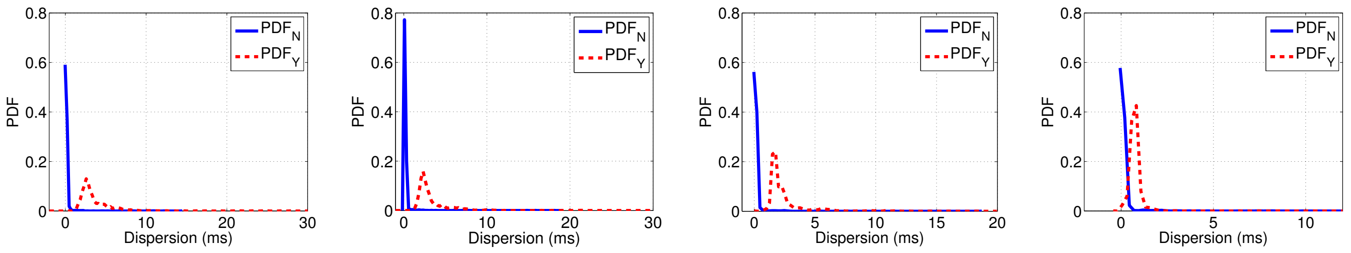
More specifically, our findings suggest that, irrespective of the number of OpenFlow switches that need to be configured in reaction to a given probe packet, the delay introduced by rule installation, $\max_{\forall k} \delta_k^i$, provides an effective distinguisher for an adversary to identify whether 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).

Our findings show that the difference of measured RTT values between two such packets provides evidence for a passive adversary whether any of those packets triggered a reaction on the control plane. Passive fingerprinting is especially detrimental since it limits the applicability of existing intrusion detection systems in detecting SDN fingerprinting attempts; indeed, passive network monitoring does not generate any extra traffic and as such cannot be deterred by relying on anomaly detection. Notice, however, that the RTT feature considerably depends on the SDN network size and is less stable over time when compared to packet-pair dispersions. For instance, the EER almost doubles in our experiments when the number of OpenFlow hardware switches that need to be configured decreases from 3 to 1.

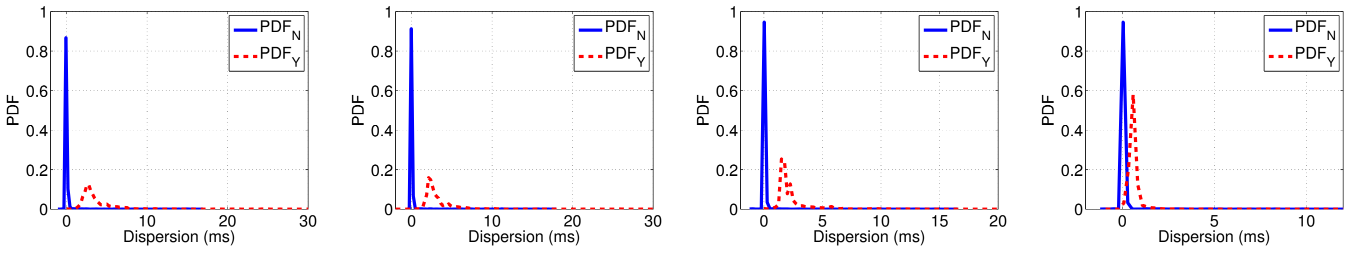
Our results also suggest that the data link bandwidth has little impact on the fingerprinting accuracy for both the RTT and the dispersion features. The presence of a software switch in the communication path, however, considerably deteriorates the EER; even in such a setting, our results nevertheless show that fingerprinting attacks can still be reliably mounted by a remote adversary.

1556-6013 (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.



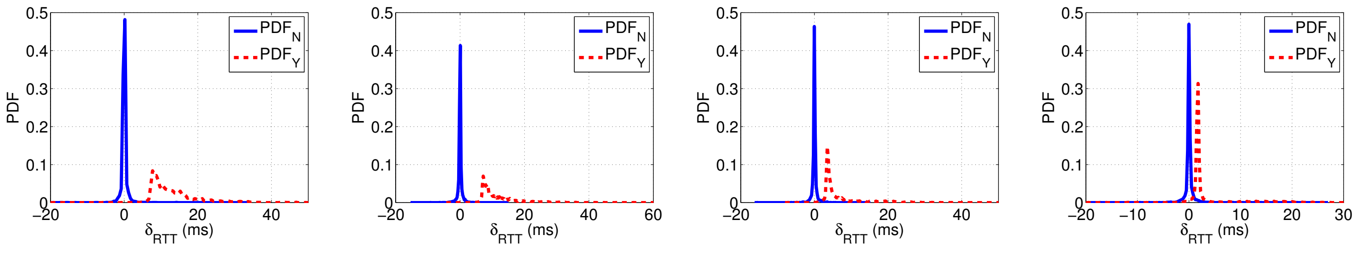
(a) $k = 3$ hardware switches (b) $k = 2$ hardware switches (c) $k = 1$ hardware switch (d) $k = 1$ software switch

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, time span 1 second (b) $k = 2$ hardware switches, time span 1 second (c) $k = 1$ hardware switch, time span 1 second (d) $k = 1$ software switch, time span 1 second

Fig. 4. Fingerprinting SDN networks (100 Mbps link) using δ_{RTT}. In our plots, we assume a bin size of 250 μs.

TABLE II
SUMMARY OF OBTAINED EERS.

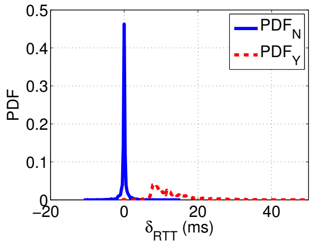
		100 Mbps data link				1 Gbps data link			
		$k = 3$ HW	$k = 2$ HW	$k = 1$ HW	$k = 1$ SW	$k = 3$ HW	$k = 2$ HW	$k = 1$ HW	$k = 1$ SW
Packet-pair Dispersion	EER	1.08%	0.94%	1.74%	4.49%	1.24%	1.19%	1.25%	3.87%
	Threshold	1.43 ms	1.37 ms	1.17 ms	0.37 ms	1.42 ms	1.36 ms	0.88 ms	0.20 ms
δ_{RTT}, time span 1 second	EER	0.43%	0.13%	1.25%	5.84%	0.45%	0.84%	2.04%	7.25%
	Threshold	4.63 ms	4.27 ms	2.13 ms	0.84 ms	4.60 ms	4.85 ms	2.18 ms	0.85 ms

Table II and Figure 8 summarize our results. We argue that our findings apply to other SDN networks in which the 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.

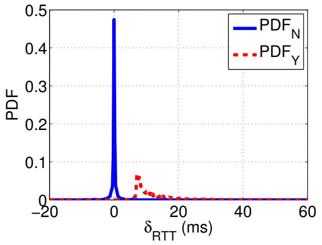
Note, however, that our findings rely on the assumption that there is a single SDN network on the path to the server. Otherwise, while our features still fingerprint controller-switch interactions in more than one SDN network, they do not reveal

1556-6013 (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

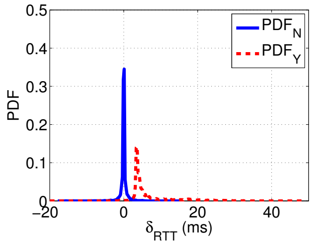
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2573756, IEEE Transactions on Information Forensics and Security



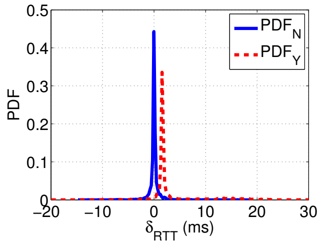
(a) $k = 3$ hardware switches, time span 1 second



(b) $k = 2$ hardware switches, time span 1 second



(c) $k = 1$ hardware switch, time span 1 second



(d) $k = 1$ software switch, time span 1 second

Fig. 5. Fingerprinting SDN networks (1 Gbps link) using δ_{RTT}. In our plots, we assume a bin size of 250 μs.

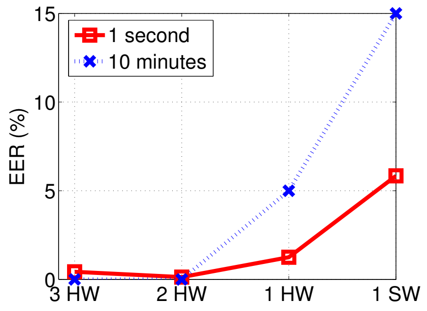


Fig. 6. Impact of the time span on the δ_{RTT} feature.

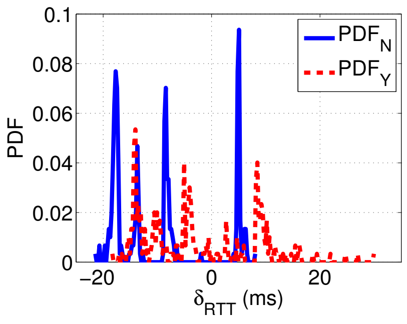
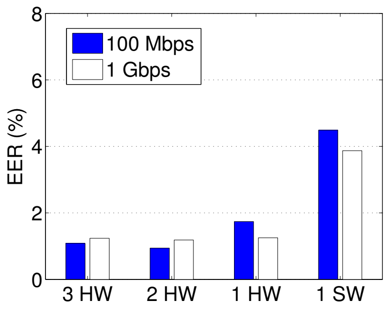
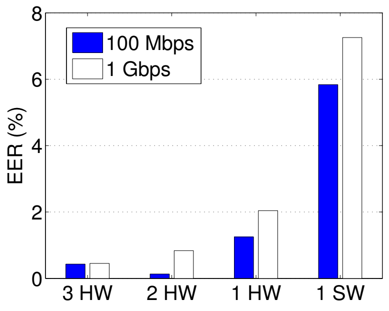


Fig. 7. δ_{RTT} over a period of 3 months when $k = 1$ hardware switch. Here, we assume a bin size of 250 μs.



(a) Dispersion feature.



(b) δ_{RTT} feature.

Fig. 8. Fingerprinting SDN networks with respect to the number of switches and the data link bandwidth.

in which network the interactions take place.

V. Implications

In the previous section, we showed that remote fingerprinting attacks on SDN networks are feasible and easily realizable by means of our proposed features. In what follows, we discuss the implications of our findings on the security of SDN networks.

A. Rule Scanning

Based on our findings, a remote adversary can clearly infer whether a flow rule has been already installed by the controller in order to handle a specific type of traffic or route towards a given destination. For example, the adversary can craft probe packets whose headers match the traffic type and/or destination address and infer by measuring the timing of the packets whether these packets triggered the installation of a rule. This provides a strong evidence for the adversary that e.g., communication with the given destination address has recently occurred. Depending on the underlying rule, the adversary might also be able to infer the used network protocol, and the destination port address. By doing so, the adversary obtains additional information about the occurrence of a particular communication event; for example, the adversary can infer whether the destination address has recently established an SSL session to perform an e-banking transaction. Notice that this leakage is only particular to SDN networks, and does not apply to traditional networks.

Moreover, the remote fingerprinting of rules enables the adversary to better understand the logic adopted by the controller in managing the SDN network. This includes inferring the timeouts set for the expiry of specific rules, whether the controller aims at fine-grained or coarse-grained control in the network, etc. Similar to existing port and traffic scanners, this knowledge can empower the adversary with the necessary means to compromise the SDN network. Even worse, the adversary can leverage this knowledge in order to attack other networks which implement a similar rule installation logic. For instance, in a geographically dispersed datacenter, different sub-domains typically implement the same policies. The adversary can train using one sub-domain and leverage the acquired knowledge in order to compromise another sub-domain.

B. Denial-of-Service Attacks

The rule space is a scarce resource in existing hardware switches. Namely, state-of-the-art OpenFlow hardware switches can only accommodate few tens of thousands rules [20], and only support a limited number of flow-table updates per second [3], [25]. While these limitations can be circumvented by means of a careful design of the rule installation logic, an adversary that knows which packets cause an interaction with the controller can abuse this knowledge to launch Denial-of-Service (DoS) attacks.

For instance, an adversary might simply try to overload the controller with handling *packet-in* events. More specifically, the adversary sends packets, where each of them most likely triggers a controller-switch interaction. Too many such interactions will overload the controller.

Another kind of DoS attack is to fill up the switches' flow tables. An analogy to this is when a computer runs out of memory and starts swapping. Usually, the computer becomes unusable. Similarly, the network performance is severely harmed when the flow tables are full (or even almost full). First, installing flow rules in an almost full table is more costly than in an almost empty flow table. Second, in case the flow table is full, either new network flows cannot be established, which would already be a DoS, or some installed flow rules need to be deleted. However, in general, it is not obvious which rules should be deleted to make room for new rules; this needs to be coordinated by the controller and is a complex operation, which can quickly overload the controller and the switches. For example, the deletion of a rule of an ongoing network flow might entail the rule's immediate reinstallation. This can escalate and the controller will have to constantly delete and reinstall rules.

An adversary can make both kinds of DoS attacks more likely to succeed by first passively fingerprinting the network traffic, instead of blindly guessing which packets trigger a controller-switch interaction.

VI. Proposed Countermeasure

In this section, we present an efficient countermeasure to prevent fingerprinting attacks on SDN networks. We also evaluate the effectiveness of our countermeasure in our testbed.

A. Our Countermeasure

One possible countermeasure against fingerprinting would be for the switches to delay every received packet before forwarding it. This countermeasure is clearly inefficient as it would severely harm network performance. Randomly deleting flow rules and reinstalling them when receiving the corresponding *packet-in* events also does not solve the problem either. First, additional interactions between the controller and the switches are introduced, resulting in an additional burden on the controller. Second, installing flow rules is a costly switch operation. In what follows, we sketch an efficient countermeasure, which relies on only delaying the first few packets of a network flow.

Our proposal does not concern the handling of new flows, but focuses on processing packets which pertain to existing flows. Namely, for a packet of an existing flow, we leverage the group table [31] and the internal timer maintained by a switch to identify whether this flow has recently appeared. The group tables are used in OpenFlow switches to describe per-packet forwarding actions. They allow one to realize forwarding strategies, such as ECMP [44], which could not be achieved using the flow table abstraction that describes only per-flow forwarding actions. A group table contains one or more *buckets*, which in turn contain an action set, similar to the one contained in the flow rules. A group table is further associated with a bucket selection logic, which is related to the group table *type*. For example, a group table of type "fast failover" implements a selection logic that associates each bucket to a switch's port. Then, the logic selects the first bucket in the table whose associated switch's port status is *live*. The action *group* in a flow rule's action set enables one at selecting which packets should be processed by which group.

Our proposed countermeasure (cf. Figure 9) defines a new bucket selection logic for the group table, such that packets of active flows are immediately forwarded, while packets of inactive flows are forwarded onto a special port that connects the switch to a network delay element. Our selection logic considers a flow to be inactive if no packets for such a flow were received by the switch in a threshold amount of time, T_{th}, which is measured (in seconds) by the switch's internal timer.

The first received packet of an inactive flow is delayed by $\delta_{RTT} \approx \max_{\forall k} \delta_k^i$, which gives the adversary little advantage in identifying whether the additional delay measured by the RTT feature is caused by a controller-switch interaction or is artificially introduced by our countermeasure. Moreover, all packets of the same flow received within a short time window W are also delayed by a small Δ; this procedure prevents fingerprinting attempts that leverage the dispersion feature. As shown in Section VI-B, Δ and δ_{RTT} can be fitted to pre-determined distributions, depending on the network size and the number of hops on the communication path. Alternatively, the controller can estimate the distributions corresponding to Δ and δ_{RTT} through a feedback loop.

Notice that our countermeasure is unlikely to deteriorate network performance, since only few packets per flow are delayed by few milliseconds (cf. Section VI-B). We further remark that our proposal requires minor modifications—which are supported to a large extent already in the OpenFlow v1.3 specification—by the switches' manufacturers. As such, we argue that our proposal can be efficiently implemented (in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2573756, IEEE Transactions on Information Forensics and Security

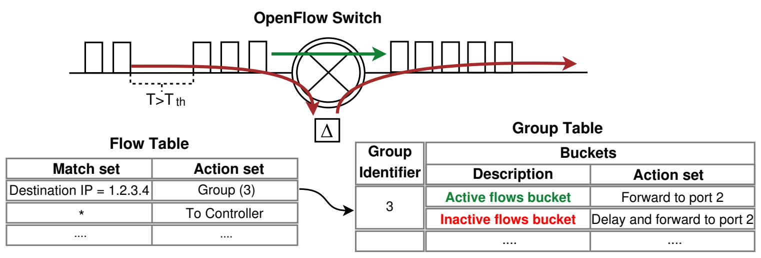


Fig. 9. Sketch of our countermeasure. The packets with destination IP 1.2.3.4 are processed by the group table 3, which implements the bucket selection logic specified by our countermeasure. If no packets for this network flow are processed by the switch for a time $T > T_{th}$, then the next few packets of the flows are delayed by an appropriate amount.

hardware) within the switches.

B. Evaluation

We evaluate the effectiveness of our proposed countermeasure using the testbed described in Section III-C. For that purpose, we connect a delay element, running on Intel Xeon 2.8 GHz CPU with 4 GB of RAM, to a reserved port on the outermost switch; as described later, this element delays network packets by a specified amount before outputting them back on another port of the switch (cf. Figure 10).

Since our countermeasure requires a modification to the bucket selection logic by the switches' manufacturers, we emulate this logic by tagging the first few packets of inactive flows, and by pre-installing a rule which forwards such tagged packets to the delay element.

Packets are delayed by the delay element according to a pre-determined distribution. To select the best-fit distribution, we applied the Kolmogorov-Smirnov test [24] on a number of candidate distributions, such as Pareto, Generalized Pareto, Weibull, using our collected measurements as ground truth. Our results show that the Generalized Pareto distribution achieves the highest Kolmogorov-Smirnov score test (of approximately 0.09 for dispersion and 0.08 for δ_{RTT}) for both our investigated features (cf. Figure 11). Recall that the Generalized Pareto distribution is of the form:

$$f_{(\xi,\mu,\sigma)}(x) = \frac{1}{\sigma}\left(1 + \frac{\xi(x-\mu)}{\sigma}\right)^{(-\frac{1}{\xi}-1)}$$

Table III summarizes the parameters for Generalized Pareto distributions extracted from our measurements, and used by the delay element to prevent fingerprinting attempts using the dispersion and the δ_{RTT} features. Namely, in our countermeasure, the delay element inserts a delay before transmitting the first packet by randomly sampling from a Generalized Pareto distribution with parameters $\xi = -0.53, \sigma = 10.58$, and $\mu = 0.57$, while all subsequent packets sent within an interval of 100 ms are delayed by randomly sampling from a Generalized Pareto distribution with parameters $\xi = -0.60, \sigma = 2.86$, and $\mu = 0.45$ (to prevent fingerprinting using the dispersion feature).

Results: In the remainder of this section, we report on the effectiveness of our proposed countermeasure using the testbed shown in Figure 10. Here, we collect our measurements using the same process described in Section III-C. Our results are shown in Table IV.

TABLE III
PARAMETERS OF THE GENERALIZED PARETO DISTRIBUTIONS USED BY THE DELAY ELEMENT.

	ξ	σ	μ
Packet-pair Dispersion	-0.60	2.86	0.45
δ_{RTT}	-0.53	10.58	0.57

TABLE IV
SUMMARY OF MEASURED EERS USING OUR COUNTERMEASURE.

	$k = 3$ HW	$k = 2$ HW	$k = 1$ HW	$k = 1$ SW
Packet-pair Dispersion	39.76%	46.25%	61.18%	84.57%
δ_{RTT}	33.42%	37.48%	67.19%	83.11%

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 comprises 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 PDF_Y and PDF_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 PDF_Y from PDF_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

1556-6013 (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2573756, IEEE Transactions on Information Forensics and Security

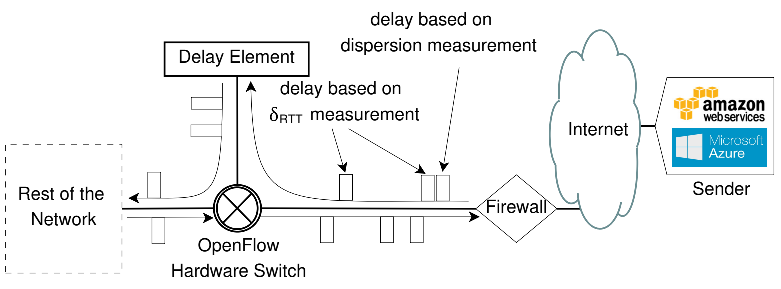
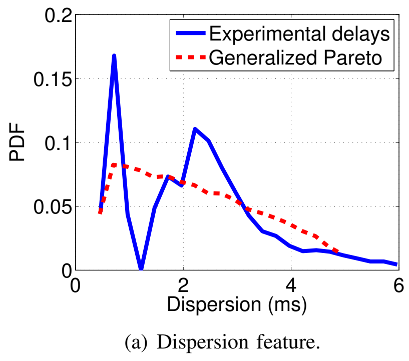
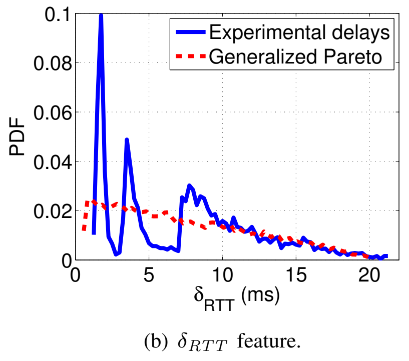


Fig. 10. Modified testbed used to evaluate our countermeasure.

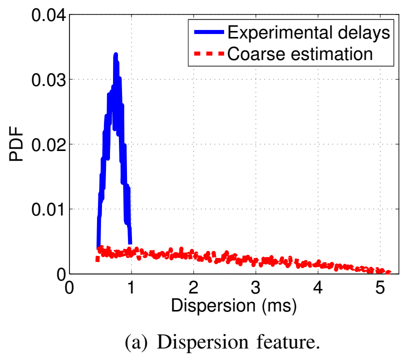


(a) Dispersion feature.

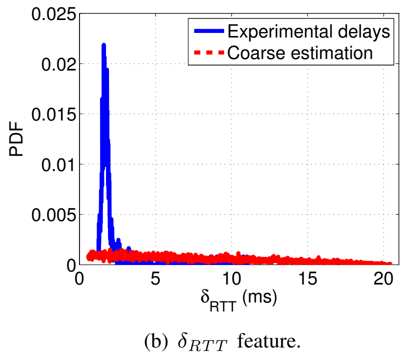


(b) δ_{RTT} feature.

Fig. 11. Fitting experimental data to the Generalized Pareto distribution.



(a) Dispersion feature.



(b) δ_{RTT} feature.

Fig. 12. Software switch experimental delays vs. the Generalized Pareto distribution.

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.

VII. Related Work

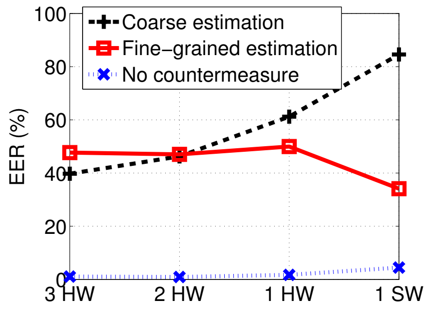
This paper extends our prior work on fingerprinting SDN networks [2]. The additional contributions are summarized as follows. *(i)* Our evaluation shows that fingerprinting of SDN networks with software switches is also feasible. *(ii)* We also investigate the fingerprinting accuracy when the link bandwidth increases to 1 Gbps. *(iii)* We conducted further measurements to evaluate the effectiveness of fingerprinting SDN networks over a substantial period of time. *(iv)* We discuss the implications of our findings on the security of SDN networks. *(v)* Finally, we analyze and evaluate our countermeasure.

In the remainder of this section, we discuss related work in the areas of network fingerprinting and SDN security.

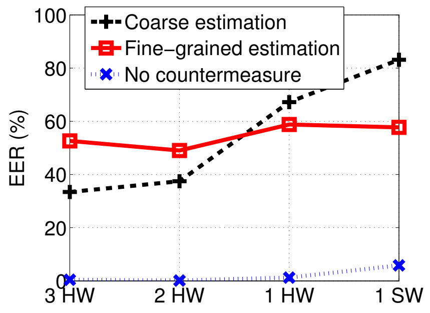
A. Network Fingerprinting

Network fingerprinting has attracted considerable attention in the research community. Markopoulou *et al.* [26] show that network delays in backbone networks are relatively stable and are only marginally affected by network congestion. Dischinger *et al.* [6] show that network features such as bandwidth and delays mainly depend on the last-mile hops in residential networks (e.g., due to ISP traffic shaping). Schulman and Spring [37] extend this observation by showing that end-to-end delays in residential networks are largely affected by weather conditions. These findings confirm our observation that the RTT is not a stable feature over time.

1556-6013 (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.



(a) Dispersion feature.



(b) δ_{RTT} feature.

Fig. 13. Impact of the estimation of the delay distribution on the measured EER.

Packet-pair dispersion was first proposed to estimate available bandwidth [16], [21], [43] and the bottleneck bandwidth of a network path [4], [9], [18], [19], [36]. Sinha *et al.* [42] observe that the distribution of packet-pair dispersions can be used to fingerprint the Internet paths. Karame *et al.* [18] show that the packet-pair dispersion technique is a stable feature which can be used to characterize Internet paths. In [47], Zeitlin shows in a closed testbed that packet latencies offer a strong distinguisher between SDN networks and non-SDN networks, and one can determine, by measuring packet latencies, whether a specific flow rule is installed at a switch. Our experiments with our testbed that comprises multiple clients across the globe confirm these observations. Furthermore, we quantify the accuracy of fingerprinting the interactions between the controller and the switches using latency-dependent features, such as RTT and packet-pair dispersion.

B. SDN Security

A comprehensive survey of security issues in SDN can be found in [23].

Shin and Gu [39] briefly hint on the possibility of fingerprinting SDN networks by leveraging timing information of the exchanged packets; however, in contrast to our work, their study is not based on a real-world evaluation and does not provide any metrics to quantify fingerprinting accuracy. So-called topology attacks of SDN networks are studied in [5], [15]. Dhawan *et al.* [5] also discuss DoS attacks, similar to the ones outlined in Section V. They describe an extension of the controller with a monitoring unit, which detects and reports abnormal behavior.

Shin *et al.* [40] outline a number of vulnerabilities in current SDN controllers such as Floodlight [12], Beacon [10], OpenDaylight [30], and POX [34]; these vulnerabilities allow malicious applications to tamper with the internal data structures maintained by the SDN controller in order to attack the entire SDN network. The authors propose a sandbox approach for network applications to deter this misbehavior. Further mechanisms for securing the control plane are proposed in [33]. Similarly, AVANT-GUARD [41] proposes two data plane extensions to enhance the resilience of an SDN network against network flooding attacks and expedite access to critical data plane activity patterns. We point out that these prior security solutions do not deter fingerprinting attacks on SDN networks. To the best of our knowledge, our countermeasure emerges as the only workable solution to alleviate SDN fingerprinting attacks. Moreover, we are the first to discuss information leakage concerning the packet-forwarding logic in SDN.

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 countermeasure that leverages the switches' group tables in order to 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) and received funding from the EU Framework Programme for Research and Innovation Horizon 2020 under grant agreement no. 671562. 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, R. Boutaba, R. P. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani. Data center network virtualization: A survey. *IEEE Communications Surveys and Tutorials*, 15(2):909–928, 2013.

[2] R. Bifulco, H. Cui, G. O. Karame, and F. Klaedtke. Fingerprinting software-defined networks. In *Proceedings of the 23rd International Conference on Network Protocols (ICNP)*. IEEE Computer Society, 2015.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2573756, IEEE Transactions on Information Forensics and Security

[3] R. Bifulco and M. Dusi. Position paper: Reactive logic in software-defined networking: Accounting for the limitations of the switches. In *Proceedings of the 3rd European Workshop on Software Defined Networks (EWSDN)*, pages 97–102. IEEE Computer Society, 2014.

[4] D. Croce, T. En-Najjary, G. Urvoy-Keller, and E. Biersack. Capacity estimation of ADSL links. In *Proceedings of the 4th ACM Conference on Emerging Network Experiments and Technology (CoNEXT)*. ACM Press, 2008.

[5] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. SPHINX: Detecing security attack in software-defined networks. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2015.

[6] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu. Characterizing residential broadband networks. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 43–56. ACM Press, 2007.

[7] D. Dobre, G. Karame, W. Li, M. Majuntke, N. Suri, and M. Vukolić. PoWerStore: Proofs of writing for efficient and robust storage. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security (CCS)*, pages 285–298. ACM Press, 2013.

[8] C. Dovrolis, P. Ramanathan, and D. Moore. What do packet dispersion techniques measure. In *Proceedings of the 20th Anual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 905–914. IEEE Computer Society, 2001.

[9] C. Dovrolis, P. Ramanathan, and D. Moore. Packet-dispersion techniques and a capacity-estimation methodology. *IEEE/ACM Transactions on Networking*, 12(6):963–977, 2004.

[10] D. Erickson. The Beacon OpenFlow controller. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 13–18. ACM Press, 2013.

[11] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 543–546. USENIX Association, 2014.

[12] Floodlight SDN Controller. http://www.projectfloodlight.org/floodlight/.

[13] Fingerprint Verification Competitions (FVC). http://bias.csr.unibo.it/fvc2006/.

[14] N. Gude, T. Koponen, J. Petit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an operating system for networks. *SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[15] S. Hong, L. Xu, K. Wang, and G. Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2015.

[16] N. Hu and P. Steenkiste. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, 21(6):879–894, 2003.

[17] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. SoftCell: Scalable and flexible cellular core network architecture. In *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 163–174. ACM Press, 2013.

[18] G. O. Karame, B. Danev, C. Bannwart, and S. Capkun. On the security of end-to-end measurements based on packet-pair dispersions. *IEEE Transactions on Information Forensics and Security*, 8(1):149–162, 2013.

[19] G. O. Karame, D. Gubler, and S. Capkun. On the security of bottleneck bandwidth estimation techniques. In *Proceedings of the 5th International ICST Conference on Security and Privacy in Communication Networks (SecureComm)*, volume 19 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 121–141. Springer, 2009.

[20] N. Katta, J. Rexford, and D. Walker. Infinite CacheFlow in software-defined networks. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*. ACM Press, 2014.

[21] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of the 6th ACM SIGCOMM Conference on Communications Architecture and Protocols (SIGCOMM)*, pages 3–15. ACM Press, 1991.

[22] R. Kloti, V. Kotronis, and P. Smith. OpenFlow: A security analysis. In *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP)*. IEEE Computer Society, 2013.

[23] D. Kreutz, F. M. V. Ramos, P. J. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[24] Kolmogorov-Smirnov test. https://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test/.

[25] A. Lazaris, D. Tahara, X. Huang, L. E. Li, A. Voellmy, Y. R. Yang, and M. Yu. Tango: Simplifying SDN programming with automatic switch behavior inference, abstraction, and optimization. In *Proceedings of the 10th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 199–212. ACM Press, 2014.

[26] A. Markopoulou, F. Tobagi, and M. Karam. Loss and delay measurements of internet backbones. *Computer Communications*, 29(10):1590–1604, 2006.

[27] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[28] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee. DevoFlow: Cost-effective flow management for high performance enterprise networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*. ACM Press, 2010.

[29] NEC ProgrammableFlow PF5240 Switch. http://www.necam.com/docs/?id=5ce9b8d9-e3f3-41de-a5c2-6bd7c9b37246.

[30] OpenDaylight SDN Controller. http://www.opendaylight.org/.

[31] OpenFlow switch specification—version 1.3.0 (wire protocol 0x04). Open Networking Foundation, 2012. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[32] B. Pfaff, J. Petit, T. Koponen, K. Amidon, M. Casado, and S. Shenker. Extending networking into the virtualization layer. In *Proceedings of the 8th ACM Workshop on Hot Topics in Networks (HotNets)*. ACM Press, 2009.

[33] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran. Securing the software-defined network control layer. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2015.

[34] POX SDN Controller. http://www.noxrepo.org/pox/about-pox/.

[35] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying middlebox policy enforcement using SDN. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 27–38. ACM Press, 2013.

[36] S. Saroiu, P. K. Gummadi, and S. D. Gribble. SProbe: A fast technique for measuring bottleneck bandwidth in uncooperative environments. http://sprobe.cs.washington.edu/sprobe.ps, 2002.

[37] A. Schulman and N. Spring. Pingin' in the rain. In *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 19–28. ACM Press, 2011.

[38] Y. Schwartz, Y. Shavitt, and U. Weinsberg. A measurement study of the origins of end-to-end delay variations. In *Proceedings of the 11th International Conference on Passive and Active Measurement (PAM)*, volume 6032 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2010.

[39] S. Shin and G. Gu. Attacking Software-Defined Networks: A First Feasibility Study. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 165–166. ACM Press, 2013.

[40] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer Communications Security (CCS)*, pages 78–89. ACM Press, 2014.

[41] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security (CCS)*, pages 413–424. ACM Press, 2013.

[42] R. Sinha, C. Papadopoulos, and J. Heidemann. Fingerprinting internet paths using packet pair dispersion. Technical Report 06-876, Department of Computer Science, University of Southern California, 2006.

[43] J. Strauss, D. Katabi, and F. Kaashoek. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 39–44. ACM Press, 2003.

[44] D. Thaler and C. Hopps. Multipath issues in unicast and multicast next-hop selection, 2000. https://tools.ietf.org/html/rfc2991.

[45] t-test. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Student%27s_t-test.

[46] A. Voellmy, J. Wang, Y. R. Yang, B. Ford, and P. Hudak. Maple: Simplifying SDN programming using algorithmic policies. In *Proceedings*

1556-6013 (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2016.2573756, IEEE Transactions on Information Forensics and Security

of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), pages 87–98. ACM Press, 2013.

[47] Z. J. Zeitlin. Fingerprinting software-defined networks and controllers. Master's thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, March 2015. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA615336.

Heng CUI received his M.Sc degree in Computer Communication Networks from Politecnico di Torino, Italy in 2009, and a Ph.D degree from Telecom ParisTech, France in 2013. From 2014, he worked as a Research Scientist at NEC Laboratories Europe in Heidelberg, Germany. From January 2016, he is working as a Site Reliability Engineer in Zalando SE in Berlin, Germany.

Ghassan Karame is a Senior Researcher at NEC Laboratories Europe. He received his Masters of Science in Information Networking from Carnegie Mellon University (CMU) in December 2006, and his PhD degree in Computer Science from ETH Zurich, Switzerland, in 2011. Between 2011 and 2012, he worked as a postdoctoral researcher in the Institute of Information Security of ETH Zurich. Dr. Karame is interested in all aspects of security and privacy with a focus on cloud security, SDN/network security, and Bitcoin security. He is a member of the IEEE and of the ACM. More information about him can be found in the web page http://ghassankarame.com/.

Felix Klaedtke received his PhD degree from Albert-Ludwigs University Freiburg, Germany, in 2004. He joined the NEC Laboratories Europe in Heidelberg in 2013, where he is a senior researcher in the lab's Security Group. In 2000 and 2001, he was an International Research Fellow in 2000 and 2001 at SRI International, Menlo Park, CA, USA, and from 2004 to 2013, he was a researcher and lecturer in the Information Security Group at ETH Zurich, Switzerland. His research interests are in building correct and secure IT systems. His current research focuses on security in software-defined networks, enforcement of security policies, and runtime verification.

Roberto Bifulco is a researcher in telecommunications and, in particular, programmable networks. His research falls in the fields of Software-defined Networking and Network Function Virtualization. Since 2012, Roberto joined the NEC Laboratories in Heidelberg, Germany, where he is a Senior Researcher. Earlier, Roberto worked as consultant for small technological enterprises and start-ups. During his career, Roberto has taken part to several research projects funded by the European Commission, such as FP7 ONELAB2, FP7 MPLANE and H2020 BEBA. He holds a Ph.D. from University of Napoli "Federico II".

1556-6013 (c) 2016 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.