Control Plane Reflection Attacks and Defenses in Software-Defined Networks

Menghao Zhang, *Graduate Student Member, IEEE*, Guanyu Li, Lei Xu, Jiasong Bai, Mingwei Xu, *Senior Member, IEEE*, Guofei Gu, *Fellow, IEEE*, and Jianping Wu, *Fellow, IEEE*

***Abstract*— Software-Defined Networking (SDN) continues to be deployed spanning from enterprise data centers to cloud computing with the proliferation of various SDN-enabled hardware switches and dynamic control plane applications. However, state-of-the-art SDN-enabled hardware switches have rather limited downlink message processing capability, especially for *Flow-Mod* and *Statistic Query*, which may not suffice the huge need of dynamic control plane applications. In this paper, we systematically study the interactions between the control plane applications and the data plane switches, and present two new attacks, namely 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 downlink messages towards SDN switches. Moreover, we propose a two-phase probing-triggering attack strategy, which makes the reflection attacks much more efficient and powerful. Experiments on a testbed with 3 different physical OpenFlow switches demonstrate that the attacks can lead to catastrophic results such as hurting the establishment of new flows and even disruption of connection between SDN controller and switches. To mitigate such attacks, we present several countermeasures from different perspectives. In particular, we propose a novel, systematical defense framework, SwitchGuard, to detect anomalies of downlink messages and prioritize these messages based on a novel monitoring granularity, i.e., host-application pair (HAP). Implementations and evaluations demonstrate that SwitchGuard can effectively reduce the latency for legitimate hosts and applications under the control plane reflection attacks with only minor overheads.**

***Index Terms*— Software-Defined Networking (SDN), side-channel attacks, denial-of-service attacks.**

I. Introduction

SOFTWARE-DEFINED Networking (SDN) has enabled flexible and dynamic network functionalities with a novel programming paradigm. By separating the control 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 the switches and distribute the packet processing decisions to switches accordingly. These applications enable SDN to adapt to the 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 [2], load balancing [3]–[6], network address translation [5], L2/L3 routing and so on. These new features and benefits have driven SDN paradigm to be deployed spanning from enterprise data centers to cloud computing and virtualized environments, among others [7].

Besides the substantial benefits and wide deployments, SDN has also encountered several problems. While the applications in the mainstream SDN controllers are emerging constantly [2]–[6], the development and evolution of current SDN-enabled hardware switches is much slower [8]–[19]. In particular, the control message processing capability of SDN-enabled hardware switches is rather limited, constrained by multiple factors. First, CPUs of hardware switches are usually relatively poor for financial reasons [8], [9], which restricts the message parsing and processing capability of software protocol agents in switches. Second and more importantly, flow tables in most commodity hardware switches use Ternary Content Addressable Memory (TCAM) to achieve wire-speed packet processing, which only allows limited flow table update rate (only support 100-200 flow rule updates per second [9]–[17]) and small flow table space (ranging from hundreds to a few thousands [8], [10], [18]). These limitations have slowed down network updates and hurt network visibility, which further constrains the control plane applications with dynamic policies significantly [12].

In this paper, we systematically study the interactions between the control plane applications and the data plane switches, and identify 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 Control Plane Reflection Attacks in the SDN-based network, i.e., Table-miss Striking Attack and Counter Manipulation Attack, which can effectively exploit the limited control message processing capability of SDN-enabled hardware switches. To further improve the accuracy and efficiency of Control Plane Reflection Attacks, we present a two-phase attack strategy, which consists of a probing phase and a triggering phase. With this strategy, Control Plane

Manuscript received May 19, 2020; revised October 9, 2020 and November 19, 2020; accepted November 23, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor W. Lou. Date of publication December 9, 2020; date of current version April 16, 2021. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB0801701 and in part by the National Science Foundation of China under Grant 61872426, Grant 61625203, and Grant 61832013. This article was presented at the Conference of RAID 2018. *(Corresponding author: Mingwei Xu.)*

Menghao Zhang, Guanyu Li, Jiasong Bai, Mingwei Xu, and Jianping Wu are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China, also with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China, and also with the Beijing National Research Center for Information Science and Technology (BNRist), Beijing 100084, China (e-mail: zhangmh16@mails.tsinghua.edu.cn; ligy18@mails.tsinghua.edu.cn; bjs17@mails.tsinghua.edu.cn; xumw@tsinghua.edu.cn; jianping@cernet.edu.cn).

Lei Xu and Guofei Gu are with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: xray2012@email.tamu.edu; guofei@cse.tamu.edu).

Digital Object Identifier 10.1109/TNET.2020.3040773

Reflection Attacks can be launched in a rather efficient and powerful manner, which can succeed even in the presence of state-of-the-art SDN detection and defense frameworks, such as DELTA [20] and SPHINX [21]. With Control Plane Reflection Attacks, attackers can deliberately induce a large number of control messages towards switches, which can lead to catastrophic results such as hurting the establishment of new flows and even disruption of connection between SDN controller and switches. Extensive experiments on a physical testbed with 3 types of hardware switches demonstrate that these attack vectors are highly effective and the attack effects are rather catastrophic.

In order to mitigate Control Plane Reflection Attacks, we present several countermeasures from different perspectives. Further, we propose a novel and systematical defense framework, namely SwitchGuard. SwitchGuard leverages a new monitoring granularity—*host-application pair (HAP)*—to detect downlink message[1] anomalies, and prioritizes downlink messages when the downlink channel congests. In this way, SwitchGuard is able to satisfy the latency requirements of different hosts and applications under the reflection attacks.

To summarize, our main contributions include:

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

The remainder of this paper is structured as follows. Section II introduces the background, observations, and motivations of these attacks. Section III illustrates the technical details of Control Plane Reflection Attacks and Section IV proves their harmful effects on a physical testbed. Section V presents several countermeasures to mitigate these attacks and Section VI introduces our systematical defense framework, SwitchGuard. We make some discussions in Section VII, illustrate the related works in Section VIII, and finally conclude our paper in Section IX.

II. Background and Motivation

In this section, we illustrate the background of the packet processing logics in SDN and further motivate our attacks.

Processing logic of data plane events. SDN introduces the open network programming interface and accelerates the proliferation of network applications, which enable network to dynamically adjust network configurations based on certain data plane events, as illustrated in Fig. 1. 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 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 for 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 its packet header does not match any other flow rule, the switch will forward this 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 of 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 corresponding 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.

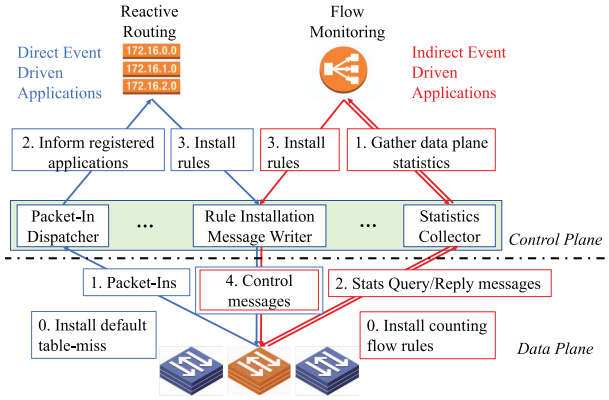


Fig. 1. Summary of current SDN architecture and its exposure to Control Plane Reflection Attacks.

Usage study of data plane events. Based on the event-driven programming paradigm, a large number of control plane applications proliferate in both academia and industry. In academia, since the advent of OpenFlow [30], a popular standard protocol between controllers and switches, many research applications have been proposed to fully leverage the benefits of direct and indirect data plane events. While direct data plane events are desired by almost all applications, indirect data plane events are also widely included. Especially, Table I shows a summarization of these indirect event-driven applications, categorized by three types, applications which help improve *optimization*, *monitoring* and *security* of network. As we can see, although each of them has different purposes, all of these works are deeply involved in the utilization of indirect data plane events, obtaining a large number of traffic features and switch attributes. Moreover, these indirect data plane events contribute a large part of communication between applications and switches. In industry, SDN applications have also experienced rapid development recently.

[1]In this paper, for brevity, we denote the messages from the data plane to the control plane as uplink messages, and the messages from the control plane to data plane as downlink messages.

TABLE I
EXAMPLES OF INDIRECT EVENT DRIVEN APPLICATIONS IN ACADEMIA

Category	Application name	Description
Optimization	Hedera [22], MicroTE [23]	SDN controller polls the data plane switches for flow statistics to detect large flows, performs resource estimation and flow scheduling.
Monitoring	OpenSketch [24], Scream [25], Mozart [26]	SDN controller configures data plane switches for measurement, and then gets the data plane statistics through a query and report procedure.
Security	FloodDefender [27], DDoS Detection [28], CloudWatcher [29]	SDN controller monitors the packets stream of each/multiple flow(s) with processing rules and extracts traffic features for classification.

TABLE II
EXAMPLES OF INDIRECT/DIRECT EVENT DRIVEN APPLICATIONS IN INDUSTRY

	Indirect event driven	Direct event driven
OpenDaylight	Statistics Application, Load Balancing, LACP (Link Aggregation Control Protocol)	Forwarding, Firewall, L2Switch
ONOS	Port Statistics, Performance Metrics Collection	Learning Switch, Reactive Forwarding, Proxy ARP/NDP
Floodlight	Link Bandwidth Utilization, FlowDiff	Firewall, Forwarding
Ryu	Traffic Monitor, QoS	Switching Hub, Firewall, Router
Nox	Load Balancing, DDoS Detection	FlowACL
Pox	DoS Mitigation	FlowQos

The mainstream SDN platforms (e.g. Open Daylight [31], ONOS [32], Floodlight [33]) foster open and prosperous markets for control plane software, which provide a great range of applications with a composition of direct and indirect data plane events. Besides, 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, as depicted in Table II, and discovered that indirect event-driven applications occupy a large part of application markets in these open source controller platforms.

Limitations of SDN-enabled hardware switches. Compared with the rapid growth in packet processing capability in logically centralized and physically distributed network operating systems (e.g., Onix [34], Hyperflow [35], Kandoo [36]) 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 only support thousands of flow entries [19]. To make matters worse, the capability to update the entries in TCAM is rather limited, usually less than 200 updates per second [9]–[17], [19]. According to our experiment on Pica8 P-3922, the maximum update rate is about 150 entries per second. Similar results of update rate are reported in a previous paper [15] for other switch brands, i.e., Broadcom, Intel, and IBM. To find out the root cause behind this phenomenon, we have conducted several conversations with different switch vendors. Even nowadays, OpenFlow support is oftentimes provided as an experimental feature in the switches, and most vendors simply adapt conventional switches to support OpenFlow. In traditional switch operation, the control channel between the ASIC and the CPU is not frequently used, so the CPU is relatively poor and the ASIC API is not so optimized. When these conventional switches are adapted to be OpenFlow-enabled, the weak CPU and unoptimized ASIC API become a prominent bottleneck. This is also a potential side channel attack surface, since these bottleneck resources are shared among all the control messages and are used differently with diverse message types. By observing these shared resources, how the resources are used would be visible to the attackers. In conclusion, the downlink channel in the switches is the dominant resource in the SDN architecture that must be carefully managed to fully leverage the benefits of the 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 potential side channel attacks.

III. CONTROL PLANE REFLECTION ATTACKS

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

A. Threat Model

We assume an adversary possesses one or more hosts or virtual machines (e.g., via malware infection) in the SDN-based network. The adversary can use his/her controlled hosts or virtual machines to initiate probe packets, monitor their responses, and generate attack traffic. Note that we do not assume the hosts that are probed by attackers are malicious, as these hosts only need to reply to the incoming packets based on the protocol stack. We also do not assume the adversary can compromise SDN controllers, applications or switches. In addition, we assume the connection between SDN controllers and switches is well protected by TLS/SSL.

B. Control Plane Reflection Attacks

Control Plane Reflection Attacks are much more 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 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. packet 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.

1) Table-miss Striking Attack : Table-miss Striking Attack is an enhanced attack vector from previous data-to-control plane saturation attack [27], [37]–[39]. Instead of leveraging a random packet generation method to conduct the attack, Table-miss Striking Attack adopts a more accurate and cost-efficient manner by utilizing probing and triggering phases.

The probing phase aims to learn the confidential information of the SDN control plane to guide the patterns of attack packet stream. The attacker could first probe the use of direct data plane events (e.g., *Packet-In*, *Packet-Out*, *Flow-Mod*) by using various low-rate probing packets whose packet header is filled with deliberately faked values. The attacker sends these probing packets to the SDN-based network and observes the corresponding responses, thus obtaining the round trip time (RTT) for each probing packet accordingly. 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. This also indicates that the specific packet header matches no flow rules in the switch and invokes *Packet-In* and *Flow-Mod* messages. Then the attacker changes 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 granularity for routing. Then the attacker could deliberately craft attack stream based on the probed granularity to deliberately trigger the expensive flow rule update operations.

2) Counter Manipulation Attack : Compared with Table-miss Striking Attack, Counter Manipulation Attack is much more sophisticated, which is based on 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 packets are required, i.e., *timing probing packets*, *test packets* and *data plane stream*. The *timing probing packets* are inspired by the *time pings* in [13], 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 on 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 the software agent of a switch, which should satisfy the following three key 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 flow rule update (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 on the switch software agent. We consider many options here for timing probing packets, e.g., ARP request/reply, ICMP request/reply, TCP SYN or UDP. For layer 2, 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 networks, it is impossible to send packets with random source MAC addresses due to pre-authorized network access control policies. To solve this challenge, the attacker could resort to the *flow rule time-out* mechanism of OpenFlow. The attacker would select N benign 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 [40], 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 [41] 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 effect of *timing probing packets* by adding extra loads to the software agent of the switch. Note that the extra loads should be appropriate to reach the *nonlinear jump* point of the hardware switch (§ IV-C.1). For the purpose, we consider test packets with a random destination IP address and broadcast destination MAC address as an ideal choice. 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. 2. The first template has a steady rate v, packet size p. The second has a rate distribution like a jump function, where three variables (v, t, p) determine the shapes of this template as well as the size of each packet. These two templates can be set with different parameters to compose complicated probing schemes to infer different control strategies.

The insight of probing phase of Counter Manipulation Attack lies in that different kinds of downlink messages

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

[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.

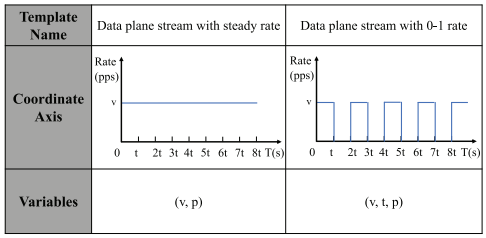


Fig. 2. Templates for data plane stream.

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 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, which 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 switch protocol agent CPU 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* by observing the probing latencies. Usually, the statistic queries are 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* under this context. Based on this phenomenon, the attacker could further infer what statistic calculation methods the application takes, such as volume-based or rate-based. We summarize this procedure in Algorithm 1, and give several concrete examples in Section IV. With several trials of two *data plane stream* templates above (lines 10-15, 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 (line 9), based on the occurrence of *double-peak phenomenon*, the attacker could quickly obtain the concrete conditions (volume/rate values, number-based or byte-based) that trigger the expensive downlink messages (lines 16-31). 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 deliberately permute the packet interval and packet size of each flow and further manipulate the counter value to the critical value. As a result, 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. All the concrete inference procedures and attack steps can be seen in Section IV.

[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.

Algorithm 1 The Probing Algorithm for Probing Phase

Data: $Generator$ - Packet streams generator, $Next$ - Function giving next value choice for v and p
Input: T - configure $Generator$ to generate T seconds of packets per stream, v_{probe} - the sending rate of *timing probing packets*, v_{test} - the sending rate of *test packets*
Output: query *policy* of applications

```
// Initialization
1  Generator.init(T);
2  v_cur = v_0, p_cur = p_0;
3  stream_probe = Generator.probe(v_probe);
4  stream_test = Generator.test(v_test);
5  stream_dp = Generator.dp(template_1, v_0, p_0, T);
   // stream_* demotes the three stream above
6  result = send_parallel(stream_*);
7  t_query = result.t;
8  while true do
       // increase represents increasing v and p
          simultaneously
9      (v_cur, p_cur) = Next(v_cur, p_cur, increase);
10     foreach template_i ∈ [template_1, template_2] do
11         stream_probe = Generator.probe(v_probe);
12         stream_test = Generator.test(v_test);
13         stream_dp = Generator.dp(template_i, v_cur, p_cur, t_query);
14         result = send_parallel(stream_*);
15         pattern_array[i] = result.pattern;
       // Data plane stream reaches the threshold of
          the policy
16     if pattern_array != [pattern3, pattern3] then
17         policy.threshold = v_cur;
           // Judge the target of the policy
18         if pattern_array == [pattern1, pattern2] then
19             policy.target = volume;
20         if pattern_array == [pattern3, pattern1] then
21             policy.target = rate;
           // modify represents increasing p and
              decreasing v with v*p unchanged
22         (v_cur, p_cur) = Next(v_cur, p_cur, modify);
23         stream_probe = Generator.probe(v_probe);
24         stream_test = Generator.test(v_test);
25         stream_dp = Generator.dp(template_2, v_cur, p_cur, t_query);
26         result = send_parallel(stream_*);
27         if result.pattern == pattern3 then
28             policy.counter = packet_number;
29         else
30             policy.counter = packet_byte;
31             policy.threshold = v_cur * p_cur;
32         return policy;
```

IV. Attack Evaluation

In this section, we demonstrate our experimental results of Control Plane Reflection Attacks with a physical testbed. We first show the experiment setup in our evaluations, and then conduct extensive experiments for Table-miss Striking Attack and Counter Manipulation Attack separately. Finally, we perform some benchmarks to provide low-level details and insights behind our proposed attacks.

A. Experimental Setup

To demonstrate the feasibility of Control Plane Reflection Attacks, we set up an experimental scenario as shown in Fig. 3.

TABLE III
INDIRECT EVENT DRIVEN APPLICATIONS IN OUR TESTBED

Application Name	Descriptions	Triggering Conditions	Triggering Actions
Heavy Hitter [44]	Perform per-flow port balancing for flows whose size exceed thresholds	Flow size difference between two statistic messages for a flow exceeds a threshold	Issue one or more *Flow-Mod* messages to perform per-flow port balancing
Microburst [45]	Perform queue-based balancing for flows whose packet numbers within a window exceed a threshold	Queue length difference between two statistic messages for a port exceeds a threshold	Issue one or more *Flow-Mod* messages to perform per-packet balancing
PIAS [46]	Inserts flow into a lower priority queue for flows whose bytes sent exceed a threshold	Flow sent bytes obtained from a statistic message for a flow exceeds a threshold	Issue one or more *Flow-Mod* messages to change flows' priority
DDoS Detection [28]	Detect DDoS attacks with adaptively balancing the coverage and granularity of detection	Flow volume or flow rate reached from several statistic messages for a flow exceeds a threshold	Issue one or more *Flow-Mod* messages to modify the granularity of attack detection

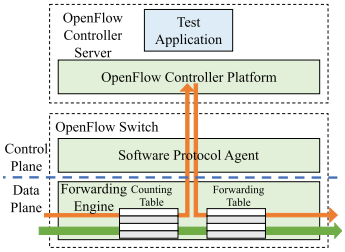


Fig. 3. A typical attack scenario.

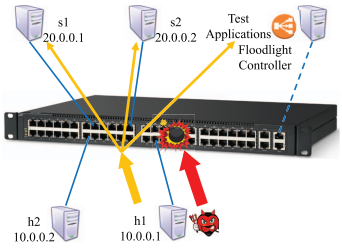


Fig. 4. Attack experimental setups.

We choose several representative applications, and run them separately on the SDN controller. Flow tables in the switch are divided into two pipelines, *Counting Table* for indirect data plane event, *Forwarding Table* for 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 application implementations today [13], [42].

Reactive Routing. *Reactive Routing* is the most common application integrated into most of the popular controller platforms [31]–[33], [43]. 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 granularity. 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. *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 one 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. In particular, we extend our *Flow Monitoring* sketch into four concrete indirect data plane events driven applications, **Heavy hitter** [44], **Microburst** [45], **PIAS** [46] and **DDoS Detection** [28], according to their logic as shown in Table III.

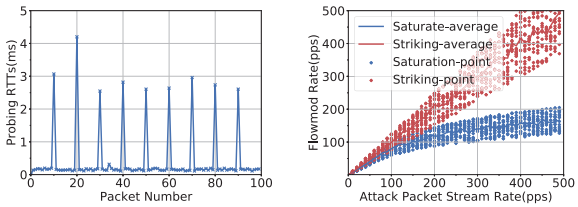
Our evaluations are conducted on a physical OpenFlow testbed, testing several representative SDN-enabled hardware switches, i.e., Pica8 P-3290, HP 5406zl, Dell 8132. These switch brands are widely used in academia/industry and support many advanced OpenFlow data plane features, such as multiple pipelines and almost full OpenFlow specifications (from version 1.0 to 1.5). Since these switches show similar limited control message processing capability (e.g., weak CPU, limited TCAM entry update rate), the experimental results are similar unless otherwise specified. The experimental topology, as shown in Fig. 4, includes four machines (i.e., h1, h2, s1, and s2) connected to the hardware switch and a server running a popular SDN controller Floodlight. 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.[5] 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 seconds, and conduct the corresponding control (e.g., issue one *Flow-Mod* message) according their logic separately. All the experiments are repeated three times to report the average circumstances. Because the results have a small deviation across runs, we do not show confidence intervals here.

B. 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*.

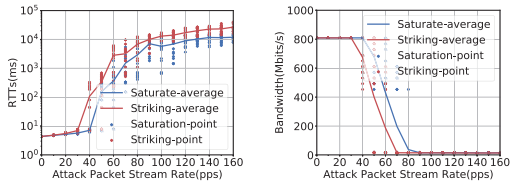
1) Table-miss Striking Attack : For the *Reactive Routing* application, when we launch a new flow, the first packet is inclined to get a high RTT, and the following several packets would get low RTTs. Since there are only three hosts on our testbed and *ping* could launch only one new flow between each host pair, we resort to UDP probing packets. We compute the time difference between the request and reply to obtain the RTT. As depicted in Fig. 5(a), we let h1 transmit 10 UDP probing packets to a destination port

[5]We have also tested these applications on other controllers, e.g., ONOS, which suffer from similar attack effects. Note that our attacks are agnostic to the concrete SDN controller implementations, because these two kinds of data plane events are heavily used by almost all the SDN controllers and all the SDN-enabled hardware switches suffer from such a bottleneck in the downlink channel.



(a) RTTs for Reactive Routing when UDP port changes. (b) Attack efficiency for Reactive Routing.

Fig. 5. 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.

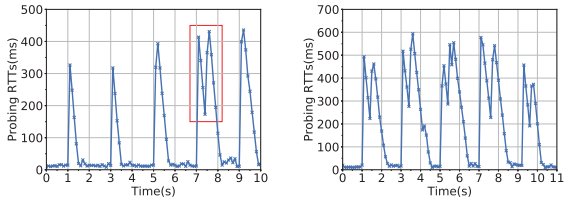
Fig. 6. RTTs and bandwidth comparison under the saturation attack and the striking attack.

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 phenomenon indicate that five-tuples grained forwarding policy is adopted by *Reactive Routing*.

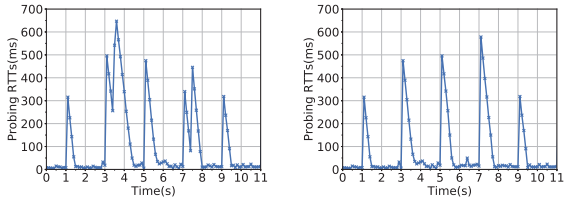
After inferring the forwarding granularity, the attacker can then carefully craft a stream of packets whose header spaces vary according to the granularity. In this way, each attack packet could strike the default *table-miss* in the switch, thus triggering *Packet-In* and *Flow-Mod* in the controller further. 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. 5(b), Control Plane Reflection Attacks are much more efficient than Data-to-control Plane Saturation Attack. Moreover, we also compare the RTTs and bandwidth for normal users under the saturation attack and the striking attack. As shown in Fig. 6, 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.

2) Counter Manipulation Attack : For the *Flow Monitoring*-based applications, we first supply a steady rate of *test packets*,[6] which would put appropriate loads on the switch control plane. The rate of *test packets* depends on the CPU capability of the target switch. For different brands of switches, it should be tested beforehand to make the periodical peaks more obvious by putting an appropriate load to the switch CPU. The rate of *timing probing packets* is set as 10 pps. The results for four applications are similar, as shown in Fig. 7(a). As we could conclude, *Flow Monitoring*-based applications poll the switch for statistics every 2 seconds. 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*

[6]300 packets per second (pps) for Pica8 P-3290, 250 pps for HP 5406zl, and 200 pps for Dell 8132. Note that hundreds of pps is a fairly secure rate, since a legitimate host could issue packets at thousand of pps under normal circumstance.



(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. 7. Timing-based patterns for counter manipulation attack.

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 *Statistics Query* are much more expensive than *Packet-Out* 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. 7(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. 7(c)), where v is also a big value, she/he could determine that volume-based (lines 18-19 in Algorithm 1) and packet-number-based (lines 22-28 in Algorithm 1) 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 IV. 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, as illustrated in Algorithm 1 (line 17 and 31). In addition, we can verify our inference with a lot of other ways, not only the proposed two data plane stream templates as shown above, which we intend to expand over time. In particular, we test our four indirect event driven applications, and find them fall in the distribution in Table V. This is consistent with the policies of each application, which demonstrates the effectiveness of our probing phase.

With the results (query period, packet number/byte-based, volume/rate values) obtained from the probing phase, we move to the second step and start to launch our Counter Manipulation Attack. We select one application, PIAS, as a representation to be shown here, and set its priority as 3 levels. We initiate 10 new flows every second, and carefully set the sent bytes of each flow in each period, making it exactly bigger than the critical value we probed. As a consequence, a number of *Flow-Mod* are issued to the switch when the statistic query/reply occurs. As shown in Fig. 8, the number of *Flow-Mod* messages could increase to as high as 60 at the end of each period. This would incur quite high loads to the software agent of the switch at this moment. As an extreme case, when the attacker controls about one thousand flows intentionally and manipulates all the flow to trigger the critical values simultaneously, thousands of

TABLE IV
RELATIONSHIP BETWEEN DATA PLANE STREAM AND DOUBLE-PEAK PHENOMENON

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

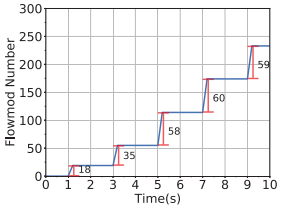


Fig. 8. *Flow-Mod* message triggering number for PISA under attacks.

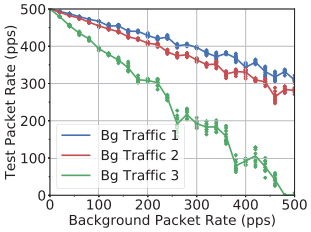


Fig. 9. Different background traffic requires different rate of test packets.

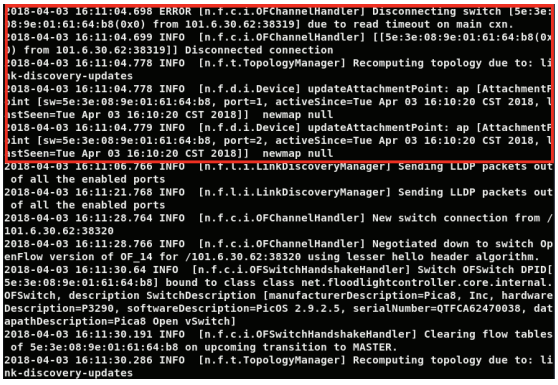


Fig. 10. Switch re-connection under the extreme case.

Flow-Mod messages are directed to the switch at the end of each period, which would cause catastrophic results, i.e., the connection between the SDN controller and the switch is disrupted, as shown in Fig. 10.

C. Attack Fundamentals and Analysis

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

1) Test Packet *Rate and* Test Packet *Type:* Fig. 11 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. Fig. 12 shows the timing probe RTTs as the *Statistics Query* rate varies. Fig. 13 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 see 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 a

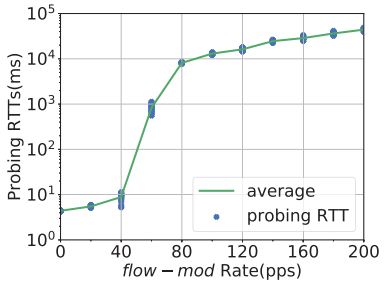


Fig. 11. Timing probe RTTs as *Flow-Mod* rate varies.

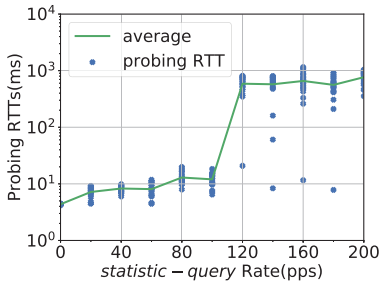


Fig. 12. Timing probe RTTs as *Statistic Query* rate varies.

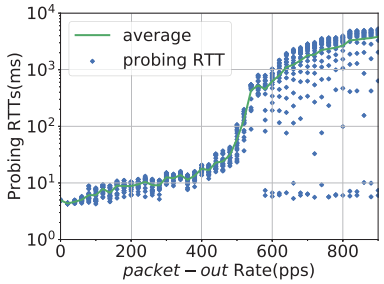


Fig. 13. Timing probe RTTs as *Packet-Out* rate varies.

Flow-Mod message for each test packet, the RTTs experience a ∼1000 times increase at ∼60 pps. For *Statistics Query* messages, the RTTs increase ∼100 times at ∼110 pps. And for *Packet-Out* messages, the RTTs increase ∼100 times at ∼500 pps. Meanwhile, we measure the resource usage of the hardware switch and the controller, and find that the switch CPU could reach above 90% at the point of nonlinear jump, while the memory of switch, the CPU and memory of control server are relatively low (at most 30%). In addition, we have conversations with several switch vendors (e.g., Pica 8, Biwei Network Technology Co., Ltd), and learn 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 the insertion of a new flow rule requires the rearrangement of rules in TCAM, which lead to the results that the expense of *Flow-Mod* >= *Statistics Query* ≫ *Packet-Out* [13], [19].

2) The Impact of Background Traffic: The background traffic does have 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; rather, 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. To illustrate this, we use three different background traffic by replaying them at an increased rate and conduct the probing phase in our testbed. For a certain rate of background traffic, we select a rate of test packets that can lead to an obvious and periodical peaks for the RTTs of timing probing packets. As we can see from Fig. 9, with the rate of background traffic increasing, the rate of test packets decreases. This is because when the rate of background traffic increases, less test packets are required to fit the load of the nonlinear jump point in the hardware switches. More importantly, under our test, Control Plane Reflection Attacks can always succeed, which demonstrate that our attacks are agnostic to the background traffic. Certainly, an excessively high-rate background traffic may lower the probing accuracy. This is because when the load exceeds the nonlinear jump, the loads incurred by *Statistics Query* would not cause the obvious and periodical peaks for the RTTs of timing probing packet, instead, the patterns may become random and irregular. However, in such cases, the switch is already suffering, and the network inclines to become problematical. Second, the background traffic may also affect the trigger phase. Actually, this influence is positive as well. The background traffic would always need some interactions with the controller, thus inevitably bringing some downlink messages to the control channel, which would boost the effects of control plane reflection attacks.

3) Study of Application Control Policies: During our investigation on the indirect event driven control plane applications, an interesting thing we find is that although the logics and purposes of different applications vary, they all follow a similar procedure: collecting, analyzing, and controlling, as illustrated in Fig. 1. Since the switch plays a complementary role to maintain the view not directly visible to the controller, current switch [47] only provides several essential data plane statistics, such as packet counters and byte counters. Then the control plane applications analyze these pulled statistics and make decisions whether issuing flow rules to adjust the state of the data plane. Surprisingly, the seemingly complicated control logics could mainly be divided into two-multiple-two fundamental dimensions, i.e., (Packet Number, Packet Byte) $\times$ (Volume-based, Rate-based). We find nearly all the known applications from both academic and industry can be categorized into these four distributions, like Table V.

TABLE V
DISTRIBUTION OF THE FOUR INDIRECT EVENT DRIVEN APPLICATIONS

	Volume-based	**Rate-based**
Packet Number	Microburst	-
Packet Byte	Heavy Hitter PIAS DDoS Detection	DDoS Detection

V. COUNTERMEASURE ANALYSIS

The control plane reflection attack is deeply rooted in SDN architecture that the performance of existing commodity SDN-enabled hardware switches could not suffice the need of the SDN applications. We therefore give several countermeasures against these reflection attacks.

The root cause of the reflection attacks is that an attacker can force the control plane applications to reflect responses to the switches. If we disable the dynamic features of the applications, just as traditional network where the traffic in the data plane will not change the configurations of the switch from the control plane, the attack surface will be eliminated. While effective, this approach is undesirable since it comes at the expense of less fine-grained control, visibility, and flexibility in traffic management, as evidently required in [9], [48], [49]. Alternatively, the data plane capacity can be increased by optimizing the software protocol agent implementation or adding computation components to the switches. However, the capacity and the update rate of TCAM is inherently rooted in the hardware design of memory chip, which could not be improved immediately and would be the bottleneck of the switches in the foreseeable future [17]. The vantage point we want to claim here is that the applications should be carefully designed and deployed under the dynamic SDN circumstance, and this potential attack surface should be taken into consideration when the applications want to use these new dynamic features.

Despite limiting the use of dynamic features for network applications, a more promising defense methodology is admitting the benefit of this dynamic control and devising some approaches to make the reflection attacks unfeasible. One promising method is to deploy some network intrusion detection systems (IDSs) with the new attack signatures in the SDN-based network. If the IDSs detect such new probing/triggering patterns, quarantine the suspicious hosts and alert the network operators. Another way is to add some detection and prevent mechanisms in the controller. The effect of the reflection attacks lies in that a flash crowd of downlink messages swarms to the switch. If we could deploy some detection measure and limit the downlink message transmission rate in the controller, this could effectively prevent the switches from being overwhelmed. In some cases, if the latency of control messages is not a concern, we can intuitively add some latency to random downlink messages. This would make the patterns/policies of direct/indirect data plane events difficult to sniff and obtain. These approaches can effectively breach either the probing phase or the triggering phase, making the attacks hard to be conducted.

VI. SWITCHGUARD: A PRIORITY-BASED SCHEDULER ON SWITCH

Although the aforementioned countermeasures can take effect under some certain scenarios, in this section, we give a more general and systematical solution, SwitchGuard, to mitigate these reflection attacks. The key idea is to discriminate good from evil, and prioritize downlink messages with discrimination results. 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

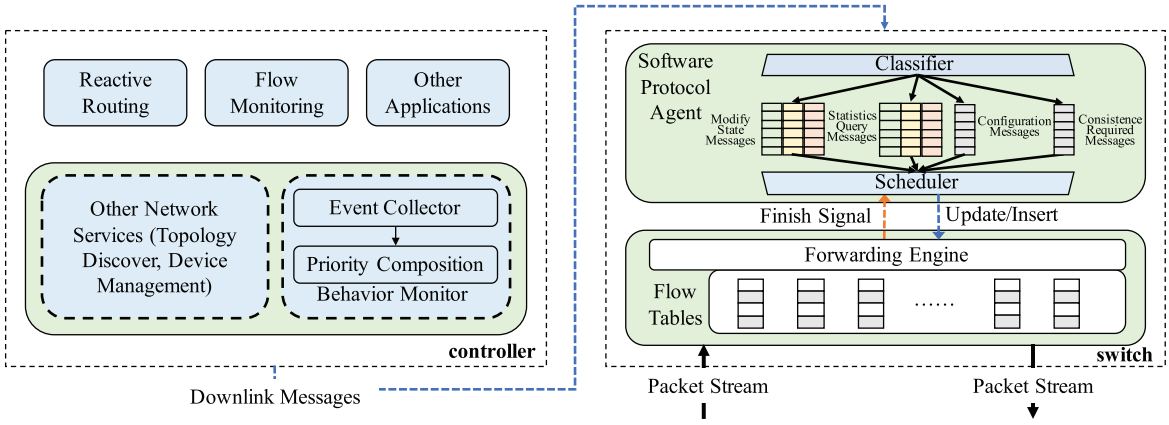


Fig. 14. SwitchGuard framework design.

a low-priority scheduling queue and the requirements of *good* messages are more likely to be satisfied.

A. SwitchGuard Design

The architecture of SwitchGuard is shown in Figure 14.[7] SwitchGuard 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*.

1) 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 such that the handshake 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*, (2) *Statistic Query Messages*, (3) *Configuration Messages*, and (4) *Consistency Required Messages*, 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 Section VI-A.3, 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.

[7]In SwitchGuard, we use the original single-queue model to process uplink messages, and omit this detail in Figure 14.

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 queue of quick level is set as 0, for slow level is set as a small time interval, while for block level is set as a relatively bigger value. With the principles illustrated above, we design the scheduling algorithm as Algorithm 2.

Algorithm 2 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();
```

2) Behavior Monitor: In order to distinguish different downlink messages with different priorities, we require an appropriate *Monitoring* granularity. Previous approaches mainly conduct the monitoring with the granularity of source host [28], [50], 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 [51],

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. As discussed in Section II, the downlink channel for processing downlink messages is the domain resource in the SDN architecture that it must be carefully managed and allocated to fully leverage the benefits of the SDN applications. Considering K applications exist on the control plane, their requirements for downlink messages are represented as vector $\vec{a_0} = \langle a_1, a_2, \ldots a_K \rangle$, and N hosts/users in the data plane, corresponding requirements vector $\vec{h_0} = \langle h_1, h_2, \ldots h_N \rangle$. $\vec{a_0}$ and $\vec{h_0}$ 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*

$$\mathbf{R_0} = \vec{a_0}^T \cdot \vec{h_0} = \begin{pmatrix} a_1h_1 & a_1h_2 & \ldots & a_1h_N \\ a_2h_1 & a_2h_2 & \ldots & a_2h_N \\ \vdots & \vdots & \ddots & \vdots \\ a_Kh_1 & a_Kh_2 & \ldots & a_Kh_N \end{pmatrix}$$

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*

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & \ldots & r_{1N} \\ r_{21} & r_{22} & \ldots & r_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ r_{K1} & r_{K2} & \ldots & r_{KN} \end{pmatrix}$$

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 SwitchGuard system as attack-driven, which means when $\frac{Sum}{Sum_0} < \alpha$, SwitchGuard 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.

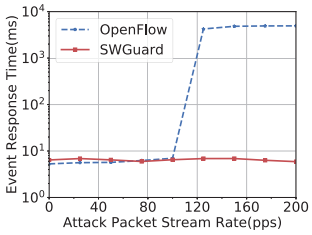


Fig. 15. Defense effect.

3) Policy Consistency: Multi-queues 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-queues 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, their 2-bit priority packet metadatas are supposed to be tagged with 11 by the *Priority Composition*. Then the *Classifier* in the software protocol agent will check the label to learn whether the message has the consistency demand. If the consistency demand is confirmed, this message will be scheduled to the queue for consistency required messages.

B. Defense Evaluation

We implement the prototype of SwitchGuard system, including *Behavior Monitor* and *Software Protocol Agent*, on Floodlight [33] and Open vSwitch [52][8] with about 4000 Lines of Code. We set up PISA as a typical application on the Floodlight controller. 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. In our testbed, the time intervals for the three levels on the *Scheduler* are set as 0, 3 and 10 respectively (nanoseconds), the two thresholds(th_h, th_l) on the *Priority Composition* Module are set as 0.5 and 0.1 accordingly.

To demonstrate the defense effect of SwitchGuard, we use the average value of the flow rule installation / statistic query latency of normal users/applications as representative metrics, which is named as *Event Response Time* in our figures. As shown in Figure 15,[9] for legitimate messages, with the native system, the event response time becomes extremely large when the rate of downlink messages is above 110 packets per second. While with SwitchGuard, the event response time is nearly unchanged. These experimental results illustrate that our SwitchGuard provides effective protection for both the legitimate flow rule installation and the legitimate statistics query.

[8]Since we cannot modify the (proprietary) software agent in hardware switches, we have to use Open vSwitch, an open-source software switch, to demonstrate the effectiveness of our SwitchGuard solution. In future, we plan to cooperate with switch vendors and integrate our solution into their next-generation products.

[9]Since this experiment is conducted on the software environment, the non-linear jump point is a little different from the previous hardware experimental results.

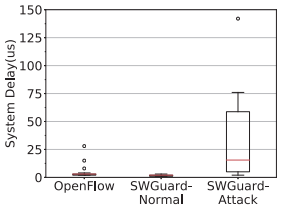


Fig. 16. Defense overhead.

As for the overheads of SwitchGuard, we measure the latency introduced by SwitchGuard. Compared with native OpenFlow, packets in SwitchGuard 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 attack happens, packets must pass a full path in Behavior Monitor and Software Protocol Agent. As shown in Fig 16, the latency is almost the same between native OpenFlow and SwitchGuard under normal circumstance. Even under attacks, Behavior Monitor and Software Protocol Agent only incur a latency less than 100 us in average. All of these demonstrate that SwitchGuard only incur negligible delay for the control channel messages.

VII. Discussion

Network access control policy. One may concern that whether effective network access control policies can prevent such Control Plane Reflection Attacks. In fact, since the signature of our second attack vector, Counter Manipulation Attack, has not been studied by academia/industry previously, it is rather hard for operators to install a set of accurate access control policies. Moreover, blindly deploying access control policies would inevitably cause collateral damages for legitimate users. As a result, it must be possible that attacker-controlled host can probe some benign hosts in the SDN-based network. Therefore, our Counter Manipulation Attack can be successfully launched.

Independence of attack vectors and applications. We only concentrate on the applications' composition of two types of data plane events, somewhat akin to the idea of *one big application* abstraction. We build our attack vectors with respect to these two data plane events, making the concrete logics of the control plane independent on our attack vectors.

Emerging programmable data planes. Current prototypes, attack and defense are based on OpenFlow-based hardware switches. We believe the core idea of control plane reflection attack is applicable to the emerging generation of programmable data planes, e.g. P4 and RMT chips [53], because these platforms also use traditional TCAM-based flow tables and control plane reflection attack addresses a property of TCAM that is invariant to underlying TCAM design.

Generality of the SwitchGuard system. SwitchGuard is also applicable for no-adversary circumstances, such as flash crowds of downlink messages under normal conditions. By prioritizing the downlink messages, SwitchGuard can provide lower latencies for more important messages under the congestion of control channel.

Evading SwitchGuard defense. Like any anomaly detection systems, SwitchGuard is not perfect or complete. If a attacker wants to evade the SwitchGuard defense, to the best of our knowledge, the only approach is to reduce the occupation of downlink resources. Nevertheless, in this way, the goal of protection is already achieved.

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 SwitchGuard, we can also harness an 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.

***Priority* field of control messages.** During our experiments, we discover that the *priority* orders of a set of flow rules have a significant impact on the insertion rate of flow rules. In particular, inserting a sequence of flow rules in order of increasing priority would be extremely heavyweight, in which case the switch could crash unexpectedly. Priority should be carefully used during dynamic configuration, especially when this configuration could be triggered by the data plane traffic.

VIII. Related Work

Our work is deeply inspired by the following two topics.

DoS attacks in SDNs. Shin and Gu [37] first proposes the concept of data-to-control plane saturation attack against SDN. To mitigate this dedicated DoS attack, AVANT-GUARD [38] introduces *connection migration* module and *actuating triggers* module to extend the data plane functions. However, it can be applicable to TCP protocol only. Later, a protocol-independent defense framework, FloodGuard [39], pre-installs proactive flow rules to reduce table-miss packets, and forwards table-miss packets to additional data plane caches. Furthermore, to obtain the benefit of no hardware modification and addition, FloodDefender [27] 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 quite straightforward attack method that attacker simply floods arbitrary attack traffic to trigger the direct data plane events while reflection attacks resort to a more advanced and sophisticated techniques, and a two-phrase 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 attack is more stealthy and same attack expenses of the attacker could cause more obvious attack effects for victims. In addition, Scotch [9] alleviates the communication bottleneck between control plane and data plane with a pool of vSwitches distributed across the network, and it shares the same observation that SDN-enabled hardware switches have very limited capacity for control channel communications.

Timing-based side channel attacks. Side channel attacks have long existed in distributed systems, and it is 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 and Gu [37] presents an SDN scanner which could determine whether a network is using SDN or not. Leng *et al.* [54] proposes to measure the response time of requests to obtain the approximate capacity of switch's flow table. Sonchack *et al.* [13] 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.* [55] 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.

IX. 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 and powerful. The experiments on a physical testbed showcase the reflection attacks can cause extremely harmful effects with acceptable attack expenses. To mitigate reflection attacks, we discuss several countermeasures from diverse perspectives, and further propose a general, systematical defense framework SwitchGuard, to detect anomalies of control messages and prioritize them based on the host-application pair. The evaluation results of SwitchGuard demonstrate its effectiveness under the reflection attacks with minor overhead. As SDN is adopted widely and more dynamic applications emerge in both academic and industry, it is likely that the reflection attack will be observed in the wild. We hope our work could act as a catalyst to make researchers and practitioner rethink the interactions between the data plane and the control plane, and devise further ways to promote the coordinated development of both to make SDN more secure.

References

[1] M. Zhang, G. Li, L. Xu, J. Bi, G. Gu, and J. Bai, "Control plane reflection attacks in SDNs: New attacks and countermeasures," in *Research in Attacks, Intrusions, and Defenses*. Cham, Switzerland: Springer, 2018, pp. 161–183.

[2] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic access control for enterprise networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw. (WREN)*, 2009, pp. 11–18.

[3] R. Wang, D. Butnariu, and J. Rexford, "OpenFlow-based server load balancing gone wild," *Hot-ICE*, vol. 11, p. 12, Mar. 2011.

[4] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari, "Plug-n-serve: Load-balancing Web traffic using openflow," *ACM Sigcomm Demo*, vol. 4, no. 5, p. 6, 2009.

[5] M. Koerner and O. Kao, "Multiple service load-balancing with openflow," in *Proc. IEEE 13th Int. Conf. High Perform. Switching Routing*, Jun. 2012, pp. 210–214.

[6] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 15–28.

[7] G. M. Insights. (Jul. 2018). *Software Defined Networking Market to Hit $88bn by 2024: Global Market Insights*. [Online]. Available: https://globenewswire.com

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, Aug. 2011.

[9] A. Wang, Y. Guo, F. Hao, T. V. Lakshman, and S. Chen, "Scotch: Elastically scaling up SDN control-plane using vSwitch based overlay," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Experiments Technol. (CoNEXT)*, 2014, pp. 403–414.

[10] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow: Dependency-aware rule-caching for software-defined networks," in *Proc. Symp. SDN Res.*, Mar. 2016, p. 6.

[11] H. Xu, Z. Yu, X.-Y. Li, C. Qian, L. Huang, and T. Jung, "Real-time update with joint optimization of route selection and update scheduling for SDNs," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–10.

[12] X. Jin *et al.*, "Dynamic scheduling of network updates," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 539–550, 2014.

[13] J. Sonchack, A. Dubey, A. J. Aviv, J. M. Smith, and E. Keller, "Timing-based reconnaissance and defense in software-defined networks," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, Dec. 2016, pp. 89–100.

[14] H. Chen and T. Benson, "The case for making tight control plane latency guarantees in SDN switches," in *Proc. Symp. SDN Res.*, Apr. 2017, pp. 150–156.

[15] K. He *et al.*, "Measuring control plane latency in SDN-enabled switches," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res. (SOSR)*, 2015, p. 25.

[16] K. He, J. Khalid, S. Das, A. Akella, E. Li, and M. Thottan, "Mazu: Taming latency in software defined networks," Dept. Comput. Sci., Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep., 2014.

[17] X. Wen *et al.*, "RuleTris: Minimizing rule update latency for TCAM-based SDN switches," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2016, pp. 179–188.

[18] A. Lazaris *et al.*, "Tango: Simplifying SDN control with automatic switch property inference, abstraction, and optimization," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2014, pp. 199–212.

[19] M. Kuzniar, P. Peresíni, D. Kostic, and M. Canini, "Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches," *Comput. Netw.*, vol. 136, pp. 22–36, May 2018.

[20] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. Porras, "DELTA: A security assessment framework for software-defined networks," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, vol. 17, 2017.

[21] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, vol. 15, 2015, pp. 8–11.

[22] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. Netw. Syst. Design Implement. Symp. (NSDI)*, vol. 10, 2010, p. 19.

[23] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine grained traffic engineering for data centers," in *Proc. 7th Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2011, p. 8.

[24] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. NSDI*, vol. 13, 2013, pp. 29–42.

[25] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "SCREAM: Sketch resource allocation for software-defined measurement," in *Proc. 11th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2015, p. 14.

[26] X. Liu, M. Shirazipour, M. Yu, and Y. Zhang, "MOZART: Temporal coordination of measurement," in *Proc. Symp. SDN Res.*, Mar. 2016, p. 13.

[27] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks," in *Proc. IEEE INFOCOM-IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[28] Y. Xu and Y. Liu, "DDoS attack detection under SDN context," in *Proc. IEEE INFOCOM-35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[29] S. Shin and G. Gu, "CloudWatcher: Network security monitoring using OpenFlow in dynamic clouds? Networks (or: How to provide security monitoring as a service in clouds?)," in *Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2012, pp. 1–6.

[30] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[31] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a model-driven SDN controller architecture," in *Proc. IEEE Int. Symp. World Wireless, Mobile Multimedia Netw.*, Jun. 2014, pp. 1–6.

[32] P. Berde *et al.*, "Onos: Towards an open, distributed SDN OS," in *Proc. HotSDN*, 2014, pp. 1–6.

[33] F. Community. (Aug. 2017). *Floodlight*. [Online]. Available: http://www.projectfloodlight.org/floodlight/

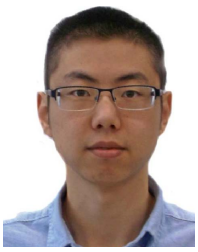
[34] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. OSDI*, vol. 10, 2010, pp. 1–6.

[35] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw.*, 2010, p. 3.

[36] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2012, pp. 19–24.
[37] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2013, pp. 165–166.
[38] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 413–424.
[39] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 239–250.
[40] J. Postel *et al. Internet Control Message Protocol*, document RFC 792, Internet Network Working Group, 1981.
[41] J. Postel, "Transmission control protocol," IETF, Tech. Rep. RFC, 1981.
[42] Y. Li, G. Yao, and J. Bi, "Flowinsight: Decoupling visibility from operability in SDN data plane," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 137–138, 2015.
[43] A. Community. (Jul. 2017). *HP SDN App Store*. [Online]. Available: http://community.arubanetworks.com/t5/SDN-Apps/ct-p/SDN-Apps
[44] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. Conf. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 101–114.
[45] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "DRILL: Micro load balancing for low-latency data center networks," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 225–238.
[46] W. Bai, K. Chen, H. Wang, L. Chen, D. Han, and C. Tian, "Information-agnostic flow scheduling for commodity data centers," in *Proc. NSDI*, 2015, pp. 455–468.
[47] ONF. (2014). *Openflow Switch Specification Version 1.5.0*. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf
[48] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, 2007.
[49] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and flexible cellular core network architecture," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2013, pp. 163–174.
[50] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *Proc. IEEE Local Comput. Netw. Conf.*, Oct. 2010, pp. 408–415.
[51] M. Zhang, J. Bi, J. Bai, Z. Dong, Y. Li, and Z. Li, "FTGuard: A priority-aware strategy against the flow table overflow attack in SDN," in *Proc. SIGCOMM Posters Demos-SIGCOMM Posters Demos*, 2017, pp. 141–143.
[52] O. vSwitch Community. (Aug. 2017). *Open Vswitch*. [Online]. Available: http://openvswitch.org/
[53] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 87–95, Jul. 2014.
[54] J. Leng, Y. Zhou, J. Zhang, and C. Hu, "An inference attack model for flow table capacity and usage: Exploiting the vulnerability of flow table overflow in software-defined network," 2015, *arXiv:1504.03095*. [Online]. Available: http://arxiv.org/abs/1504.03095
[55] S. Liu, M. K. Reiter, and V. Sekar, "Flow reconnaissance via timing attacks on SDN switches," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 196–206.

Menghao Zhang (Graduate Student Member, IEEE) received the B.S. degree in computer science from Tsinghua University, China, where he is currently pursuing the Ph.D. degree with the Institute of Network Science and Cyberspace. His research interests include software-defined networking, network function virtualization, and cyber security.

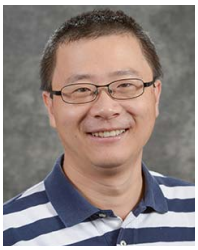
Guanyu Li received the B.S. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, China. He is currently pursuing the Ph.D. degree with the Institute of Network Science and Cyberspace, Tsinghua University. His research interests include software-defined networking, network function virtualization, and cyber security.



Lei Xu received the B.S. degree from the School of Computer Science, Nanjing University of Posts and Telecommunications, China, and the Ph.D. degree in computer science from Texas A&M University, College Station, TX, USA. He is currently an Engineer with Palo Alto Networks. His research interests include network/system security, software-defined networking, and network function virtualization.

Jiasong Bai received the B.S. degree in computer science from Tsinghua University, China, where he is currently pursuing the master's degree with the Institute of Network Science and Cyberspace. His research interests include software-defined networking, network function virtualization, and cyber security.

Mingwei Xu (Senior Member, IEEE) received the B.S. and Ph.D. degrees from Tsinghua University. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. His research interest includes computer network architecture, high-speed router architecture, and network security.



Guofei Gu (Fellow, IEEE) received the Ph.D. degree in computer science from the College of Computing, Georgia Tech, in 2008. He is currently a Full Professor with the Department of Computer Science and Engineering, Texas A&M University. His research interests include network and systems security, such as malware and APT defense, software-defined networking (SDN/NFV) security, mobile and IoT security, and intrusion/anomaly detection.

Jianping Wu (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees from Tsinghua University, Beijing, China. He is currently a Full Professor and the Director of the Network Research Center and a Ph.D. Supervisor with the Department of Computer Science and Technology, Tsinghua University. Since 1994, he has been an in charge of the China Education and Research Network. His research interests include the next-generation Internet, IPv6 deployment and technologies, and Internet protocol design and engineering.