Control Plane Reflection Attacks in SDNs: New Attacks and Countermeasures

Menghao Zhang[1,2,3], Guanyu Li[1,2,3], Lei Xu[4], Jun Bi[1,2,3(✉)], Guofei Gu[4], and Jiasong Bai[1,2,3]

[1] Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China
{zhangmh16,bjs17}@mails.tsinghua.edu.cn, dracula.guanyu.li@gmail.com, junbi@tsinghua.edu.cn
[2] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[3] Beijing National Research Center for Information Science and Technology (BNRist), Beijing, China
[4] SUCCESS LAB, Texas A&M University, College Station, USA
xray2012@email.tamu.edu, guofei@cse.tamu.edu

Abstract. Software-Defined Networking (SDN) continues to be deployed spanning from enterprise data centers to cloud computing with emerging of various SDN-enabled hardware switches. In this paper, we present Control Plane Reflection Attacks to exploit the limited processing capability of SDN-enabled hardware switches. The reflection attacks adopt direct and indirect data plane events to force the control plane to issue massive expensive control messages towards SDN switches. Moreover, we propose a two-phase probing-triggering attack strategy to make the reflection attacks much more efficient, stealthy and powerful. Experiments on a testbed with physical OpenFlow switches demonstrate that the attacks can lead to catastrophic results such as hurting establishment of new flows and even disruption of connections between SDN controller and switches. To mitigate such attacks, we propose a novel defense framework called SWGuard. In particular, SWGuard detects anomalies of downlink messages and prioritizes these messages based on a novel monitoring granularity, i.e., host-application pair (HAP). Implementations and evaluations demonstrate that SWGuard can effectively reduce the latency for legitimate hosts and applications under Control Plane Reflection Attacks with only minor overheads.

Keywords: Software-Defined Networking
Timing-based side channel attacks · Denial of service attacks

1 Introduction

Software-Defined Networking (SDN) has enabled flexible and dynamic network functionalities with a novel programming paradigm. By separating the control

M. Bailey et al. (Eds.): RAID 2018, LNCS 11050, pp. 161–183, 2018.
https://doi.org/10.1007/978-3-030-00470-5_8

plane from the data plane, control logics of different network functionalities are implemented on top of the logically centralized controller as *applications*. Typical SDN applications are implemented as event-driven programs which receive information directly or indirectly from switches and distribute the processing decisions of packets to switches accordingly. These applications enable SDN to adapt to data plane dynamics quickly and make responses to the application policies timely. A wide range of network functionalities are implemented in this way, allowing SDN-enabled switches to behave as firewall, load balancing, network address translation, L2/L3 routing and so on.

Despite the substantial benefits, the deployment of SDN has encountered several problems. In particular, a major limitation is the control message processing capability on SDN-enabled hardware switches of various brands (e.g., IBM RackSwitch, Juniper Junos MX-Series, Brocade NetIron CES 2000 Series, Pica8 Series, Hewlett-Packard Series), constrained by multiple factors. First, CPUs of hardware switches are usually relatively wimpy [8,31] for financial reasons, which restricts the message parsing and processing capability of software protocol agents in switches. Second and more importantly, flow tables in most commodity hardware OpenFlow switches use Ternary Content Addressable Memory (TCAM) to achieve wire-speed packet processing, which only allows limited flow table update rate (only supporting 100–200 flow rule updates per second [5,12,13,16,29,31,33]) and small flow table space (ranging from hundreds to a few thousand [8,16,18]) due to manufacturing cost and power consumption. These limitations have slowed down network updates and hurt network visibility, which further constrain the control plane applications with dynamic policies significantly [15].

In this paper, we systematically study the event processing logic of the SDN control plane and locate two types of data plane events which could reflect expensive control messages towards the data plane, i.e., *direct* data plane events (e.g., *Packet-In* messages) and *indirect* data plane events (e.g., *Statistics Query/Reply* messages). By manipulating those data plane events, we present two novel Control Plane Reflection Attacks in SDN, i.e., Table-miss Striking Attack and Counter Manipulation Attack, which can exploit the limited processing capability for control messages of SDN-enabled hardware switches. Moreover, in order to improve accuracy and efficiency of Control Plane Reflection Attacks, we propose a two-phase attack strategy, i.e., probing phase and triggering phase, inspired by timing-based side channel attacks. Control Plane Reflection Attacks are able to adjust attack stream patterns adaptively and cleverly, thus could gain a great increment of downlink messages[1]. Extensive experiments with a physical testbed demonstrate that the attack vectors are highly effective and the attack effects are pretty obvious.

In order to mitigate Control Plane Reflection Attacks, we present a novel and effective defense framework, namely SWGuard. SWGuard proposes a new monitoring granularity, *host-application pair (HAP)* to detect downlink message

[1] For brevity, we denote the messages from the data plane to the control plane as uplink messages, and the messages vice versa as downlink messages.

anomalies, and prioritizes downlink messages when downlink channel congests. In this way, SWGuard is able to satisfy the latency requirements of different hosts and applications under the reflection attacks.

To summarize, our main contributions in this paper include:

- We systematically study the event processing logic of SDN applications and further locate two types of data plane events, i.e., direct/indirect events, which could be manipulated to reflect expensive control messages towards SDN switches.
- We present two novel Control Plane Reflection Attacks, Table-miss Striking Attack and Counter Manipulation Attack, to exploit limited processing capability of hardware switches. Moreover, we develop a two-phase attack strategy to launch such attacks in an efficient, stealthy and powerful way. The experiments with a physical SDN testbed exhibit their harmful effects.
- We present a defense solution, called SWGuard, with an efficient priority assignment and scheduling algorithm based on the novel abstraction of host-application pair (HAP). Implementations and evaluations demonstrate that SWGuard provides effective protection for legitimate hosts and applications with only minor overheads.

The remainder of this paper is structured as follows. Section 2 introduces the background that motivates this work. Section 3 illustrates the details of Control Plane Reflection Attacks and Sect. 4 proves the harmful effects with a physical testbed. We present our SWGuard defense framework in Sect. 5 and make some discussions in Sect. 6. Related works are illustrated in Sect. 7, and the paper is concluded in Sect. 8.

2 Background

Processing Logic of Data Plane Events. SDN introduces the open network programming interface and accelerates the growth of network applications, which enable network to dynamically adjust network configurations based on certain data plane events. These events could be categorized into the following two types: ***direct*** data plane events such as *Packet-In* messages, where the event variations are reported to the controller from the data plane directly, and ***indirect*** data plane events such as *Statistics Query/Reply* messages, where the event variations are obtained through a query and reply procedure at the controller. In the *first* case, the controller installs a default *table-miss* flow rule on the switch. When a packet arrives at the switch and does not match any other flow rule, the switch will forward the packet to the control plane for further processing. Then the controller makes decisions for the packet based on the logics of the applications, and assigns new flow rules to the switch to handle subsequent packets with the same match fields. In the *second* case, the controller first installs a *counting flow rule* reactively or proactively on the switch for a measurement purpose. When a packet matches the counting flow rule in the flow table, the specific counter increments with packet number and packet bytes. To obtain the

status of the data plane, the controller polls the flow counter values for statistics periodically and performs different operations according to the analysis of statistics. A large number of control plane applications combine these two kinds of data plane events to compose complicated network functions, which further achieve advanced packet processing.

Usage Study of Data Plane Events. Based on the event-driven programming paradigm, a large number of control plane applications emerge in both academia and industry. In academia, since the publication of OpenFlow [23], many research ideas have been proposed to fully leverage the benefits of direct and indirect data plane events. While the direct data plane events are needed by almost all applications, the indirect data plane events are also widely included. In particular, we have categorized these indirect event-driven applications into three types, applications which help improve *optimization*, *monitoring* and *security* of network. Please see our technical report [36] for details. Although each of them has different purposes, all of these works are deeply involved in the utilization of the indirect data plane events, obtaining a large number of traffic features and switch attributes. Meanwhile, these indirect data plane events contribute a large part of communication between applications and switches. SDN applications have also experienced great development in industry recently. The mainstream SDN platforms (e.g. Open Daylight, ONOS, Floodlight) foster open and prosperous markets for control plane softwares, which provide a great range of applications with a composition of the direct and indirect data plane events. Meanwhile, since these applications are obtained from a great variety of sources, their quality could not be guaranteed and their logics may contain various flaws or vulnerabilities. In particular, we have investigated all mainstream SDN controllers, and discovered that indirect event-driven applications occupy a large part of application markets in these open source controller platforms. Due to the page limit, please see the application summary in our technique report [36].

Limitations of SDN-enabled Hardware Switches. Compared with the rapid growth of packet processing capability in logically centralized and physically distributed network operating systems (e.g., Onix [17], Hyperflow [30], Kandoo [11]) and controller frameworks (e.g., Open Daylight, ONOS), the downlink message processing capability of SDN-enabled hardware switches evolves much slower. State-of-the-art SDN-enabled hardware switches [24] only support 8192 flow entries. To make matters worse, the capability to update the entries in TCAM is pretty limited, usually less than 200 updates per second [5,12,13,15,16,29,31,33]. According to our experiment on Pica8 P-3922, the maximum update rate is about 150 entries per second. We observe that the downlink channel in switches is the dominant resource in SDN architecture that must be carefully managed to fully leverage the benefits of SDN applications. However, existing SDN architecture does not provide such a mechanism to protect the downlink channel in the switches that it is vulnerable to Control Plane Reflection Attacks.

3 Control Plane Reflection Attacks

In this section, we first provide our threat model and then describe the details of two Control Plane Reflection Attacks including Table-miss Striking Attack and Counter Manipulation Attack.

3.1 Threat Model

We assume an adversary could possess one or more hosts or virtual machines (e.g., via malware infection) in the SDN-based network. The adversary can utilize his/her controlled hosts or virtual machines to initiate probe packets, monitor their responses, and generate attack traffic. However, we do not assume the adversary can compromise the controller, applications or switches. In addition, we assume the connections between the controller and switches are well protected by TLS/SSL.

3.2 Control Plane Reflection Attacks

Control Plane Reflection Attacks are much more stealthy and sophisticated than previous straightforward DoS attacks against SDN infrastructure, and generally consist of two phases, i.e., *probing phase* and *triggering phase*. During the probing phase, the attacker uses *timing probing packets*, *test packets* and *data plane stream* to learn the configurations of control plane applications and their involvements in direct/indirect data plane events. With several trials, the attacker is able to determine the conditions that the control plane application adopts to issue new flow rule update messages. Upon the information obtained from probing phase, the attacker can carefully craft the patterns of attack packet stream (e.g., header space, packet interval) to deliberately trigger the control plane to issue numerous flow rule update messages in a short interval to paralyze the hardware switches. We detail two vectors of Control Plane Reflection Attacks as follows.

Table-miss Striking Attack. Table-miss Striking Attack is an enhanced attack vector from previous Data-to-control Plane Saturation Attack [9,27,28,32]. Instead of leveraging a random packet generation method to carry out the attack, Table-miss Striking Attack adopts a more accurate and cost-efficient manner by utilizing probing and triggering phases.

The probing phase is to learn confidential information of the SDN control plane to guide the patterns of attack packet stream. The attacker could first probe the usage of the direct data plane events (e.g., *Packet-In*, *Packet-Out*, *Flow-Mod*) by using various low-rate probing packets whose packet headers are filled with deliberately faked values. The attacker can send these probing packets to the SDN-based network and observe the responses accordingly, thus the round trip time (RTT) for each probing packet could be obtained. If several packets with the same packet header get different RTT values, especially, the first packet goes through a long delay while the other packets get relatively quick responses,

we can conclude that the first packet is directed to the controller and the other packets are forwarded directly in the data plane, which indicates that the specific packet header matches no flow rules in the switch and invokes *Packet-In* and *Flow-Mod* messages. Then the attacker could change one of the header fields with the variable-controlling approach. With no more than 42 trials[2], the attacker is able to determine which header fields are sensitive to the controller, i.e., the grain for routing. Then the attacker could carefully craft attack packet stream based on probed grains to deliberately trigger the expensive downlink messages.

Counter Manipulation Attack. Compared with Table-miss Striking Attack, Counter Manipulation Attack is much more sophisticated, which is based on the indirect data plane events (e.g., *Statistics Query/Reply* messages). In order to accurately infer the usage of the indirect data plane events, three types of packet streams are required, i.e., *timing probing packets*, *test packets* and *data plane stream*. The *timing probing packets* are inspired by the *time pings* in [29], which must involve the switch software agent and get the responses accordingly. However, we believe that they have a wider range of choices. The *test packets* are a sequence of packets which should put extra loads to the software agent of the switch, and must be issued at an appropriate rate for the accuracy of probing. The *data plane stream* is a series of stream templates, and should directly go through the data plane (i.e., do not trigger table-miss entry in the flow table of the switch), which is intended to obtain more advanced information such as the specific conditions which trigger indirect event-driven applications.

Timing probing packets are used to measure the workload of software agent of a switch, which should satisfy three properties: first, they should go to the control plane by hitting the table-miss flow rule in the switch, and trigger the operations of the corresponding applications (e.g., *Flow-Mod* or *Packet-Out*). Second, each of them must evoke a response from the SDN-based network, so the attacker could compute the RTT for each timing probing. Third, they should be sent in an extremely *low* rate (10 pps is enough), and put as low loads as possible to the switch software agent. We consider there are many options for timing probing packets, e.g., ARP request/reply, ICMP request/reply, TCP SYN or UDP. For layer 2, we consider ARP request is an ideal choice since the SDN control plane must be involved in the processing of ARP request/reply. We note that sometimes the broadcast ARP request will be processed in the switches. However, the corresponding ARP reply is a unicast packet so that the control plane involvement is inevitable if the destination MAC (i.e., the source MAC address of the ARP Request) has not been dealt by the controller before. As a result, the attacker could use spoofed source MAC address to deliberately pollute the device management service of the controller as well as incur the involvement of the controller. In some layer 2 network, it is not possible to send packets with random source MAC addresses due to pre-authorized network access control policies. To address this, the attacker could resort to the *flow rule time-out* mechanism of OpenFlow. The attacker would select N benign

[2] The latest OpenFlow specification only support 42 header fields, which constrains the field the controller could use to compose different forwarding policies.

hosts and send ARP request to them to get the responses. N should satisfy that $N > R * T$, where R denotes the probing rate and T denotes the *flow-rule time-out* value[3]. For Layer 3, ICMP is a straightforward choice, since its RTT calculation has been abstracted as *ping* command already. The attacker should choose a number of benign hosts to send ICMP packets and get the corresponding responses. As for layer 4, TCP and UDP are both feasible when a layer 4 forwarding policy is configured in the control plane. According to RFC 792 [26], when a source host transmits a probing packet to a port which is likely closed at the destination host, the destination is supposed to reply an ICMP *port unreachable* message to the source. Similarly, RFC 793 [25] requires that each TCP SYN packet should be responded with a TCP SYN/ACK packet (opened port) or TCP RST packet (closed port) accordingly. With the probing packet returned with the corresponding response, the RTT could be calculated and the time-based patterns could be obtained.

Test packets are used to strengthen the effects of *timing probing packets* by adding extra loads to the software agent of the switch. For the purpose, we consider test packets with a random destination IP address and broadcast destination MAC address are ideal choices. By hitting the table-miss entry, each of them would be directed to the controller. Then the SDN controller will issue *Packet-Out* message to directly forward the test packet. As a result, the aim of burdening switch software agent is achieved.

Data plane stream is a series of templates, which should go directly through the data plane to obtain more advanced information such as the specific conditions for indirect event-driven applications. We provide two templates here, as shown in Fig. 1. The first template has a steady rate v, packet size p, which is mainly used to probe volume-based statistic calculation and control method. The second has a rate distribution like a jump function, where three variables (v, t, p) determine the shape of this template as well as the size of each packet, which is often used to probe the rate-based strategy.

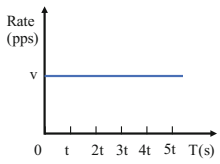
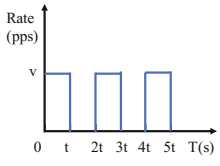
Template Name	Coordinate Axis	Variables
Data plane stream with steady rate		(v, p)
Data plane stream with 0-1 rate		(v, t, p)

Fig. 1. Templates for data plane stream.

The insight of probing phase of Counter Manipulation Attack lies in that *different kinds of downlink messages have diverse expenses for the downlink channel*. Among the interaction approaches between the applications and the data plane, there are mainly three types of downlink messages, i.e., *Flow-Mod*, *Statistics Query* and *Packet-Out*. *Flow-Mod* is the most expensive one among

[3] As R is less than 10 usually, and T is set as a small value in most controllers (e.g. 5 in Floodlight), thus N cannot be a large number.

them, since it not only consumes the CPU of switch agent to parse the message, but also involves the ASIC API to insert the new flow rules[4]. *Statistics Query* comes at the second, for it needs the involvement of both switch agent CPU for packet parsing and ASIC API for statistic querying. These two types of messages are extremely expensive when the occupation of flow table is high on the switch. *Packet-Out* is rather lightweight, since it only involves the CPU of switch protocol agent to perform the corresponding action encapsulated in the packet. As these three types of downlink messages incur different loads for the switch, the latencies of *timing probing packets* will vary when the switch encounters different message types. Thus, the attacker could learn whether the control plane issue a *Flow-Mod*, or a *Statistics Query*, or a *Packet-Out*. As for the indirect data plane events, the statistic queries are usually conducted periodically by the applications. As a result, each of these queries would incur a small rise for the RTTs of *timing probing packets*, which would reveal the period of application's statistic query. If a subsequent *Flow-Mod* is issued by the controller, there would be a higher rise of RTT just following the RTT for *Statistics Query*, which is named as *double-peak phenomenon*. Based on the special phenomenon, the attacker could even infer what statistic calculation methods the application takes, such as volume-based or rate-based. With several trails of two *data plane stream* templates above (t is set as the period of statistic messages, which has been obtained above) and the variations of v and p in a binary search approach, the attacker could quickly obtain the concrete conditions (volume/rate values, number-based or byte-based) that trigger the expensive downlink messages. The confidential information such as statistic query period, the exact conditions (volume/rate values, packet number-based or byte-based) that trigger the downlink messages, helps the attacker permute the packet interval and packet size of each flow, to deliberately manipulate the counter value to the critical value, thus each flow would trigger a *Flow-Mod* in every period. By initiating a large number of flows, *Flow-Mod* of equal number would be triggered every period, making the hardware switch suffer extremely.

4 Attack Evaluation

In this section, we demonstrate our experimental results of Control Plane Reflection Attacks with a physical testbed. The evaluations are divided into two parts. First, we conduct our experiments for Table-miss Striking Attack and Counter Manipulation Attack separately, to show the effectiveness of Control Plane Reflection Attacks. Second, we perform some benchmarks to provide low-level details of our proposed attacks.

4.1 Experiment Setup

To demonstrate the feasibility of Control Plane Reflection Attacks, we set up an experimental scenario as shown in Fig. 2. We choose several representative

[4] Moving old flow entry to make room for the new flow rule is an important reason to make this operation expensive and time-consuming.

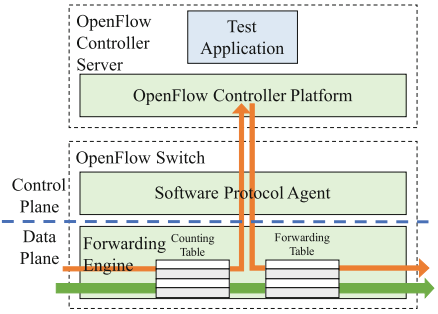


Fig. 2. A typical attack scenario.

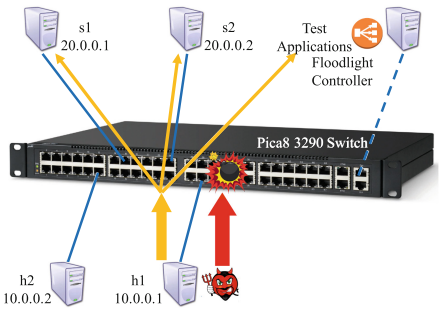


Fig. 3. Attack experiment setups.

applications, and run them separately on the SDN controller. Flow tables in the switch are divided into two pipelines, *Counting Table* for the indirect data plane event, *Forwarding Table* for the direct data plane events. Each pipeline contains multiple flow entries for the two data plane events, and flow tables of each pipeline are independent and separated, which is the state-of-the-art approach for multiple application implementations today [20,29].

Reactive Routing is the most common application integrated into most of the popular controller platforms. It monitors *Packet-In* messages with a default *table-miss* in *Forwarding Table*, and computes and installs a path for the hosts of the given source and destination addresses with an appropriate grain. When one table-miss occurs, $2N$ downlink *Flow-Mod* messages would be issued to the data plane, where N is the length of the routing path.

Flow Monitoring is another common application in SDN-based networks. It is generally implemented with a *Counting Table* which counts the number and the bytes of a flow or multiple flows. The controller polls the statistics of the *Counting Table* periodically, conducts analysis on the collected data, and makes decisions with the analysis results. Further, we extend our *Flow Monitoring* sketch into four indirect data plane events driven applications, **Heavy hitter** [22], **Microburst** [10], **PIAS** [1] and **DDoS Detection** [34]. The implementation details are illustrated in our technical report [36].

Our evaluations are conducted on a physical OpenFlow Switch, i.e., Pica8 P-3290, since it is widely used in academia/industry and supports many advanced OpenFlow data plane features, such as multiple pipelines and almost full OpenFlow specifications (from version 1.0 to 1.4). The experimental topology, as shown in Fig. 3, includes four machines (i.e., h1, h2, s1, and s2) connected to the hardware switch and a server running Floodlight Controller. The HTTP service is run on s1 and s2 separately. We consider h2 is a benign client of the HTTP service and h1 is controlled by the attacker to launch the reflection attack. All the tested applications discussed above are hosted in the Floodlight controller. In our experiments, *Reactive Routing* adopts a five-tuples grained forwarding policy, and four *Flow Monitoring*-based applications query the data plane switch every

2 s, and conduct the corresponding control (e.g., issue one *Flow-Mod* message) according to their logic separately.

4.2 Attack Feasibility and Effects

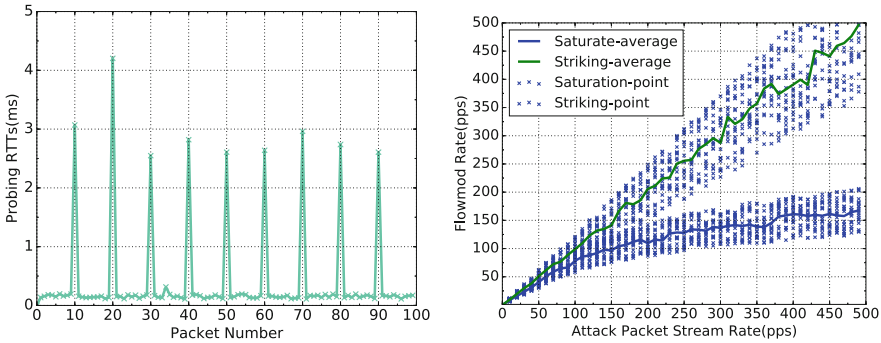
In this subsection, we conduct the experiments for Table-miss Striking Attack and Counter Manipulation Attack separately, and show a detailed procedure for *probing phase* and *triggering phase*.

Table-miss Striking Attack. For the *Reactive Routing* application, when we launch a new flow, the first packet is inclined to get a high RTT, and the following several packets would get low RTTs. Since there are only three hosts on our testbed and *ping* could launch only one new flow between each host pair, we resort to UDP probing packets to cope with this problem. We compute the time difference between the request and reply to obtain the RTT. As depicted in Fig. 4(a), we let h1 transmit 10 UDP probing packets to a destination port and then change the destination port. The RTT for the first packet of each flow is quite distinct from that of the other packets. When we change any field pertained to five-tuples, the similar results would be obtained. The modification to other packet fields would always lead to a quick response. All the phenomena indicate that five-tuples grained forwarding policy is adopted by the *Reactive Routing*.

With the inference of forwarding grain, the attacker is able to carefully craft a stream of packets whose header spaces vary according to the grain. In this way, each attack packet could strike the default *table-miss* in the switch, thus triggering *Packet-In* and *Flow-Mod* in the control channel. Data-to-control Plane Saturation Attack resorts to a random packet generation approach, making the attack not so cost-efficient for the attacker. As we can see in Fig. 4(b), Table-miss Striking Attack is much more efficient than Data-to-control Plane Saturation Attack. Further, we also compare the RTTs and bandwidth for normal users under the saturation attack and the striking attack. As shown in Fig. 5, the striking attack could easily obtain a higher RTT and a lower bandwidth usage for normal users with the same attack expense, which demonstrates that our Table-miss Striking Attack is much more cost-efficient and powerful.

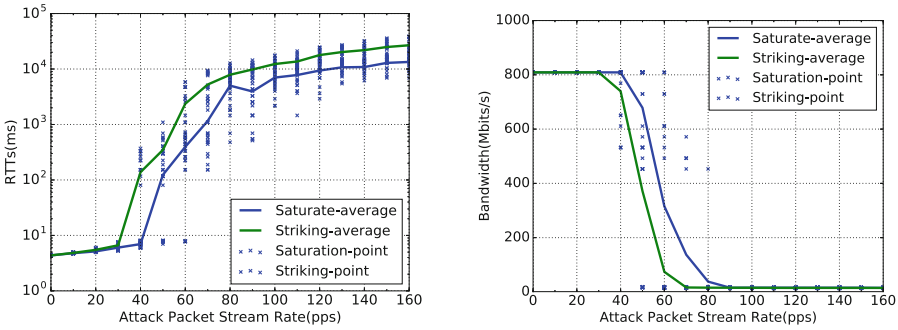
Counter Manipulation Attack. For the *Flow Monitoring*-based applications, we first supply a steady rate of *test packets* at 300 packets per second (pps)[5], which would put appropriate loads on the control plane as required in [29]. The rate of *timing probing packets* is set as 10 pps. The results for four applications are similar, as shown in Fig. 6(a). As we could conclude, *Flow Monitoring*-based applications poll the switch for statistics every 2 s. In particular, the double peaks in red rectangle (*double-peak phenomenon*) denote two expensive downlink messages are issued successively. The first peak is attributed to the periodical *Statistics Query* message, while the second is caused by the *Flow-Mod* message for the control purpose. We make this inference because both *Flow-Mod* and

[5] 300 pps is a pretty secure rate, since a legitimate host could issue packets at thousand of pps under normal circumstance.



(a) RTTs for Reactive Routing. (b) Reactive Routing attack efficiency.

Fig. 4. Attack feasibility and efficiency for Table-miss Striking Attack.



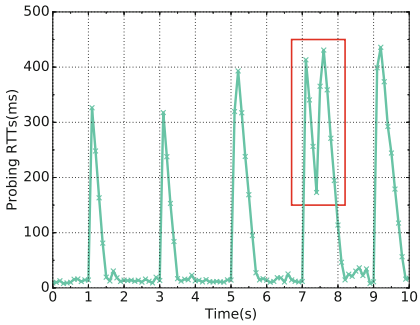
(a) RTTs for normal users under the saturation attack and the striking attack.

(b) Bandwidth for normal users under the saturation attack and the striking attack.

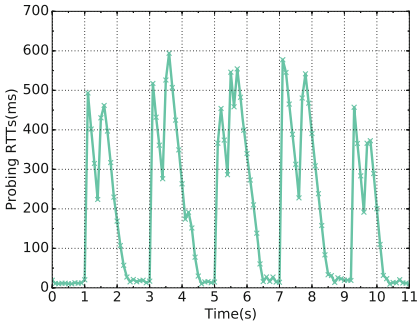
Fig. 5. RTTs and bandwidth for normal users under the saturation attack and the striking attack.

Statistics Query are much more expensive than *Packet-Out* while they two have a similar expense for the downlink channel.

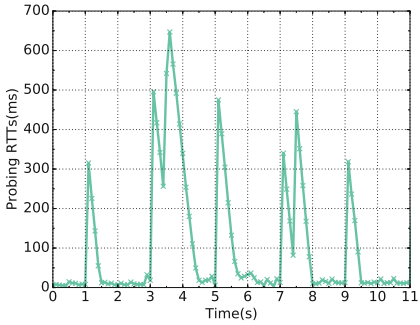
Furthermore, more confidential information could be obtained with the joint trials and analysis of *data plane stream* and *double-peak phenomenon*. If the attacker obtains a series of successive double-peak phenomenon (as shown in Fig. 6(b) with the input of data plane stream template1, where v is a big value, and obtains a series of intermittent double-peak phenomenon (Fig. 6(c) with template2, where v is also a big value, she/he could determine that packet number volume-based statistic calculation method is adopted. This is because packet number volume-based statistic calculation approach is sensitive to stream with a high pps. The other three cases are also listed in Table 1. From this table, we can conclude the concrete statistic calculation approach the application adopts. Furthermore, with the variations of v and p, the attacker could infer the critical value of volume or rate. In addition, we can verify our inference with a lot of



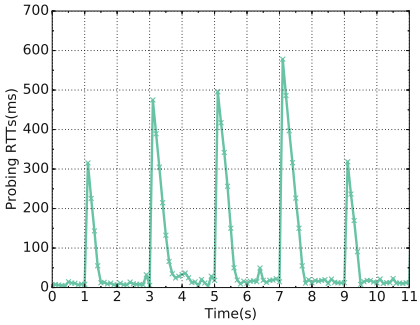
(a) Timing probe RTTs for *Flow Monitoring*-based applications.



(b) Timing probe RTTs patterns 1.



(c) Timing probe RTTs patterns 2.



(d) Timing probe RTTs pattern 3.

Fig. 6. Timing-based patterns for Counter Manipulation Attack.

other ways, not only the proposed two data plane stream templates as shown above. We are planning to develop more representative templates in our future works. In particular, we test our four indirect event driven applications, and find them fall into the distribution in Table 2. This is consistent with the policies of each application, which demonstrates the effectiveness of our probing phase.

With the results and information (query period, packet number/byte-based, volume/rate values) obtained from the probing phase, we move to the second step and start to commit our Counter Manipulation Attack. We select one application, PIAS, setting its priority as 3 levels, and initiate 10 new flows per second. We carefully set the sent bytes of each flow in each period (2 s), which is bigger than the critical value we probed. As a consequence, a number of *Flow-Mod* messages are issued to the switch when statistic query/reply occurs. As shown in Fig. 7, the number of *Flow-Mod* messages could increase as high as 60 at the end of each period. This would incur pretty high loads to the software agent of the switch at this moment. Even in some cases, when the attacker controls thousands of flows intentionally and manipulates all the flow to reach the critical values simultaneously, thousands of *Flow-Mod* messages are directed to the switch, which would cause catastrophic results such as the disruption of connections between the controller and the switches.

Table 1. Relationship between data plane stream and double-peak phenomenon.

	Volume-based	Rate-based
Packet number	Template1(v↑, p) → patterns 1	Template1(v↑, p) → patterns 3
	Template2(v↑, p) → patterns 2	Template2(v↑, p) → patterns 1
Packet byte	Template1(v, p↑) → patterns 1	Template1(v, p↑) → patterns 3
	Template2(v, p↑) → patterns 2	Template2(v, p↑) → patterns 1

Table 2. Distribution of the four indirect event driven applications.

	Volume-based	Rate-based
Packet number	Microburst	-
Packet byte	Heavy Hitter PIAS DDoS Detection	DDoS Detection

4.3 Attack Fundamentals and Analysis

In this subsection, we study more about low-level details of Control Plane Reflection Attacks.

***Test Packet* Rate and *Test Packet* Type.** Fig. 8 shows the timing probe RTTs as the rate of test packets varies where the controller is configured to issue a *Flow-Mod* message for each test packet. Figure 9 shows the timing probe RTTs as *Statistics Query* rate varies. Figure 10 shows the timing probe RTTs as the rate of test packets varies where the controller processes each test packet with a *Packet-Out* message. As we can conclude from these figures, different downlink messages have diverse expenses for the downlink channel, and all of the three scenarios encounter a significant *nonlinear jump*. In particular, when the controller generates *Flow-Mod* message for each test packet, the RTTs can reach 1000 times higher at approximately 50 pps. For *Statistics Query* messages, the RTTs are about 100 times at 100 pps. And for *Packet-Out* messages, the RTTs double 100 times at about 500 pps. Meanwhile, we measure the resource usage of the hardware switch and the controller, and find that the CPU usage of the switch could reach above 90% at the point of the nonlinear jump, while the memory usage of the switch, the CPU and memory usage of the control server is relatively low (at most 30%). In addition, we have a conservation with the Pica8 team via email, and obtain that the switch control actions (e.g. *Flow-Mod*, *Statistics Query*) must contend for the limited bus bandwidth between a switch's CPU and ASIC, and insertion of a new flow rule requires the rearrangement of rules in TCAM, which lead to the results that the expense of *Flow-Mod* $\geq$ *Statistics Query* $\gg$ *Packet-Out*.

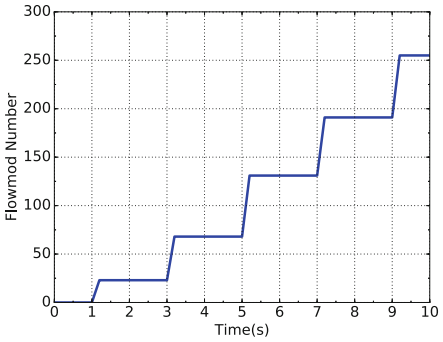


Fig. 7. Attack effect

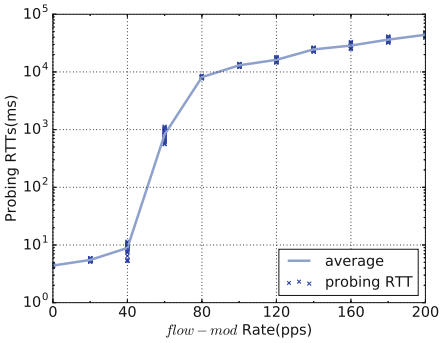


Fig. 8. Timing probe RTTs as Flow-Mod rate varies.

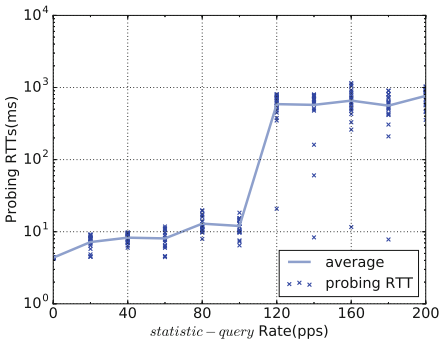


Fig. 9. Timing probe RTTs as statistic query rate varies.

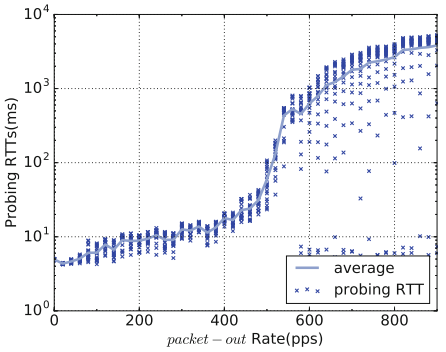


Fig. 10. Timing probe RTTs as Packet-Out rate varies.

The Impact of Background Traffic. The background traffic has two impacts for the Control Plane Reflection Attacks. First, it may affect the accuracy of probing phase. In fact, a moderate rate of background traffic would not weaken the effectiveness of the probing. Conversely, it amplifies the probing effect. The reason behind this is that the effect of background traffic is somewhat like the role played by *test packets*, and it would put some baseline loads to the switch protocol agent, which would make the probing more accurate. An excessively high rate of background traffic would certainly lower the probing accuracy, since there is already a high load for the protocol agent of the switch. As a consequence, the loads incurred by *Statistics Query* would not cause the obvious and periodical peaks for the RTTs of *timing probing packets*, instead, the patterns may become random and irregular. However, in such cases, the switch is already suffering, thus the aim of the attack has already been achieved. Second, the background traffic may also affect the trigger phase. Actually, this influence is positive, too. The existence of the background traffic would inevitably bring about some downlink

messages to the control channel, which would boost the effects of Control Plane Reflection Attacks.

5 Defense Approach

5.1 Countermeasure Analysis

The control plane reflection attack is deeply rooted in SDN architecture since the performance of existing commodity SDN-enabled hardware switches could not suffice the need of the SDN applications. A straightforward method to mitigate this attack is *limiting the use of dynamic features for network applications*, nevertheless, this comes at the expense of less fine-grained control, visibility, and flexibility in traffic management, as evidently required in [4,14,31]. Another straightforward defense approach is *limiting the downlink message transmission rate directly in the controller*, preventing the switches from being overwhelmed. However, the exact downlink message processing capabilities for different switches vary, even for a specific switch, the rate control in the controller cannot precisely guarantee underload or overload for the remote switch[6], making the unified control inaccurate and complicated. *Adding some latency to random downlink messages* seems feasible, which can make the patterns/policies of direct/indirect data plane events difficult to sniff and obtain. Nevertheless, this technique increases the total latency for the overall downlink messages, and would inevitably violate the latency requirements of some latency-sensitive downlink messages, making it high cost and infeasible.

To address the challenges above, we propose SWGuard to mitigate the reflection attack and fulfill the requirements of different downlink messages. Our basic idea is to discriminate good from evil, and prioritize downlink messages with discrimination results. To this end, we propose a multi-queue scheduling strategy, to achieve different latency for different downlink messages. The scheduling strategy is based on the statistics of downlink messages in a novel granularity during the past period, which takes both fairness and efficiency into consideration. When the downlink channel is becoming congested, the *malicious* downlink messages are inclined to be put into a low-priority scheduling queue and the requirements of *good* messages are more likely to be satisfied.

5.2 SWGuard: A Priority-Based Scheduler on Switch

The architecture of SWGuard is shown in Fig. 11. SWGuard mainly redesigns two components of SDN architecture. On the switch side, it changes the existing software protocol agent to *multi-queue* based structure, and schedules different downlink messages with their *types* and *priorities*. On the controller side, it adds a *Behavior Monitor* module as a basic service, which collects the downlink message events and assigns different *priorities* to different messages *dynamically*.

[6] There may be several hops between the switch and the controller, and the network condition is unpredictable.

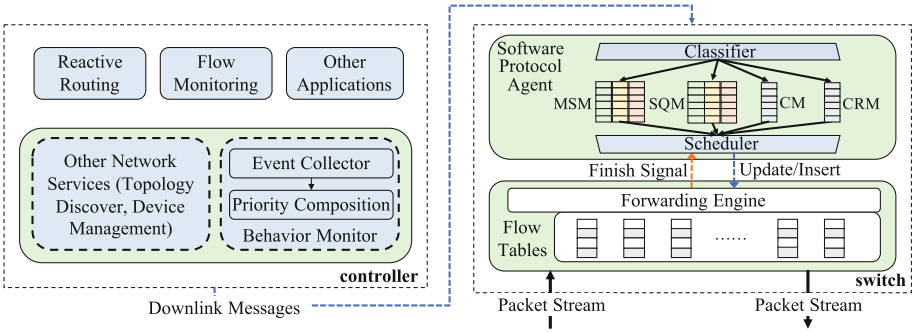


Fig. 11. SWGuard framework design.

Multi-queue Based Software Protocol Agent. In order to prioritize the downlink messages, we redesign the software protocol agents of the existing switches. A naive approach is to modify the existing single queue model directly into priority-based multi-queue model, and enqueue all the downlink messages into different queues with their priorities and dequeue at different scheduling rates. However, the *types* of downlink messages vary, and different message types have diverse requirements, for example, if *Handshake* messages and *Modify State* messages are put into the same queue, the latency requirement of the former may be delayed by the latter so that the handshakes between the controller and the switches could not be established timely.

To this end, we summarize the downlink messages into the following four categories: (1) *Modify State Messages* (MSM), (2) *Statistic Query Messages* (SQM), (3) *Configuration Messages* (CM), and (4) *Consistency Required Messages* (CRM), and design a *Classifier* to classify the downlink messages into different queues accordingly. The first two types are related to the behaviors of *hosts* and *applications*, so we design a multi-queue for each of them. The multi-queue consists of three levels (*quick*, *slow*, *block*), and each level is designed for the corresponding priority. The third type serves for basic services of the controller (e.g., Handshake, LLDP), while the detail of the last type is illustrated in Sect. 5.2, and both of them inherit from the original single queue. Classifier makes use of *ofp_header* field in OpenFlow Header to distinguish message type, and a 2-bit packet metadata to obtain priority.

With the downlink messages in the queues, a *Scheduler* is designed to dequeue the messages with a scheduling algorithm. In order not to overwhelm the capability of ASCI/Forwarding Engine, a *Finish Signal* should be sent back to the *Scheduler* once a *Modify State/Statistic Query* message is processed. Then the *Scheduler* knows whether to dequeue a next message of the same type from queues. We design a time-based scheduling algorithm, setting different *strides* for different queues. For the last two queues (Configuration Messages, Consistence Required Messages), the stride is set as 0, which means whenever there is a message, it would be dequeued immediately. For the first two multi-queues,

the stride for the queue of *quick* level is set as 0, for that of *slow* level is set as a small time interval, while for that of *block* level is set as a relatively bigger value. With the principles illustrated above, we design the scheduling algorithm as Algorithm 1.

Algorithm 1. The Scheduling Algorithm for Protocol Agent.

```
// Initialization
foreach que ∈ queues do
    set que.stride;
    que.time = getcurrenttime();
// Enter the Scheduler thread
while true do
    foreach que ∈ queues do
        if que.stride ≤ getcurrenttime() − que.time then
            if que.empty() == false then
                que.time = getcurrenttime();
                que.dequeue();
            else
                que.time = getcurrenttime();
```

Behavior Monitor. In order to distinguish different downlink messages with different priorities, an appropriate *Monitoring* granularity is in urgent need. Previous approaches mainly conduct the monitoring with the granularity of source host [3,34], and react to the anomalies on the statistics. However, in the control plane reflection attacks, these approaches are no longer valid and effective. For example, if we only take the features of the data plane traffic into consideration, and schedule with the statistics of source hosts [35], it would inevitably violate the heterogeneous requirements of various applications.

To address this challenge, we propose the novel abstraction of *Host-Application Pair (HAP)*, and use it as the basic granularity for monitoring and statistics. These two dimensions are easy to be obtained from the uplink messages and the configurations of the controller. Considering K applications exist on the control plane, their requirements for downlink messages are represented as vector $\boldsymbol{a_0} = \langle a_1, a_2, \ldots a_K \rangle$, and N hosts/users in the data plane, corresponding requirements vector $\boldsymbol{h_0} = \langle h_1, h_2, \ldots h_N \rangle$. $\boldsymbol{a_0}$ and $\boldsymbol{h_0}$ are both set by the network operators, depending on the property of the applications and the pay of hosts/users. Thus the *expected resource allocation matrix* is $\mathbf{R_0} = {\boldsymbol{a_0}}^T \cdot \boldsymbol{h_0}$. And the *expected resource allocation ratio matrix* is $\mathbf{I_0} = \frac{\mathbf{R_0}}{\sum_{k=1}^{K} \sum_{n=1}^{N} a_k h_n}$. During the past period (T seconds), the statistics of *HAP* is represented as *resource*

occupation matrix $\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & \dots & r_{1N} \\ r_{21} & r_{22} & \dots & r_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ r_{K1} & r_{K2} & \dots & r_{KN} \end{pmatrix}$. And the sum of the elements in $\mathbf{R}$ is denoted as $Sum = \sum_{k=1}^{K} \sum_{n=1}^{N} r_{kn}$. Suppose the maximum capability of downlink channel in T seconds is Sum_0, $\frac{Sum}{Sum_0}$ denotes the resource utilization rate of the downlink channel. In order to save resources of the control channel, we design our SWGuard system as attack-driven, which means when $\frac{Sum}{Sum_0} < \alpha$, SWGuard is in sleep state except for *Event Collector*. All the downlink messages flow through the third queue (queue for *Configuration Messages*). α is a danger value between 0 and 1, set by the network operators.

When the reflection attacks are detected, the *Priority Composition* Module is wakened and starts to calculate the *penalty coefficient* of each HAP, $\beta_{kn} = \frac{r_{kn} - i_{kn} Sum_0}{r_{kn}}$. i_{kn}, r_{kn} denote the corresponding element in matrix $\mathbf{I_0}, \mathbf{R}$. If β_{kn} is negative, we set it as 0. Then we use two *thresholds* (th_h, th_l) to map the penalty coefficient β_{kn} into *priority* (00, 01 or 10) and tag a 2-bit field into packet metadata to encapsulate priority.

Policy Consistency. Multi-queue based software protocol agent may violate the consistency of some downlink messages. For example, some control messages need to be sent in a particular order for correctness reasons, however, in this multi-queue based software agent, if a previous arriving message is put into a queue with high load while a later arriving message is put into a queue with low load, the order to maintain correctness may be violated.

To address this issue, we design a coordination mechanism between the *Behavior Monitor* and *Classifier* in software protocol agent. If a series of downlink messages require consistency, they are supposed to reuse the 2-bit priority packet metadata (fill it with 11) in the packet header to express their intents. Then the *Classifier* in the software protocol agent will check the label to learn whether the message has the consistency demand. If consistency demand is confirmed, this message will be scheduled to the queue for consistency required messages.

5.3 Defense Evaluation

We implement the prototype of SWGuard system, including *Behavior Monitor* and *Software Protocol Agent*, on Floodlight [6] and Open vSwitch [7] with about 4000 Lines of Code. We use Open vSwitch and set corresponding thresholds to limit its control channel throughput, making its flow rule update rate (130 pps) and flow table size (2000) analogous to the hardware switches.

To demonstrate the defense effect of SWGuard, we use the average value of flow rule installation/statistic query latencies of normal users/applications as the representative metric, which is named as *Event Response Time* in our figures. As

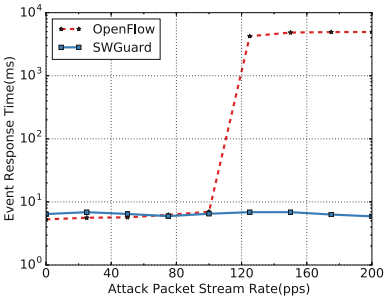


Fig. 12. Defense effect.

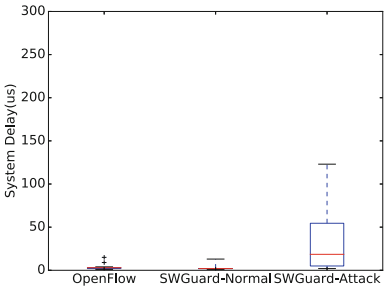


Fig. 13. Defense overhead.

shown in Fig. 12[7], with native system, event response time becomes extremely large when the rate of downlink messages is above 110 packets per second. While with SWGuard, event response time is nearly unchanged. All of these are due to the limited capability of SDN-enabled switches for processing downlink messages. The experimental results illustrate that our SWGuard provides effective protection for both the flow rule installation and statistics query.

For the overheads of SWGuard, we measure the latency introduced by SWGuard. Compared with native OpenFlow, packets in SWGuard need to go through two extra components, Event Collector of Behavior Monitor and Configuration Message queue of Software Protocol Agent under normal circumstance, since other components are in sleep state when no attack is detected. When an attack happens, packets must pass a full path in Behavior Monitor and Software Protocol Agent. As shown in Fig. 13, the latency is almost the same for native OpenFlow and SWGuard under normal circumstance. Even under attacks, Behavior Monitor and Software Protocol Agent only incur a latency less than 100 us. All of these demonstrate that SWGuard only brings about a negligible delay for the control channel messages.

6 Discussion

Emerging Programmable Data Planes: Current prototypes, attacks and defenses are based on OpenFlow-based hardware switches. We believe the core idea of Control Plane Reflection Attacks is applicable to the emerging generation of programmable data planes, e.g. P4 and RMT chips [2], because these platforms also use traditional TCAM-based flow tables and Control Plane Reflection Attacks address a property of TCAM that is invariant to underlying TCAM design.

Generality of the SWGuard System: SWGuard is also applicable for no-adversary circumstances, such as flash crowds of downlink messages under normal conditions. By prioritizing the downlink messages, SWGuard can provide

[7] Since this experiment is conducted on the software environment, the nonlinear jump point is a little different from the previous hardware experimental results.

lower latencies for more important messages under the congestion status of control channel.

Source Address Forgery Problem: One concern is that an attacker may forge another host's source address to pollute the HAP statistics of other hosts. Nevertheless, in SWGuard, we can also harness the edge switch port to identify a host. As the header fields of the upstream messages are assigned by the hardware switch, the attacker is not able to forge or change this field.

7 Related Work

DoS Attacks Against SDN: Shin et al. [28] first proposes the concept of Data-to-control Plane Saturation Attack against SDN. To mitigate this dedicated DoS attack, AVANT-GUARD [27] introduces *connection migration* and *actuating triggers* to extend the data plane functions. However, it is applicable to TCP protocol only. Further, a protocol-independent defense framework, FloodGuard [32], pre-installs proactive flow rules to reduce table-miss packets, and forwards table-miss packets to additional data plane caches. To gain the benefit of no hardware modification and addition, FloodDefender [9] offloads table-miss packets to neighbor switches and filters out attack traffic with two-phase filtering. Control Plane Reflection Attacks distinguish themselves from previous works in both attack methods and attack effects. On one hand, the saturation attack uses a pretty straightforward attack method that attacker just floods arbitrary attack traffic to trigger the direct data plane events while the reflection attacks resort to more advanced and sophisticated techniques, and a two-phase probing-trigger approach is specially developed to exploit both direct and indirect data plane events. On the other hand, since the simplicity of the saturation attack, it is not hard to capture the attack, thus it could have limited attack effects. By contrary, the reflection attacks are much more stealthy and the same attack expenses of the attacker could cause more obvious attack effects for victims. Scotch [31] alleviates the communication bottleneck between control plane and data plane leveraging a pool of vSwitches distributed across the network, and it shares the same observation that SDN-enabled hardware switches have a very limited capacity for control channel communications.

Timing-Based Side Channel Attacks: Side channel attacks have long existed in computer systems, and they are usually used to leak the secret information (e.g. secret cryptographic keys) of dedicated systems. Publications more related to our work are various works applying side channel attacks to SDN. Shin et al. [28] presents an SDN scanner which could determine whether a network is using SDN or not. Leng et al. [19] proposes to measure the response time of requests to obtain the approximate capacity of switch's flow table. Sonchack et al. [29] demonstrates an inference attack to time the control plane, which could be used to infer host communication patterns, ACL entries and network monitoring policies. Liu et al. [21] permits the attacker to select the best probes with a Markov model to infer the recent occurrence of a target flow. Our attack

methods are somewhat inspired by these previous works. However, all of them only focus on the direct data plane events, and remain at a low level to infer the existence of network policies/device configurations. To the best of our knowledge, our work proposes the exploitation of indirect data plane events for the first time and take the next step that we not only take the existence into consideration, but also obtain more concrete policies and policy thresholds to promote the attack effects.

8 Conclusion

In this paper, we present Control Plane Reflection Attacks to exploit the limited processing capability of SDN-enabled hardware switches by using direct and indirect data plane events. Moreover, we develop a two-phase attack strategy to make such attacks efficient, stealthy and powerful. The experiments showcase the reflection attacks can cause extremely harmful effects with acceptable attack expenses. To mitigate reflection attacks, we propose a novel defense solution, called SWGuard, by detecting anomalies of control messages and prioritizing them based on the host-application pair. The evaluation results of SWGuard demonstrate its effectiveness under reflection attacks with minor overheads.

Acknowledgement. This material is based upon work supported by National Key R&D Program of China (2017YFB0801701), the National Science Foundation of China (No.61472213) and CERNET Innovation Project (NGII20160123). It is also based upon work supported in part by the National Science Foundation (NSF) under Grant No. 1617985, 1642129, 1700544, and 1740791. Jun Bi is the corresponding author. We also thank Yi Qiao, Chen Sun, Yongbin Li and Kai Gao from Tsinghua University for joining the discussion of this paper.

References

1. Bai, W., et al.: Information-agnostic flow scheduling for commodity data centers. In: NSDI, pp. 455–468. USENIX Association, Oakland (2015). https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/bai
2. Bosshart, P., et al.: P4: programming protocol-independent packet processors. SIGCOMM CCR **44**(3), 87–95 (2014)
3. Braga, R., et al.: Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: LCN, pp. 408–415. IEEE (2010)
4. Casado, M., et al.: Ethane: taking control of the enterprise. In: SIGCOMM, vol. 37, pp. 1–12. ACM (2007)
5. Chen, H., Benson, T.: The case for making tight control plane latency guarantees in SDN switches. In: SOSR, pp. 150–156. ACM (2017)
6. Floodlight Community: Floodlight, August 2017. http://www.projectfloodlight.org/floodlight/
7. Open vSwitch Community: Open vSwitch, August 2017. http://openvswitch.org/
8. Curtis, A.R.: DevoFlow: scaling flow management for high-performance networks. SIGCOMM **41**(4), 254–265 (2011)

9. Gao, S., et al.: FloodDefender: protecting data and control plane resources under SDN-aimed DoS attacks. In: INFOCOM, pp. 1–9 (2017)
10. Ghorbani, S., et al.: DRILL: micro load balancing for low-latency data center networks. In: SOGCOMM, pp. 225–238. ACM (2017)
11. Hassas Yeganeh, S., Ganjali, Y.: Kandoo: a framework for efficient and scalable offloading of control applications. In: HotSDN, pp. 19–24. ACM (2012)
12. He, K., et al.: Mazu: taming latency in software defined networks. Technical report, University of Wisconsin-Madison (2014)
13. He, K., et al.: Measuring control plane latency in SDN-enabled switches. In: SOSR, p. 25. ACM (2015)
14. Jin, X., et al.: SoftCell: scalable and flexible cellular core network architecture. In: CoNEXT, pp. 163–174. ACM (2013)
15. Jin, X., et al.: Dynamic scheduling of network updates. In: SIGCOMM, vol. 44, pp. 539–550. ACM (2014)
16. Katta, N., et al.: CacheFlow: dependency-aware rule-caching for software-defined networks. In: SOSR, p. 6. ACM (2016)
17. Koponen, T., et al.: Onix: a distributed control platform for large-scale production networks. In: OSDI, vol. 10, pp. 1–6 (2010)
18. Lazaris, A., et al.: Tango: simplifying SDN control with automatic switch property inference, abstraction, and optimization. In: CoNEXT, pp. 199–212. ACM (2014)
19. Leng, J., et al.: An inference attack model for flow table capacity and usage: exploiting the vulnerability of flow table overflow in software-defined network. arXiv preprint arXiv:1504.03095 (2015)
20. Li, Y., et al.: Flowinsight: decoupling visibility from operability in SDN data plane. SIGCOMM Demo **44**(4), 137–138 (2015)
21. Liu, S., et al.: Flow reconnaissance via timing attacks on SDN switches. In: ICDCS, pp. 196–206. IEEE (2017)
22. Liu, Z., et al.: One sketch to rule them all: rethinking network flow monitoring with UnivMon. In: SIGCOMM, pp. 101–114. ACM (2016)
23. McKeown, N.: OpenFlow: enabling innovation in campus networks. SIGCOMM CCR **38**(2), 69–74 (2008)
24. Pica8: Flow scalability per broadcom chipset, March 2018. https://docs.pica8.com/display/picos2102cg/Flow+Scalability+per+Broadcom+Chipset
25. Postel, J.: Transmission control protocol (1981)
26. Postel, J., et al.: RFC 792: Internet control message protocol. InterNet Network Working Group (1981)
27. Shin, S., et al.: AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In: CCS, pp. 413–424. ACM (2013)
28. Shin, S., Gu, G.: Attacking software-defined networks: a first feasibility study. In: HotSDN, pp. 165–166. ACM (2013)
29. Sonchack, J., et al.: Timing-based reconnaissance and defense in software-defined networks. In: ACSAC, pp. 89–100. ACM (2016)
30. Tootoonchian, A., Ganjali, Y.: HyperFlow: a distributed control plane for OpenFlow. In: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, p. 3 (2010)
31. Wang, A., et al.: Scotch: elastically scaling up SDN control-plane using vSwitch based overlay. In: CoNEXT, pp. 403–414. ACM (2014)
32. Wang, H., et al.: FloodGuard: a DoS attack prevention extension in software-defined networks. In: DSN, pp. 239–250. IEEE (2015)
33. Xu, H., et al.: Real-time update with joint optimization of route selection and update scheduling for SDNs. In: ICNP, pp. 1–10. IEEE (2016)

34. Xu, Y., Liu, Y.: DDoS attack detection under SDN context. In: INFOCOM, pp. 1–9. IEEE (2016)
35. Zhang, M., et al.: FTGuard: a priority-aware strategy against the flow table overflow attack in SDN. In: SIGCOMM Demo, pp. 141–143. ACM (2017)
36. Zhang, M., et al.: Control plane reflection attacks in SDNs: new attacks and countermeasures. Technical report, June 2018. https://www.dropbox.com/s/bnwe8apx5w06a85/sdns-attacks-countermeasures-tr.pdf?dl=0