Disrupting the SDN Control Channel via Shared Links: Attacks and Countermeasures

Renjie Xie, Jiahao Cao, Qi Li, *Senior Member, IEEE*, Kun Sun, *Member, IEEE*, Guofei Gu, *Fellow, IEEE*, Mingwei Xu, *Senior Member, IEEE*, and Yuan Yang, *Member, IEEE*

***Abstract*— Software-Defined Networking (SDN) enables network innovations with a centralized controller controlling the whole network through the control channel. Because the control channel delivers all network control traffic, its security and reliability are of great importance. For the first time in the literature, we propose the *CrossPath* attack that disrupts the SDN control channel by exploiting the shared links in paths of control traffic and data traffic. In this attack, crafted data traffic can implicitly disrupt the forwarding of control traffic in the shared links. As the data traffic does not enter the control channel, the attack is stealthy and cannot be easily perceived by the controller. In order to identify the target paths containing the shared links to attack, we develop a novel technique called *adversarial path reconnaissance*. Our experimental results show its feasibility and efficiency of identifying the target path. We systematically study the impacts of the attack on various network applications in a real SDN testbed. Experiments show the attack significantly degrades the performance of existing network applications and causes serious network anomalies, e.g., routing blackhole, flow table resetting, and even network-wide DoS. To defeat the *CrossPath* attack, we design a lightweight defense system named *CrossGuard*. Experiments demonstrate that it can effectively protect the control channel and quickly locate the attack flow with 98% accuracy while introducing a small overhead.**

***Index Terms*— SDN, shared link, control channel attack, defense system.**

I. Introduction

SOFTWARE-DEFINED NETWORKING (SDN) becomes increasingly popular and is being widely deployed in data centers [2], cloud networks [3], and wide area networks [4]. In SDN, the control plane and data plane are decoupled. A logically centralized controller communicates with SDN switches to exchange control messages, e.g., routing decisions, via the control channel built upon a southbound protocol, e.g., OpenFlow [5]. SDN enables diversified packet processing and drives network innovation. A large number of network services and applications [6]–[12] benefit from it.

Unfortunately, the SDN control channel between the control plane and data plane is not well protected and can be exploited though the confidentiality and integrity of the communication over the channel are protected by the TLS/SSL protocol. We find that the control channel is under the risk of the Denial-of-Service (DoS) attack. A small portion of traffic may tear down the communication over the control channel. Existing studies focus on many security aspects of SDN, including malicious or buggy applications [13], [14], attacks on crashing controllers [15]–[17], attacks on disrupting switches [18], [19], and information leakage in SDN [20]–[23], but the security of the SDN control channel is still an open problem.

In this paper, we propose a novel attack named *CrossPath Attack*, which disrupts the SDN control channel by exploiting the shared links between paths of control traffic and data traffic. Our attack is stealthy and cannot be easily perceived by the controller since it does not directly send a large volume of control traffic to the controller. Instead, it generates well-crafted data traffic in the shared links to implicitly interfere with the delivery of the control traffic while the data traffic does not reach the controller. Thereby, real-time control messages delivered between the SDN controller and the switches are significantly delayed or dropped. In particular, since the controller performs centralized control over all network switches via the control channel, an attacker can easily break down all network functionalities enabled by various SDN applications running on the controller. The root cause of the vulnerability is the side effect incurred by shared links between paths of control traffic and data traffic in SDN. Such link sharing is a common practice in SDN with in-band control [17], [24], which can greatly reduce the cost of building a dedicated control network and simplify network maintenance, especially for large networks. However, it also opens the door for an attacker to disrupt the control channel by sending malicious data traffic to the shared links.

It is challenging to construct the attack in real networks. Unlike traditional networks where almost all links deliver both control traffic (e.g., OSPF or BGP updates [25], [26]) and data traffic at the same time, only a few number of links

Manuscript received September 7, 2021; revised February 15, 2022; accepted March 20, 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Liu. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61832013, Grant 62132011, and Grant 61872209; in part by the Office of Naval Research (ONR) under Grant N00014-18-2893 and Grant N00014-20-1-2407; in part by the National Science Foundation (NSF) under Grant CNS-1815650, Grant 1642129, and Grant 1700544; and in part by the Shuimu Tsinghua Scholar Program. The preliminary version was presented at USENIX Security'19 [1] [DOI: https://dl.acm.org/doi/10.5555/3361338.3361341]. *(Renjie Xie and Jiahao Cao contributed equally to this work.) (Corresponding authors: Mingwei Xu; Qi Li.)*

Renjie Xie, Jiahao Cao, Qi Li, Mingwei Xu, and Yuan Yang are with the Beijing National Research Center for Information Science and Technology, the Department of Computer Science and Technology, and the Institute of Network Science and Technology, Tsinghua University, Beijing 100084, China, and also with the Peng Cheng Laboratory, Shenzhen 518066, China (e-mail: xrj21@mails.tsinghua.edu.cn; caojh2021@mail.tsinghua.edu.cn; qli01@mail.tsinghua.edu.cn; xmw@cernet.edu.cn; yangyuan_thu@mail.tsinghua.edu.cn).

Kun Sun is with the Department of Information Sciences and Technology, George Mason University, Fairfax, VA 22030 USA (e-mail: ksun3@gmu.edu).

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

Digital Object Identifier 10.1109/TNET.2022.3169136

forward control traffic in SDN. For instance, an SDN network with m switches can have $O(m^2)$ links. However, there may be m links forming a spanning tree connecting m switches with a controller to deliver the control traffic. Thus, an attacker must find a target path containing the shared links between control and data traffic to send the attack traffic. However, it is difficult since the network topology and the routing information are invisible to end users. Moreover, none of the information can be inferred by scanning tools used in traditional networks due to different forwarding actions in SDN. For example, Traceroute [27] cannot work since SDN switches usually do not decrease the time-to-live (TTL) values in packet headers.

To address the above challenge, we present a probing technique called *adversarial path reconnaissance* to find a target path that contains the shared links. The key insight is that the delays of control messages on the SDN control channel will become higher if a short-term burst of data traffic passes through the shared links. Hence, the delays can indicate whether the path of the current data traffic has shared links with control traffic. Meanwhile, such delays can be estimated by a host with timing packets. By crafting timing packets to explore the latency variation of the control messages with/without injecting a short-term burst of data traffic, a path containing the shared links can be correctly identified. Therefore, a target path can be finally found by repeating the above reconnaissances on each possible path. We conduct experiments with 261 real network topologies [28] and demonstrate that a target path can be easily found in practice. Moreover, experiments show our reconnaissance can achieve more than 90% accuracy in a real SDN testbed.

In order to ensure the stealthiness of the attack, we leverage the low-rate TCP-targeted DoS [29] to generate data traffic consisting of periodic pulses in the shared links, instead of directly flooding shared links to disrupt the network. The low-rate TCP targeted DoS incurs repeated TCP retransmission timeouts for TCP connections of the control channel. Compared with direct link flooding on the shared links, it significantly reduces the volume of attack traffic. Moreover, our attack is significantly different from the packet-in flooding attacks [15], [30] that trigger a huge volume of *control traffic* with bogus packets to saturate the SDN control channel. Instead, the CrossPath attack leverages low-rate *data traffic* to disrupt the control channel and can thus succeed even in the presence of state-of-the-art SDN defenses, such as FloodGuard [15], FloodDefender [31], and SPHINX [32].

We systematically study the impacts of the attack on different SDN applications that achieve diversified network functionalities. We find that almost all SDN applications can be affected by our attack since our attack disrupts the core services in SDN controllers that support these applications. In order to understand the impacts, we conduct experiments with three typical applications that have been widely deployed in SDN controllers, i.e., Learning Switch [33], Reactive Routing [34], and Load Balancer [35]. The results show: (1) Learning Switch cannot successfully install forwarding decisions in the data plane and thus the throughput of the data plane is reduced to 0 Mbps; (2) Reactive Routing cannot update routing information in time and obtains incorrect topology information, which incurs various routing anomalies, e.g., routing blackhole, routing path eviction, and flow table resetting; and (3) Load Balancer generates wrong decisions, resulting in link overloading.

As the CrossPath attack directly disrupts the SDN control channel, defeating it is challenging. Existing SDN defense systems [15], [31], [32] typically rely on the control messages that are successfully delivered between the controller and switches. However, the CrossPath attack directly congests the control channel before the controller is able to take actions for mitigating the attack in time. Hence, a countermeasure easily fails to work since control messages cannot be successfully delivered. Besides, it is difficult to locate the attack flow from all flows with limited bandwidth. The bandwidth of the control channel between the controller and an SDN switch is limited, which is typically less than 10 Mbps [15], [30], [31]. Locating the attack flow from a large number of flows by querying their flow statistics may consume huge bandwidth and exceed the capability of the control channel. Particularly, the bandwidth consumption increases linearly with the number of flows.

To solve the above challenges, we provide CrossGuard, which is an effective and lightweight defense system against the attack. CrossGuard consists of two modules: protection rule activator and malicious flow locator. Protection rule activator proactively installs protection rules into switches. Once the control channel is congested, protection rules will be automatically activated using timeout mechanisms to ensure enough bandwidth for delivering control messages. Thus, CrossGuard can enable malicious flow locator to take further actions for the attack. Considering the limited bandwidth of the control channel, the locator leverages a bandwidth-saving malicious flow location algorithm to iteratively locate the attack flow instead of querying all flows. We prototype and evaluate CrossGuard in a real SDN testbed. Experiments demonstrate that it can effectively protect the SDN control channel. Moreover, CrossGuard can quickly locate the attack flow with 98% accuracy in about 2 seconds while consuming less than 0.5 Mbps of the control channel bandwidth.

In summary, our paper makes the following contributions:

- We present the CrossPath attack to significantly disrupt the SDN control channel by exploiting the shared links between paths of control traffic and data traffic.
- We develop a probing technique called adversarial path reconnaissance that can find a target path containing the shared links with a high accuracy.
- We perform a systematical study and conduct extensive experiments on typical SDN applications to demonstrate the impacts of the attack on various SDN functionalities.
- We propose an effective and lightweight defense system named CrossGuard to protect the control channel and defeat the CrossPath attack.
- We prototype CrossGuard and evaluate its accuracy, effectiveness, and overhead in a real SDN testbed.

II. Background

Software-Defined Networking (SDN). SDN enables network innovations by decoupling the control and data planes and

provide programmability as well as flexibility. The SDN architecture can be divided into three layers. The control layer and the application layer constitute the control plane, which runs as a network operating system, a.k.a. a controller. Various network applications can be deployed in the application layer to enable diversified network functions, such as routing, anomaly detection, and load balancing. The data plane layer, which consists of "dumb" SDN switches, performs low-level packet processing and forwarding based on the decisions generated by the control layer.

OpenFlow. The dominant communication protocol between the control and data planes is OpenFlow [5], which has been standardized by the Open Networking Foundation (ONF) [36]. OpenFlow allows a controller to specify SDN switches' forwarding behaviors by installing flow rules. Each flow rule contains *match fields* to match against incoming packets, a set of *instructions* that describe how to process the matched packets, and *counters* that count the number and the total bytes of matched packets. Moreover, to reduce the cost of building a dedicated control network and operating networks, OpenFlow allows the control and data traffic to share some links in the network, which is called in-band control [17], [24].

Note that flow rules can be installed in two ways [15], [31], i.e., reactively and proactively. For the reactive approach, flow rules are dynamically installed when flows arrive at SDN switches. The first packet of each flow triggers a packet_in message to controllers. Then, the switches receive the corresponding flow_mod messages to install rules matching and processing the packets. Thus, the network could provide flexible control of each flow. However, frequent flow rule queries consume much bandwidth of control channels and huge computing resources of SDN switches and controllers. Besides, as packets can only be forwarded after the rule has been enforced by the flow_mod messages, the connection delay of flows are increased. For the proactive approach, flow rules are preinstalled before all flows arrive at SDN switches. Thus, flows are forwarded without querying controllers. This approach saves control bandwidth and computing resources. It also leads to a small connection delay for each flow. However, it sacrifices the flexibility of controlling each flow since the controller cannot know the arrival of each flow in time. Consequently, the controller cannot dynamically adjust the forward path of each flow in a fine-grained manner according to real-time network environments.

Low-rate TCP-targeted DoS. It consists of the periodic on-off "square-wave" traffic and congests the TCP flows with a high burst rate. During the burst, the queues of victim switches are filled with LDoS packets, which results in the high packet loss of TCP flows. Therefore, periodic bursts of LDoS will force TCP flows into TCP retransmission timeouts for a long time and the throughput of affected TCP flows can be significantly reduced. Moreover, the average rate of the LDoS attack is not as high as the burst rate, which makes it more stealthy compared to the brute-force DoS attacks.

III. The CrossPath Attack

In this section, we present the CrossPath attack on disrupting the SDN control channel.

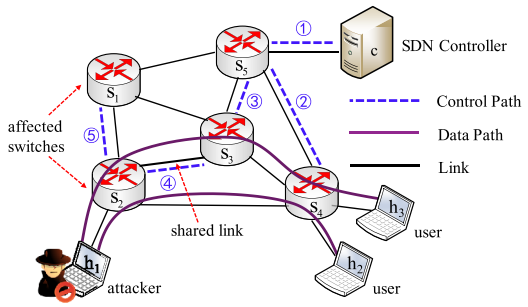


Fig. 1. An example of disrupting the SDN control channel.

A. Threat Model

We consider an SDN network deployed with the OpenFlow protocol. The SDN controller manages switches over an in-band control channel [17], [24]. We do not require the network applying a reactive approach to install flow rules. Instead, it can enforce either reactive or proactive rule installation [15], [31]. We assume that an attacker has or compromises at least one host attached in the network, which can be easily achieved, e.g., by renting a virtual machine in an SDN-based cloud network. The goal of the attacker is to craft data traffic to disrupt the SDN control channel that delivers control traffic.

An attacker does not need to have prior knowledge on the network and any privileges of network operation. The CrossPath attack does not require the attacker to compromise the controllers, applications, and switches, or to construct man-in-the-middle attacks on the control channel to manipulate the control messages. The control channel can be protected with TLS/SSL. Furthermore, we assume that controllers, switches, and applications are well protected. For example, the network applies strict access control policies to prevent communication between controllers and attackers.

B. Overview

The CrossPath attack aims to disrupt the SDN control channel by exploiting the shared links between paths of control traffic and data traffic. An attacker interferes with the transmission of control traffic by generating data traffic passing through the shared links. Thereby, the real-time control messages delivered in the control channel are delayed or dropped. As the SDN controller performs centralized control over all switches via the control channel, the attack can almost break down all network functionalities enabled by SDN. To achieve this, an attacker needs to use a host attached in the network to generate probing traffic so as to identify which path of data traffic (i.e., a target path) shares links with paths of control traffic. Then, the attacker can send attack traffic to the target path to disrupt the control channel. In order to decrease the attack rate, we utilize LDoS to generate attack traffic.

Now let us use a simple example to illustrate the attack. For the ease of explanation, we use *data path* to denote the path where the data traffic is delivered and *control path* to denote the path where the control traffic is delivered. As shown in Figure 1, the network has five switches $\{s_1, s_2, s_3, s_4, s_5\}$.

Host h_1 and h_3 communicate with each other via the data path $h_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow h_3$, while the control path between s_2 and the controller is $s_2 \rightarrow s_3 \rightarrow s_5 \rightarrow c$. We can observe that the link between s_2 and s_3 is shared by the control and data path. Assume host h_1 compromised by an attacker sends crafted LDoS traffic to h_3. Since the link and corresponding queues of switch ports are also used by the control paths of s_2 and s_1, the control messages delivered between the two switches and the SDN controller can be significantly delayed or dropped, resulting in abnormal network behaviors.

In order to successfully launch the attack, an attacker should correctly choose a target path that contains shared links. However, it is challenging to find target paths in SDN. Different from traditional IP networks that almost each link delivers data and control traffic at the same time, there are only a few number of links delivering control traffic in SDN. For instance, given an SDN network with m switches, there may be $m^2/2$ links. m links may be used to deliver control traffic so that the connectivity between the controller and all SDN switches can be ensured. Thus, only a limited number of data paths include the links shared with control paths. To identify such data paths, the attacker needs to know the network topology and routing information. Nevertheless, they are stored in the SDN controller and are invisible to the attacker. Moreover, existing scanning tools cannot be used in SDN to infer the network topology and routing information since SDN has different forwarding behaviors compared to traditional IP networks. For example, Traceroute [27] cannot infer the routing path of the packets, as SDN usually does not decrease the time-to-live (TTL) values in packet headers.

C. Adversarial Path Reconnaissance

To address the challenges above, we develop a probing technique called *adversarial path reconnaissance* to find a target data path that have links shared with control paths. Our key observation is that the delay of a control path is higher if a short-term burst of the data traffic passes through the shared links. Thus, an attacker can use a host in SDN to identify a target path by generating data traffic and measuring the delay variations of the control paths. Specifically, our adversarial path reconnaissance consists of three phases: estimating the delays of control paths, identifying a target data path and improving accuracy with T-test.

Estimating Delays of Control Paths. In SDN, there are some types of packets that are directly forwarded to controller no matter whether the network applies reactive or proactive approaches to install flow rules. They must be processed by the controller since the controller stores the information requested by the packets. Specifically, Address Resolution Protocol (ARP) packets, Dynamic Host Configuration Protocol (DHCP) packets and Border Gateway Protocol (BGP) packets are the representatives. For example, Floodlight offers the *ARP Proxy* [37] application that directly replies an ARP response packet containing the destination MAC address for an ARP request packet. Similarly, the SDN controller directly replies DHCP response packets for a DHCP request packet to automatically configure IP addresses of end hosts [38]. Meanwhile, the SDN controller may communicate with other legacy routers by replying BGP packets [39].

As these packets are forwarded mainly through the control path, an attacker can craft these packets and time related response packets to estimate delays of control paths. No matter whether the network applies reactive or proactive rule installation, the estimation can still be successfully conducted. We show an example of how an ARP packet can be used to estimate the delays of control paths. As is shown in Figure 1, we assume that a host h_1 sends an ARP request packet. When the packet arrives at the ingress switch s_2 through the data path $h_1 \rightarrow s_2$, it will be forwarded to the controller through the control path $s_2 \rightarrow s_3 \rightarrow s_5 \rightarrow c$. Then, an ARP response packet will be forwarded back to the host h_1 through the reversed control path and the reversed data path. Hence, the round-trip time (RTT) between the ARP request packet and the ARP response packet consists of the delay of the data path and the delay of the control path. As the delay of the data path is much smaller than the delay of the control path due to its shorter forwarding path and faster data plane forwarding, the RTT mainly reflects the delay of the control path. For example, our experiments on the real SDN testbed show that the total RTT is about 5 ms and the delay of the one-hop data path is less than 0.5 ms. Thus, the RTT can approximate the delay of the control path.

Identifying a Target Data Path. An attacker needs to send two packet streams for each possible data path in order to find a target data path crossing with some control paths, i.e., a data path containing shared links. The first packet stream is a *timing stream*, which aims to estimate the delay of the control path. The timing stream must contain packets that will be forwarded to the controller and trigger response packets back to the end host. Hence, an attacker compromising a host can calculate the RTT of the packet to estimate the delay of the control path. As we mentioned before, crafting ARP, DHCP, and BGP packets can achieve this goal.

The second packet stream is a *testing stream*. It contains a short-term burst of packets sent to the destination host in the current data path. These packets in the stream can be typically UDP packets. TCP packets can also be chosen if we send them with raw sockets [40] to eliminate the automatic rate control in TCP. The testing stream can be used to test whether the current data path crosses with some control paths or not in collaboration with the testing stream. An attacker can first measure the delay δ by the timing stream without transmitting the testing stream to the destination. Next, an attacker can measure the delay again (denoted by δ') with the testing stream being transmitted at the same time. By comparing these two delays, an attacker can obtain:

(i) If δ' is significantly higher than δ, the short-term burst of packets affects the delays of some control paths. Thus, the data path currently being explored crosses with some control paths.

(ii) If δ' is similar to δ, no available evidence indicates that the data path crosses with some control paths.

A target path can thus be found by testing each path.

Improving Accuracy with *T-test*. Although our reconnaissance allows an attacker to know whether a data path crosses

with control paths by sending timing and testing packets, it may achieve low accuracy in practice. Various network noises can affect the reconnaissance. For example, a burst of benign traffic can cause high delays of control paths, which makes a non-target data path misidentified as a target data path. We find that t-test [41] can be a straightforward approach to eliminate the influences of network noises as much as possible. T-test is a statistical method that compares whether two groups of samples with random noises belong to the same distribution. It produces a p value to denote the likelihood of the two groups of samples from the same distribution. Typically, if p is less than a predetermined value, i.e., the significance level α [41], the two groups are considered significantly different. Thus, we can collect two groups of delays with or without a testing stream for a data path, and apply t-test to determine whether a data path crosses with control paths according to the p value.

Algorithm. Algorithm 1 shows the pseudo-code of the adversarial path reconnaissance, which can be performed by a host in the target network. The input η is the number of repeated reconnaissances for each data path and is also the number of data in each group used in the t-test. The input α is the significance level used in the t-test. Step 1 gets all hosts in the network in order to explore the data paths between the compromised host and them. Here, we can apply traditional scanning tools such as Nmap [42] to scan all hosts in SDN. This is because SDN does not change the traditional network protocols such as ICMP and TCP. Hence, hosts can still be discovered in SDN by crafting ICMP ECHO Request packets or TCP SYN packets to a range of IP addresses. The main loop is from Step 3 to Step 19. In each loop iteration, the algorithm tests one data path. Step 5 to Step 12 collect $2 \cdot \eta$ delays of a control path that may possibly cross with a data path. The delays without sending the testing stream is obtained in Step 6 to Step 7. Step 8 to Step 11 obtain the delays while transmitting the testing stream. After obtaining all the delays, the t-test is applied to determine whether a data path crosses with a control path in Step 13 to Step 17. If the group of delays with testing stream is dramatically higher than the other group, the algorithm outputs the destination host indicating that the current data path is a target path. Otherwise, the algorithm prepares for the next round of iteration in Step 19.

Algorithm 1 Adversarial Path Reconnaissance

Input: $\eta,\ \alpha$
Output: h;
1: $H \leftarrow ScanAllHosts()$
2: $i \leftarrow 0$
3: **while** $i < |H|$ **do**
4: $\delta \leftarrow [\ ],\ \delta' \leftarrow [\ ]$
5: **for** $j = 0 \rightarrow \eta - 1$ **do**
6: $RTT \leftarrow$ sendTimingStreamTo($H[k]$)
7: $\delta.append(RTT)$
8: startSendTestingStreamTo($H[k]$)
9: $RTT' \leftarrow$ sendTimingStreamTo($H[k]$)
10: stopSendTestingStreamTo($H[k]$)
11: $\delta'.append(RTT')$
12: **end for**
13: **if** $tTest(\delta, \delta') < \alpha\ and\ sum(\delta) < sum(\delta')$ **then**
14: /* The data path from the compromised host to $H[k]$ crosses with control paths. */
15: $output(H[k])$
16: $exit()$
17: **end if**
18: $i \leftarrow i + 1$
19: **end while**

IV. Attack Evaluation

The section conducts simulations and real experiments to evaluate the feasibility and effectiveness of the attack.

A. Large-Scale Simulation Experiments

Simulation Setup. We perform simulations with 261 real network topologies [28] around the world. As these network topologies do not contain hosts and routing information, we generate 100 hosts [1] in each topology and apply Dijkstra's algorithm [43] to generate the shortest data path between two hosts. Note that shortest path forwarding is commonly used in the intra-domain routing system. We add another host in each network topology as the SDN controller. The controller can connect switches via shortest paths (SP) to minimize delays, a minimum spanning tree (MST) to minimize costs, or randomly searching available paths (RS). We conduct experiments with different types of connections in turn. Moreover, for simplicity and without loss of generality, we assume that the attacker only controls one host in the network and we attach such a host to each network topology. As the positions of hosts in a network will affect our experimental results, we conduct 1,000 experiments for each network topology and randomly change the positions of all hosts in each experiment. We show the average results over 1,000 experiments for each topology.

Average Percentage of Identified Target Paths. Figure 2a shows the CCDF of the average percentage of identified target paths with 261 various network topologies. From the results, we can see all the network topologies have at least 5% identified target paths among total data paths in a network regardless of types of connections. More than 98% of the network topologies have at least 30% identified target paths. Moreover, the network tends to have more identified data paths when the controller connects switches via MST. The results demonstrate that our reconnaissance can find target data paths in various network topologies.

Average Percentage of Affected Switches. As attacking different target paths will affect the average percentage of switches in a network topology, we randomly attack a target path in the 1,000 experiments for a network topology and calculate the average percentage of affected switches. Figure 2b shows that more than 20% of the switches can be affected by attacking a target path for 90%, 99% and 99% of the 261 network topologies with SP, MST and RS connections, respectively. For some network topologies, attacking a target path can even affect half of the whole switches. Thus, it is

[1]In reality, we also conduct our experiments with 50, 500, 1000 hosts, respectively. The results are similar to those in Figure 2.

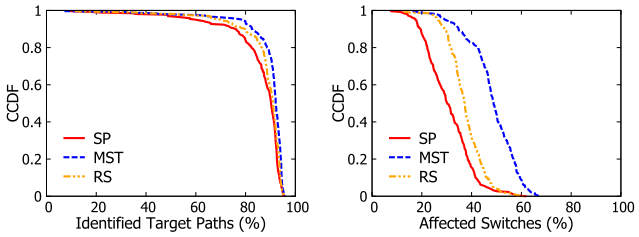


Fig. 2. Complementary Cumulative Distribution Function (CCDF). (a) shows the CCDF of the average percentage of identified target paths with 261 real topologies; (b) shows the CCDF of the average percentage of affected switches by attacking a target path with 261 real topologies.

possible for an attacker to attack multiple target paths to cause damages for the whole switches and incur network-wide DoS.

B. Experiments in a Real SDN Testbed

Experiment Setup. Our testbed contains a popular SDN controller Floodlight [44], five hardware SDN switches (AS4610-54T [45]), and three physical hosts. The controller is deployed on a server with a quad-core Intel Xeon CPU E5504 and 32GB RAM. Each physical host has a quad-core Intel i3 CPU and 4GB RAM. The network topologies, control paths and data paths are illustrated in Figure 1. An attacker first compromises host h_1 to conduct Algorithm 1 for the data paths of the other hosts. The burst rate of short-term testing packets is 1 Gbps, which is the maximal rate the host can send.

The attacker then generates LDoS data traffic to disrupt the control channels of switches s_1 and s_2 by attacking the data path between h_1 and h_3. Basically, there are three parameters for the LDoS flows: burst length, inter-burst period, and peak magnitude. The previous study [46] has conducted comprehensive experiments on how different parameters determine the attack impacts of LDoS flows and how to choose these parameters. As our paper mainly focuses on studying the impacts for the SDN functionalities after the control channel is attacked by the data traffic, we apply fixed parameters in our attack. We choose the burst length as 100 ms, inter-burst period as 200 ms, and peak magnitude as the maximal speed 1 Gbps that the host can send for our all experiments in the paper. These parameters show how an attacker can affect the SDN functionalities to the maximum extent by generating data traffic to disrupt the control channel. Moreover, compared to simply flooding the target paths, which needs to send traffic with 1 Gbps all the time, the rate of our LDoS flow is only about 0.33 Gbps on average.

Accuracy of Reconnaissances. We first collect the delay variations on delivering control messages. The delay variation is defined as the absolute difference between the delays of control messages measured with and without testing stream. We collect 5,000 records both for two data paths in the network. Figure 3 shows the distribution of the probability of the delay variation. The results demonstrate that the target data path has a significantly different probability distribution compared with the non-target data path. In particular, most delay variations with the non-target data path are less

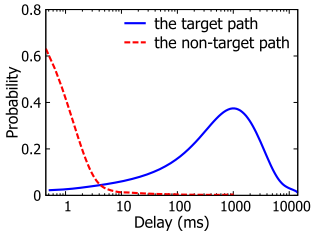


Fig. 3. Probability distribution of delay variations.

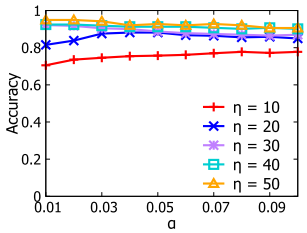


Fig. 4. Accuracy of the reconnaissance algorithm.

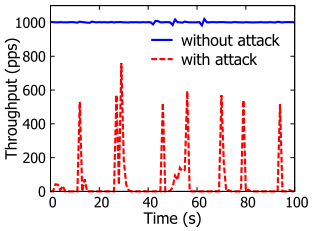


Fig. 5. Throughput of control packets.

than 2 ms, while most delay variations are much larger for the target data path. These results illustrate that the discrimination between target data paths and non-target data paths can be easily identified according to the delay variations.

We then calculate the accuracy of our reconnaissance by conducting 1,000 repeated experiments with different settings of η and α. Here, η denotes the number of measured delays for each data path, which is also the size of each group in the t-test used to identify a target path. α is the significance level used in the t-test. As shown in Figure 4, the accuracy increases with the increase of η. Moreover, we can observe that the accuracy increases with the increase of α when η is smaller, e.g., 10 or 20. However, the accuracy tends to be stable when η becomes large. The reason is that two different groups will statistically different from each other and two similar groups will be statistically closer to each other with more data. It is easier to distinguish the two types of paths if we have enough data, which is not significantly impacted by the setting of α. The accuracy always reaches more than 90% with different settings of α when η is 40 or 50.

Effectiveness of the Attack. To evaluate the impact of the attack on the control packets, we configure the controller to generate 1,000 control packets per second [2] to the switch s_2. Figure 5 shows the throughput of control packets. The throughput can achieve 1,000 packets per second. However, it almost

[2]There can be thousands of control packets per second [47]. For simplicity but without loss of generality, we choose a practical value, i.e., 1,000.

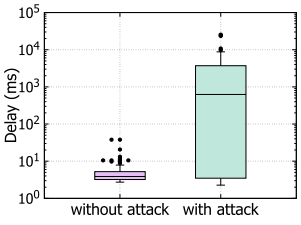
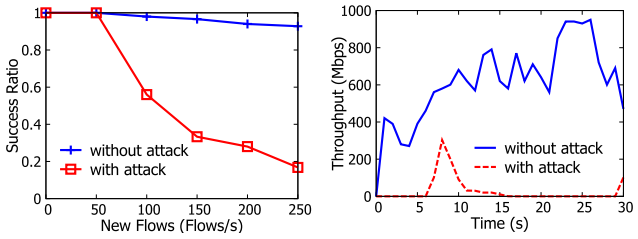


Fig. 6. Delay of control packets.



(a) Success ratio of rule installation for a switch.

(b) Throughput for a switch with 250 flows/s.

Fig. 7. Attack impacts on learning switch.

drops to 0 under the attack though there are short-term peaks of throughput. The reason is that our attack triggers TCP of control flows to periodically enter the phase of retransmission timeouts. In this case, no packets will be sent within the retransmission timeouts. Figure 6 shows the delay of control packets. The median value of delays for control packets under the attack is 687 ms, which is more than about 100 times higher than that in absence of the attack. Moreover, the delays under the attack vary within a large range from below 10 ms and to more than 10,000 ms. However, most delays without the attack are less than 10 ms. The results above demonstrate our attack can significantly degrade the throughput of control packets and incur high delays.

V. Attack Impacts on Network Functionalities

In this section, we perform a systematical study on the attack impact on various network functionalities.

A. Core Services of SDN

SDN controllers can be abstracted as a two-layer architecture though different controllers have different implementations. Applications can be deployed in the top layer to enable different network functionalities, while the low layer provides different core services that interact with switches and provide basic functionalities for the top-tier applications. There are four major core services:

Packet Service. The service manages packets exchanged between the control and data planes. It paraphrases $packet\_in$ messages containing data packets received from switches and dispatches them to applications. Meanwhile, it sends data packets back to switches via $packet\_out$ messages.

Flow Rule Service. The service manages flow rules. It installs or updates rules in switches via $flow\_mod$ messages according to the results computed by applications.

Topology Service. The service maintains the topology of end hosts, links, and switches. It discoveries new hosts and tracks their locations via $packet\_in$ messages embedded with an ARP or DHCP payload. It periodically sends and receives LLDP packets encapsulated in $packet\_in$ or $packet\_out$ messages to maintain link information. Besides, it establishes the control channel between switches and controllers via several $handshake$ messages. The liveness of switches is periodically checked via $echo\_request$ and $echo\_reply$ messages.

Flow Metrics Service. The subsystem is responsible for collecting flow statistics. It periodically queries the flows on network devices via $stats\_request$ and $stats\_reply$ messages, and then provides various statistics to applications.

We note that almost all applications enabling network functionalities in SDN is built on at least one of the four services. Our attack thus can affect various SDN functionalities by disrupting the transmission of control messages exchanged between these core services and switches. We will choose three typical applications that are widely deployed in SDN controllers to show the impacts of the attack on various network functionalities. The implementations of the three applications [33]–[35] are from Floodlight [44].

B. Learning Switch

The *learning switch* application [33] allows SDN switches act as normal switches in legacy networks. The application examines a packet matching no rules in a switch and looks up the recorded mapping between the source MAC address and the port. If the destination MAC address has already been associated with a port, the packet will be sent to the port and corresponding rules will be installed to match subsequent packets. Otherwise, the packet will be flooded on all ports. The application relies on two services. The packet service sends the packet to the controller via $packet\_in$ messages and back to the switch via $packet\_out$ messages, and the flow rule service installs rules in the switch via $flow\_mod$ messages.

Our attack can effectively block installation of forwarding decisions generated by the application by disturbing the messages exchanged between the core services and switches. Figure 7 shows the impacts of the attack on the functionalities of *learning switch*. Here, we define the success ratio of rule installation as the number of successfully installed rules over the number of rule requests within a second. As shown in Figure 7a, the success ratio of rule installation in a switch always maintains over 90% with various numbers of new flows without our attack. However, it drops significantly in presence of our attack. When the rate of new flows reaches 250 flows/s, the success ratio reduces to below 20%. Thus, *learning switch* cannot work correctly. As shown in Figure 7b, the throughput of a switch is 0 Mbps for a long time under attack when there are 250 flows/s.

C. Reactive Routing

The *reactive routing* [34] application enables flexible and fine-grained routing decisions for different flows. When a new flow matching no rules is generated, the first packet of the flow will be sent to the *reactive routing* application. The application analyzes the packet and calculates routing paths

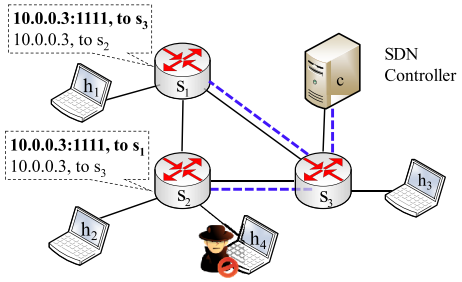
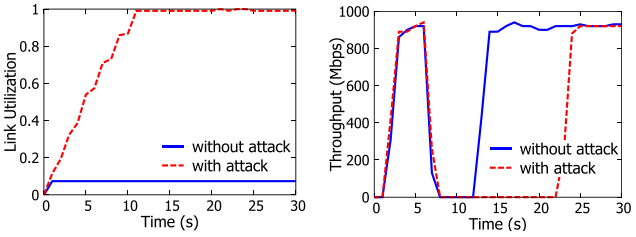


Fig. 8. The network topology used in reactive routing.



(a) Increasing link utilization due to long-term routing rule inconsistency.

(b) Long-term routing blackhole due to delayed messages when a host is migrated.

```
17:37:46.344 INFO [n.f.t.TopologyInstance] Route [id=RouteId [src=1c:48:cc:37:ab:a0:a8:41
dst=9d:54:cc:37:ab:a0:a8:41], switchPorts=[[id=1c:48:cc:37:ab:a0:a8:41, port=37],
[id=9d:54:cc:37:ab:a0:a8:41, port=31]]]
17:38:01.62 INFO [n.f.l.i.LinkDiscoveryManager] Inter-switch link removed: Link
[src=a4:e7:cc:37:ab:a0:a8:41 outPort=38, dst=9d:54:cc:37:ab:a0:a8:41, inPort=42, latency=6]
17:38:01.95 INFO [n.f.t.TopologyManager] Recomputing topology due to: link-discovery-
updates
17:38:01.345 INFO [n.f.t.TopologyInstance] Route [id=RouteId [src=1c:48:cc:37:ab:a0:a8:41
dst=9d:54:cc:37:ab:a0:a8:41], switchPorts=[[id=1c:48:cc:37:ab:a0:a8:41, port=32],
[id=a4:e7:cc:37:ab:a0:a8:41, port=36], [id=a4:e7:cc:37:ab:a0:a8:41, port=38],
[id=9d:54:cc:37:ab:a0:a8:41, port=42]]]
```

(c) Eviction of a routing path due to a deactivated link.

```
20:55:25.510 INFO[n.f.c.i.OFChannelHandler] [[1c:48:cc:37:ab:a0:a8:41(0x0) from
192.168.100.1:59413]] Disconnected connection
20:55:26.218 INFO[n.f.c.i.OFChannelHandler] New switch connection from
/192.168.100.1:59414
20:55:27.755 INFO[n.f.c.i.OFChannelHandler] Negotiated down to switch OpenFlow version
of OF_14 for /192.168.100.1:59414 using lesser hello header algorithm.
20:55:31.698 INFO[n.f.c.i.OFSwitchHandshakeHandler] Clearing flow tables of
1c:48:cc:37:ab:a0:a8:41 on upcoming transition to MASTER.
```

(d) Cleaning of flow tables due to the reset of a switch.

Fig. 9. Attack impacts on reactive routing.

for the new flow. Besides depending on the packet service processing data packets and flow rule service installing rules, the application also queries the topology service that provides the information of the locations of hosts, the state of switches and links.

In order to demonstrate the effectiveness of our attack, we build a network topology with four hosts and three switches, as shown in Figure 8. The IP addresses of the four hosts h_1, h_2, h_3 and h_4 are 10.0.0.1, 10.0.0.2, 10.0.0.3, and 10.0.0.4, respectively. The hosts h_1 and h_2 send packets to the host h_3. The default routing path of packets from h_1 to h_3 is $< l_{h_1 \to s_1}, l_{s_1 \to s_2}, l_{s_2 \to s_3}, l_{s_3 \to h_3} >$. The default routing path of packets from h_2 to h_3 is $< l_{h_2 \to s_2}, l_{s_2 \to s_3}, l_{s_3 \to h_3} >$. Also, a flow with TCP port 1111 from h_2 to h_3 has a different path due to a QoS requirement. The compromised host h_4 sends attack traffic to h_3 and hence exploits the control path of switch s_2.

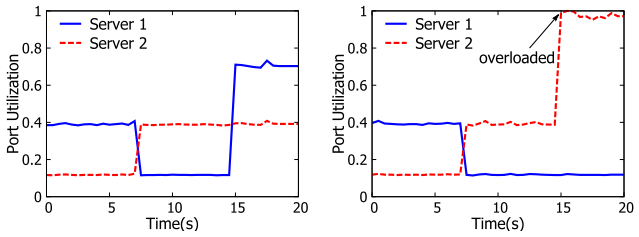
As shown in Figure 9a, our attack incurs long-term routing rule inconsistency, which makes the link utilization reach 100%. The reason is that SDN exists transient rule inconsistency [48] which can be leveraged by our attack. In the network shown in Figure 8, packets with an IP destination address 10.0.0.3 and a destination port 1111 loop between s_1 and s_2 when the application deletes rule "$10.0.0.3 : 1111, to\ s_3''$ while rule "$10.0.0.3 : 1111, to\ s_1''$ remains. The rule inconsistency normally lasts for a very short period before all the commands of deleting corresponding rules of the flow are issued. However, our attack can delay the commands exchanged between the flow rule service and s_2 for tens of seconds. Thus, the packets loop between s_1 and s_2 for a long period and the link utilization between the two switches increases with more packets injected.

Figure 9b shows the long-term routing blackhole when h_3 is migrated from s_3 to s_2. The migration is finished within five seconds without the attack, as the topology service can track the new location via $packet\_in$ messages containing the DHCP payload when the host moves to s_2. However, the messages are significantly delayed under our attack, and thereby the routing between other hosts and h_3 cannot be updated in time, causing more than 10 seconds routing blackhole. Moreover, by blocking LLDP packets between the topology service and switches, our attack can deactivate links in the topology database and thus the corresponding routing paths will be removed. In the Floodlight controller, a link will be deactivated if no LLDP packets pass through the links within 35s. Figure 9c shows the original routing path from h_2 to h_3 is removed since our attack deactivates the link from s_2 to s_3. Moreover, our attack can reset the connections between switches and the controller by delaying control messages. Figure 9d shows the connection of switch s_2 is reset and all the flow tables are cleaned.

D. Load Balancer

Load balancing has been widely used to improve resource usage and throughput as well as reduce response delays, which balances the workload among multiple nodes. SDN controllers deploy the *load balancer* [35] application to achieve the goal. The application in the Floodlight controller can balance requests of clients in two ways, i.e., round robin and statistics-based scheduling. Round robin scheduling randomly chooses a server from a server pool to serve a new request. The statistics-based scheduling chooses a server with the lowest utilization to serve a new request, where the utilization is calculated according to the real-time statistics of the switch ports. It relies on the flow metrics service to collect the statistics.

We configure the application to enable statistics-based scheduling, as it can provide better load balancing under different flow distribution of clients. In our experiments, two hosts consist of a server pool and another two hosts send flows to the servers. Figure 10a shows the utilization of switch ports connecting the two servers over time without our attack. Initially, two different elephant flows are sent to the servers, which causes the port utilization to increase to 40% and 10%, respectively. At the 7th second, the rate of the



(a) Port utilization of servers without attack.

(b) Port utilization of servers with attack.

Fig. 10. Attack impacts on load balancer for misallocating the workloads across servers.

two flows exchanges. The utilization of one server reduces from 40% to 10% while another server increases from 10% to 40%. At the 14th second, a new elephant flow starts, and the application directs the flow to server #1 that has the lowest port utilization. The port utilization of server #1 reaches 70%. Unfortunately, the application will mistakenly direct the flow to server #2 under our attack. As shown in Figure 10b, the port utilization of server #2 reaches 100%. The reason is that our attack can significantly delay the $stats\_request$ and $stats\_reply$ messages exchanged between the flow metric service and switches, and thus the application cannot know the port utilization in time. Actually, it considers that the port utilization of server #2 is still 10% when the new flow comes.

VI. CrossGuard

In this section, we provide an effective and lightweight defense system named CrossGuard to defeat the attack and demonstrate its effectiveness with real experiment results.

A. System Architecture

Figure 11 shows the architecture of CrossGuard. CrossGuard introduces two modules to defend against the attack: protection rule activator and malicious flow locator. Protection rule activator installs specific protection rules into switches to protect control channels. Once the control channel is congested, protection rules will be automatically activated using timeout mechanisms to ensure enough bandwidth for delivering control flows. Meanwhile, as the attack still occupies the valuable bandwidth and may congest data flows, malicious flow locator locates the attack flow from all flows based on flow statistics. Considering the limited bandwidth of the control channel, it leverages a bandwidth-saving malicious flow location algorithm to iteratively locate the attack flow rather than queries all flow statistics. We will detail the two modules of CrossGuard in the following sections.

B. Protection Rule Activator

As the control messages from the controller to switches may be interrupted by the attack, the protection rule activator module installs protection rules into switches in advance to ensure enough bandwidth for delivering control messages. Specifically, protection rule activator installs three crafted rules in the protection table. The first rule matches the control flow and persistently exists. The second rule matches data flows with a

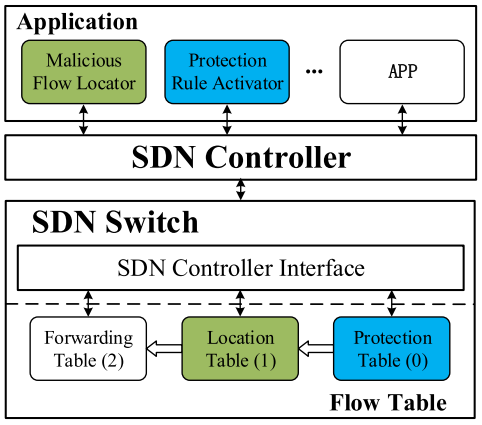


Fig. 11. Architecture of CrossGuard.

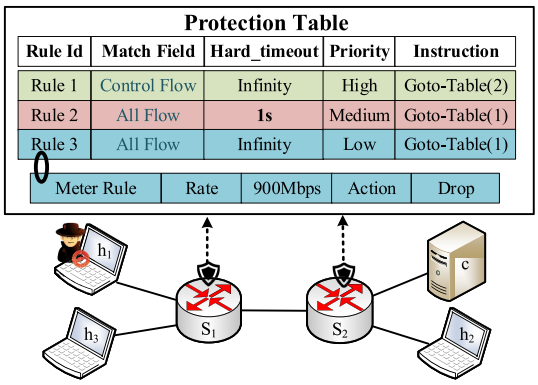


Fig. 12. An example on our protection rule activator.

high priority and doesn't limit the rate of data flows. However, it has a timeout and hence is periodically refreshed by the controller. By default, all data flows are always forwarded through this rule. The third rule also matches data flows but with a low priority and rate limiting. It persistently exists with no timeout settings. When the attack congests the control channel, the second rule cannot be periodically refreshed by the controller. Hence, it disappears due to timeouts. Data flows are then automatically forwarded by the third rule in a rate-limiting manner without the controller involved. In this way, protection rule activator ensures enough bandwidth for the control channel when attack exists, which enables taking following measures that depends on control channels, such as identifying malicious flows.

We take an example to illustrate our mechanism. As shown in Figure 12, the attack flow and the control flow both goes through port 1 in s_1. The attack flow can quickly congest the control flow. Before we have enough time to detect the attack and take mitigation measures, the switch may have already been disconnected from the controller. Any subsequent countermeasures relying on control messages between the switch and the controller cannot work. To prevent this issue, protection rule activator installs three specific rules for flows going through port 1 in the protection table. Rule 1 matches the control flow with high priority. Rule 2 and rule 3 matches all the flows with medium priority and low priority, respectively. However, rule 2 is set with a hard_timeout of 1 seconds.

Hence, we force the controller to periodically issue a *flow_mod* message that refreshes rule 2. Moreover, a meter rule is attached to rule 3, which limits the rate of flows matching rule 3. By default, all data flows including attack flows are forwarded through rule 2 without rate limiting. However, once attack flows remarkably congest and disrupt the control channel, rule 2 will be automatically removed since rule 2 cannot periodically receive *flow_mod* messages from controllers. Thus, all data flows including attack flows are then forwarded through rule 3 with rate limiting. Remaining bandwidth can be totally used by control flows and the malicious flow cannot congest the control channel.

Note that setting a large hard_timeout for rule 2 may result in a long downtime. This is because rule 2 cannot be timely reinstalled to eliminate the effect of the rate limiting for rule 3, which affects the throughput of normal flows. On the contrary, setting a small hard_timeout requires more bandwidth to frequently reinstall rule 2. However, according to our experiments, even we set the minimal configurable hard_timeout, i.e., 1s specified by the OpenFlow protocol, the consumed bandwidth is only several Kbps, which is small. Hence, we set the hard_timeout for rule 2 as 1s to reduce the downtime as much as possible.

It is possible that the allocated bandwidth may be too much or too little for control flows at different time. Hence, we may waste valuable bandwidth or affect control flow throughput which network functionalities depends on. In order to meet the requirement of control flows, protection rule activator predicts control flow throughput through machine learning and dynamically adjusts the bandwidth according to the prediction. Specifically, we predict the control flow bandwidth with the random forest regression model [49], which is one of the most effective machine learning models for regression and is lightweight. We extract the following features from control flows to train our model:

- The sequence of historical throughput of control flows η_n. It directly reflects the possible trend of throughput. We periodically collect the throughput according to the counters of protection rules every interval T_p.
- The sequence of latency increase of control flows ΔRTT_n. If the remaining bandwidth is insufficient for control flows, ΔRTT_n will increase. We periodically collect the latency increase by measuring the time difference between a pair of $echo\_reply$ and $echo\_request$ messages every interval T_p.
- The sequence of the packet_in increase ΔP_n. The bigger ΔP_n is, the more bandwidth is possibly needed in the future. We periodically collect the number of packet_in messages by counting the number of received messages in the controller every interval T_p.
- The sequence of the congestion window of the TCP connection of control flows $cwnd_n$. We periodically fetch the congestion window from the network stack of the SDN controller every interval T_p.

Each feature sequence has the same length of n. Note that n and T_p are parameters. We will choose the best values of them according to our experiments in Section VI-D.

C. Malicious Flow Locator

The protection rule activator module protects control flows from being congested. However, an attack can still consume the bandwidth of links and congest data flows. Thus, we develop a malicious flow locator module for CrossGuard, which identifies and throttles the attack flows based on the statistics of the flows. The malicious flow locator module is activated by the protection rule activator module and analyzes data flows sharing the control path. However, it is challenging to identify the malicious flow from all data flows considering the limited bandwidth of the control channel between the controller and a switch. Assuming that a lot of flows pass a switch and one of them is the malicious flow. The corresponding forwarding rules hence are installed into the switch to forward the flows. If we attempt to check whether a flow is malicious based on flow statistics, we need to frequently query the counters of all the flow rules. However, as the bandwidth of the control channel is typically less than 10 Mbps [31], we cannot afford to query all flow statistics. Particularly, frequently querying all flow rules may incur the congestion of the control flow.

To solve the problem, we exploit the idea of divide and conquer to iteratively locate the attack flow with limited bandwidth. At the beginning, we treat all the data flows as a set, split it into several subsets, and generate a location rule to match each subset. As the number of location rules is much less than that of forwarding rules, it consumes a little bandwidth of the control channel to collect statistics from location rules. Meanwhile, it finds out the subset that is most likely to contain the malicious flow, which is then treated as a new set. We change the match fields of location rules to divide the new set into several subsets. We repeat the above process until a subset contains only the malicious flow. Here, we select the subset with the maximum D-value [50] as the subset most likely containing the malicious flow in each location round. The previous study [50] has shown the metric can well character the singularity and bursty of network traffic under LDoS attacks, and the D-value of the subset containing the malicious flow is the highest.

Figure 13 shows an example of locating malicious flows. For simplicity, we assume that there are 256 flows passing a switch and their source IP addresses belong to the subnet 10.0.0.0/24. One of them is malicious and its source IP is 10.0.0.193. Consider we apply four rules to conduct malicious flow location. In the first round, we split the flows into four subnets, i.e., 10.0.0.0/26, 10.0.0.64/26, 10.0.0.128/26, and 10.0.0.192/26. The four location rules are used to collect the aggregated statistics of each subset. Next, our method finds that the malicious flow belongs to the subnet 10.0.0.192/26. In the second round, the location rules are changed to match four subnets of 10.0.0.192/26. Our method confirms that the subnet 10.0.0.192/28 contains the malicious flow. In the third round, location rules are changed again to detect the four subnets of 10.0.0.192/28 and show that the subnet containing the malicious flow is 10.0.0.192/30. Finally, the source IP of the malicious flow is confirmed to be 10.0.0.193 from 10.0.0.192/30 in the fourth round.

Algorithm 2 Bandwidth-Saving Malicious Flow Location

Input: S, m, l, T_l
Output: M;
1: **while** $|S| > 1$ **do**
2: $SS \leftarrow split(S, m)$
3: **for** $i = 1 \rightarrow l$ **do**
4: **for** $j = 1 \rightarrow m$ **do**
5: $d \leftarrow CalAggBandwidth(SS[j])$
6: $v[j].add(d)$
7: **end for**
8: $sleep(T_l)$
9: **end for**
10: $D \leftarrow 0$
11: $index \leftarrow 0$
12: **for** $i = 1 \rightarrow m$ **do**
13: **if** $D > CalDval(v[i])$ **then**
14: $index \leftarrow i$
15: $D \leftarrow CalDval(v[i])$
16: **end if**
17: **end for**
18: $S \leftarrow SS[index]$
19: **end while**
20: $output(S)$

Algorithm 2 shows the pseudo-code of our bandwidth-saving malicious flow location. The input S is the set of flows to be detected, m is the number of location rules, l is the sequence length of flow statistics, and T_l is the query interval. The main loop is from Step 1 to Step 19. In each loop iteration, the algorithm splits the set of flows into several subsets and finds out which subset contains the malicious flow through D-value. Step 2 split S into m subsets of flows. Step 3 to step 9 collect the sequences of aggregated flow statistics of each subset for l times with a interval T_l. Step 10 to step 17 calculate the D-value of each subset and exploit $index$ to record the subset having the maximum D-value. Step 18 assigns $SS[index]$ to S for the next loop iteration. The algorithm stops when the S only contains the malicious flow. Though the algorithm focuses on mitigating one malicious flow, it can also mitigate multiple concurrent malicious flows with minor modification. Specifically, as the attack flows have much higher D-values than benign flows [50], we can add a threshold α to locate multiple malicious flows, i.e., flows with D-values higher than the threshold are classified as malicious flows. Hence, when choosing the subset in step 10 to step 17, we can record multiple subsets with D-values higher than the threshold α to locate multiple concurrent malicious flows.

D. Evaluation

Experiment Setup. The topology of our experiments is shown in Figure 1. We inject the CAIDA traffic [51] into our network as background traffic. We randomly choose flows from the traffic and ensure that the number of rules generated by flows does not exceed the table capacity. We launch the CrossPath attack using the same configurations in Section IV-B.

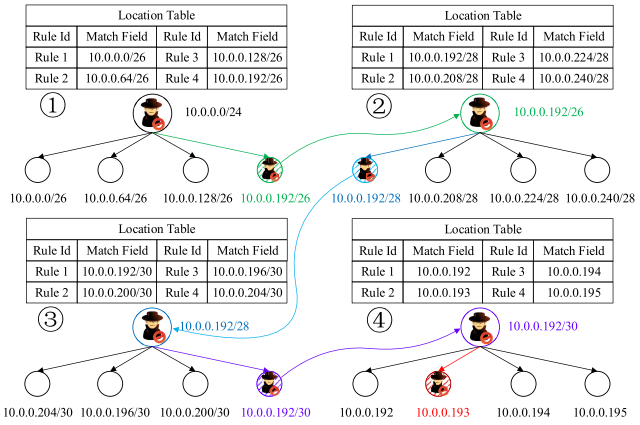
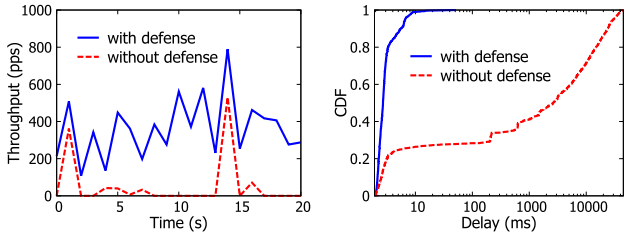


Fig. 13. An example of locating malicious flows. It iteratively locates the attack flow through divide and conquer to save the valuable bandwidth of the control channel.



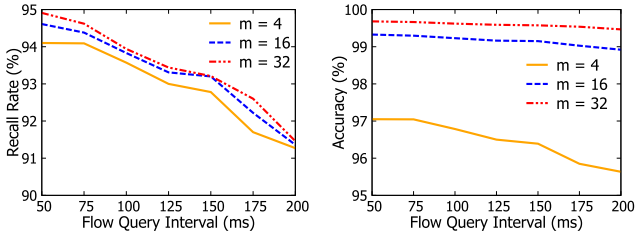
(a) Throughput of control traffic (b) CDF of control delays

Fig. 14. Effectiveness on protecting control traffic.

Effectiveness on Protecting Control Flows. Figure 14a shows the throughput of control packets with and without the defense. As we can see, the attack causes the throughput of control flows into 0 for a long time without CrossGuard. However, CrossGuard can effectively maintain high throughput of control flows even with the attack. Figure 14b shows the delay of control packets with and without our defense. The delays of more than 99% control packets are less than 10 ms with CrossGuard under the attack. However, the delays of almost 70% control packets are more than 100 ms under the attack without CrossGuard. These results demonstrate that CrossGuard can effectively protect control flows.

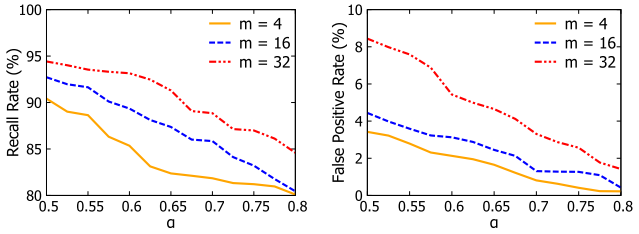
Accuracy on Malicious Flows Location. As the granularity of the aggregated flow statistics can affect the accuracy of malicious flow location, we conduct experiments with different flow query interval T_l ranging from 50 ms to 200 ms and different number of location rules m ranging from 4 to 32. As Figure 15 shows, the recall rate and accuracy decrease when T_l increases. As the controller will obtain less information about the flows with larger T_l, it is more difficult to find out the malicious flow. Meanwhile, the benign flows are easier to be misclassified. Besides, the recall rate and accuracy increase when the number of location rules m increases. It is because a location rule will match less flows when m increases. The aggregated flow statistics of each rule is more fine-grained, which is more accurate for our method to infer the subset containing the malicious flow.

We also conduct experiments to evaluate the accuracy on locating multiple concurrent malicious flows with different



(a) Recall rate (b) Accuracy

Fig. 15. Accuracy of malicious flow location.



(a) Recall rate (b) False Positive rate

Fig. 16. Accuracy of locating multiple malicious flows.

threshold α. As shown in Figure 16, the recall rate and false positive rate decrease when α increases. More malicious flows escape from our detection and less benign flows are misidentified with the increase of α. Meanwhile, the recall rate and false positive rate increase when the number of location rule m increases. This is because the aggregated flow statistics of each rule can be more fine-grained with more location rules. Consequently, it will be more accurate to infer the subsets containing malicious flows. We can see that more than 90% recall rate and less than 4% false positive rate can be achieved when $\alpha = 0.5$ and $m = 4$.

Precision of Bandwidth Prediction. To collect data for training the random forest regression model that predicts bandwidth, we collect 5,000 sequences of features with different collection interval T_p and sequence length n. We calculate Mean Squared Error (MSE) to evaluate the precision of bandwidth prediction and thus choose the best parameters. Our experiments show that $n = 30$ and $T_p = 0.5$s achieve the minimal MSE for our model. Hence, we set the two values in our model and show the detailed bandwidth estimation in Figure 17. As we can see, our method achieves good results on predicting the bandwidth of control flows.

Malicious Flow Location Time. Figure 18 shows the location time on locating the malicious flow with different number of location rules and flows. Here, we ensure that a specific number of flows (i.e., 500, 1,000, 2,000) pass a switch and one of them is malicious in our experiments. As we can see, it costs less location time to locate the malicious flow when the number of location rules increases or the number of flows decreases. The location time is less than 3s with 12 location rules even though there are 2,000 flows in switches.

System Overhead. To evaluate the overhead of CrossGuard, we evaluate its bandwidth consumption and CPU utilization with different flow query intervals T_l ranging from 0.05s to 0.2s, and different location rules ranging from 2 to 32.

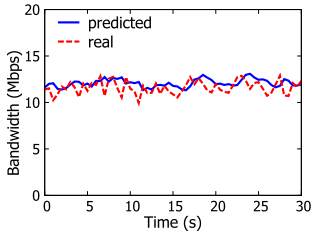


Fig. 17. Predicted and real bandwidth of control flows.

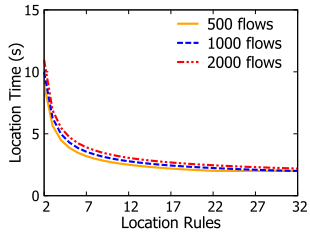
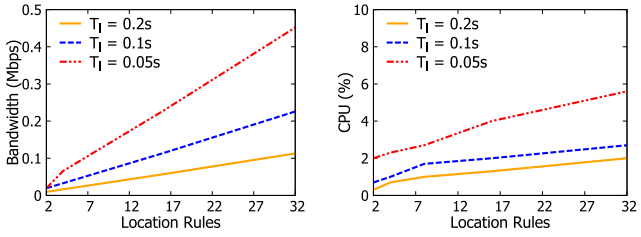


Fig. 18. Location time with different location rules.



(a) Bandwidth Consumption. (b) CPU Utilization.

Fig. 19. CrossGuard overhead.

As shown in Figure 19a, the bandwidth consumption increases with the decrease of the query interval and the increase of location rules. However, the bandwidth consumption is limited. For example, it achieves less than 0.5 Mbps even with 0.05s query interval and 32 location rules. Figure 19b shows the CPU utilization of the controller. A low T_l and a high number of location rules can increase the CPU utilization for the controller. If T_l is set to 0.1s, the CrossGuard introduces an additional CPU usage of 3%. The results show that CrossGuard introduces small overhead to defeat the CrossPath attack.

VII. Related Work

Reconnaissances in SDN. SDN reconnaissances has been extensively studied. Shin and Gu [52] designed an SDN scanner to determine whether a network is SDN by measuring response delays of pings. Cui *et al.* [20] further conducted experiments in real SDN testbed to demonstrate its feasibility. Klöti *et al.* [53] presented a reconnaissance technique to determine if an SDN has rules for aggregated TCP flows by timing the TCP setup time. Achleitner *et al.* [22] designed SDNMap to reconstruct composition of flow rules by analyzing probing packets with specific protocols. Liu *et al.* [23] developed a Markov model to reveal rule distribution among switches. Sonchack *et al.* [21] presented a sophisticated inference attack to learn host communication patterns and ACL

entries even if injected packets do not trigger replies. However, none of the methods can find target paths containing shared links with control paths.

Attacks on SDN and Related Defenses. SE-Floodlight [14] and SDNShield [13] are developed to provide permission control for malicious SDN applications. Some studies focus on the security of controllers, including network poisoning [54], identifier binding attacks [55], subverting SDN controllers [16], and exploiting harmful race conditions in SDN controllers [17]. Other studies focus on data plane security, including low-rate flow table overflow attacks [18], SDN teleportation, and detection on abnormal data plane [19]. Our paper focuses on the security of control channel, which is orthogonal to the existing work. We uncover a new type of attack, which has not been discovered by existing automatic attack discovery tools [56]–[58] in SDN.

The packet_in flooding attack [15], [30] is mostly closest to ours. It saturates the control channel with a large amount of packet_in messages. To trigger the control messages, the attack requires generating massive bogus packets matching no rules in switches. Different from it, our attack generates low-rate data traffic to implicitly disrupt control traffic in the shared links instead of directly generating massive control traffic. Our attack can bypass the previous defenses [15], [30]–[32] against packet_in flooding attacks since they detect attacks by identifying and throttling malicious control traffic.

LDoS Attacks and Defenses in Traditional IP Networks. Kuzmanovic and Knightly [29] developed low-rate TCP-targeted DoS attacks to disrupt TCP connections. Zhang *et al.* [59] demonstrated the attack has severe impact on the Border Gateway Protocol (BGP) by conducting real experiments. Schuchard *et al.* [60] extended the attack and designed the Coordinated Cross Plane Session Termination attack (CXPST) that allows an attacker to disrupt the Internet control plane by using only data traffic. Our attack differs from the previous work in three aspects. First, our attack focuses on disrupting the SDN control channel that shares a limited number of links with data paths. Second, probing techniques are required in the attack to identify target data paths containing shared links, which is necessary to ensure the effectiveness of the attack. Third, our attack in SDN has more significant impacts on diversified network functionalities including layer 2, 3 and 4 functions.

To defend against LDoS, some countermeasures have been provided in traditional IP networks, such as randomizing RTO [46] and complex signal analysis [61]–[66]. However, randomizing RTO cannot fully mitigate the attack [59], and none of the methods are shown to be sufficiently accurate and scalable for deployment in real networks. Besides, they are general defenses against LDoS in traditional IP networks and are not designed to protect the SDN control channel. Defenses against LDoS attacks on BGP was described in [60], such as BGP Graceful Restart. However, it is not suitable to protect the SDN control channel with "dumb" SDN switches.

Link Flooding Attacks and Defense in Traditional IP Networks. Studer and Perrig [67] and Kang *et al.* [68] introduced link flooding attacks, which generate large-scale legitimate low-speed flows to flood and congest network critical links. They use traceroute to find critical links in traditional IP networks. Our CrossPath attack also congests the critical links that deliver control traffic and data traffic in SDN at the same time. However, one major difference is that our CrossPath attack identifies the critical links with the unique SDN reconnaissance technique. Moreover, the CrossPath attack can incur various damages in the whole network by disrupting the control channel due to the centralized control in SDN. Though there exist some SDN defense systems [69]–[71] that detect link flooding attacks, they cannot defend the CrossPath attack that disrupts the control channel they depend on.

VIII. Conclusion

In this paper, we present a novel attack in SDN. It disrupts the control channel by crafting data traffic to implicitly interfere with control traffic in the shared links. We develop the adversarial path reconnaissance to find a target data path containing shared links for the attack. Both theoretic analysis and experimental results show that our reconnaissance works in real networks. We demonstrate that the attack can significantly disrupt various network functionalities in SDN. To defeat the attack, we design and prototype a defense system named CrossGuard. Our experiments show it can effectively defeat the attack while introducing small overhead.

References

[1] J. Cao *et al.*, "The crosspath attack: Disrupting the SDN control channel via shared links," in *Proc. USENIX Secur. Symp.*, 2019, pp. 19–36.

[2] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined WAN," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[3] (2018). *Microsoft Azure and Software Defined Networking*. [Online]. Available: https://docs.microsoft.com/en-us/windows-server/networking/sdn/azure and sdn

[4] (2018). *ATT SD-WAN*. [Online]. Available: https://www.business.att.com/solutions/Family/network-services/sd-wan/

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

[6] J. Deng *et al.*, "On the safety and efficiency of virtual firewall elasticity control," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 129–131.

[7] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Dec. 2015.

[8] R. Jang, D. Cho, Y. Noh, and D. Nyang, "RFlow+: An SDN-based WLAN monitoring and management framework," in *Proc. IEEE Conf. Comput. Commun.*, May 2017, pp. 1–9.

[9] Q. Li, X. Zou, Q. Huang, J. Zheng, and P. P. C. Lee, "Dynamic packet forwarding verification in SDN," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 6, pp. 915–929, Nov. 2019.

[10] Q. Li, Y. Chen, P. P. C. Lee, M. Xu, and K. Ren, "Security policy violations in SDN data plane," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1715–1727, Aug. 2018.

[11] J. Cao, R. Xie, K. Sun, Q. Li, G. Gu, and M. Xu, "When match fields do not need to match: Buffered packets hijacking in SDN," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2020, pp. 1–15.

[12] Q. Li, Y. Liu, Z. Liu, P. Zhang, and C. Pang, "Efficient forwarding anomaly detection in software-defined networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2676–2690, Mar. 2021.

[13] X. Wen *et al.*, "SDNShield: Reconciliating configurable application permissions for SDN app markets," in *Proc. DSN*, 2016, pp. 121–132.

[14] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the software defined network control layer," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 1–15.

[15] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2015, pp. 239–250.

[16] C. Röpke and T. Holz, "SDN rootkits: Subverting network operating systems of software-defined networks," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Cham, Switzerland: Springer, 2015, pp. 339–356.

[17] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu, "Attacking the brain: Races in the SDN control plane," in *Proc. USENIX Secur. Symp.*, 2017, pp. 451–468.

[18] J. Cao, M. Xu, Q. Li, K. Sun, Y. Yang, and J. Zheng, "Disrupting SDN via the data plane: A low-rate flow table overflow attack," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Cham, Switzerland: Springer, 2017, pp. 356–376.

[19] A. Shaghaghi, M. A. Kaafar, and S. Jha, "WedgeTail: An intrusion prevention system for the data plane of software defined networks," in *Proc. Asia Conf. Comput. Commun. Secur.*, 2017, pp. 849–861.

[20] H. Cui, G. O. Karame, F. Klaedtke, and R. Bifulco, "On the fingerprinting of software-defined networks," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 10, pp. 2160–2173, May 2016.

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

[22] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial network forensics in software defined networking," in *Proc. Symp. SDN Res. ACM*, 2017, pp. 8–20.

[23] S. Liu, M. K. Reiter, and V. Sekar, "Flow reconnaissance via timing attacks on sdn switches," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 196–206.

[24] W. Braun and M. Menth, "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices," *Future Internet*, vol. 6, no. 2, pp. 302–336, 2014.

[25] *OSPF Version 2*, document RFC 2328, 1998. [Online]. Available: https://tools.ietf.org/html/rfc2328/

[26] *Border Gateway Protocol 4 (BGP-4)*, document RFC 4271, 2006. [Online]. Available: https://tools.ietf.org/html/rfc4271/

[27] (2018). *Traceroute*. [Online]. Available: https://en.wikipedia.org/wiki/Traceroute/

[28] (2011). *The Internet Topology Zoo*. [Online]. Available: http://www.topology-zoo.org/dataset.html

[29] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: The shrew vs. the mice and elephants," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2003, pp. 75–86.

[30] 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.*, 2013, pp. 413–424.

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

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

[33] (2014). *Floodlight Learning Switch*. [Online]. Available: https://github.com/floodlight/floodlight/blob/master/src/main/java/net/floodlightcontroller/learningswitch/

[34] (2016). *Floodlight Reactive Routing*. [Online]. Available: https://github.com/floodlight/floodlight/tree/master/src/main/java/net/floodlightcontroller/routing/

[35] (2014). *Floodlight Load Balancer*. [Online]. Available: https://github.com/floodlight/floodlight/tree/master/src/main/java/net/floodlightcontroller/loadbalancer

[36] (2018). *Open Networking Foundation (ONF)*. [Online]. Available: https://www.opennetworking.org/

[37] (2013). *Floodlight ARP Proxy*. [Online]. Available: https://github.com/mbredel/floodlight-proxyarp/

[38] (2013). *Floodlight DHCP Server*. [Online]. Available: https://github.com/rizard/DHCPServerForFloodlight/

[39] P. Lin, J. Bi, and H. Hu, "BTSDN: BGP-based transition for the existing networks to SDN," *Wireless Pers. Commun.*, vol. 86, no. 4, pp. 1829–1843, 2016.

[40] (2018). *Raw Sockets*. [Online]. Available: https://en.wikipedia.org/wiki/Network socket#Rawsocket

[41] J. F. Box *et al.*, "Guinness, Gosset, Fisher, and small samples" *Stat. science*, vol. 2, no. 1, pp. 45–52, 1987.

[42] (1997). *Nmap*. [Online]. Available: https://nmap.org/

[43] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematiks*, vol. 1, no. 1, pp. 269–271, 1959.

[44] (2019). *Floodlight Controller*. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/overview

[45] (2018). *AS4610-54T Data Center Switch*. [Online]. Available: https://www.edge-core.com/productsInfo.php?cls=1&cls2=9&cls3=46&id=21

[46] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks and counter strategies," *IEEE/ACM Trans. Netw.*, vol. 14, no. 4, pp. 683–696, Aug. 2006.

[47] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, Oct. 2013.

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

[49] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[50] Z. Wu, L. Zhang, and M. Yue, "Low-rate DoS attacks detection based on network multifractal," *IEEE Trans. Dependable Secure Comput.*, vol. 13, no. 5, pp. 559–567, Sep. 2015.

[51] (2015). *CAIDA Passive Monitor: Chicago B*. [Online]. Available: http://www.caida.org/data/passive/trace stats/chicago-B/2015/?monitor=20150219-130000.UTC

[52] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proc. 2nd ACM SIGCOMM workshop Hot Topics Softw. Defined Netw.*, 2013, pp. 165–166.

[53] R. Klöti, V. Kotronis, and P. Smith, "OpenFlow: A security analysis," in *Proc. Int. Conf. Netw. Protocols*, 2013, pp. 1–6.

[54] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, vol. 15, 2015, pp. 8–11.

[55] S. Jero, W. Koch, R. Skowyra, H. Okhravi, C. Nita-Rotaru, and D. Bigelow, "Identifier binding attacks and defenses in software-defined networks," in *Proc. USENIX Secur. Symp.*, 2017, pp. 415–432.

[56] S. Jero, X. Bu, C. Nita-Rotaru, H. Okhravi, R. Skowyra, and S. Fahmy, "Beads: Automated attack discovery in openflow-based sdn systems," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2017, pp. 311–333.

[57] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. Porras, "Delta: A security assessment framework for software-defined networks," in *Proc. NDSS*, vol. 17, Feb. 2017, pp. 1–15.

[58] B. E. Ujcich, U. Thakore, and W. H. Sanders, "Attain: An attack injection framework for software-defined networking," in *Proc. DSN*, 2017, pp. 567–578.

[59] Y. Zhang, Z. M. Mao, and J. Wang, "Low-rate tcp-targeted DoS attack disrupts internet routing," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2007, pp. 1–15.

[60] M. Schuchard, A. Mohaisen, D. F. Kune, N. Hopper, Y. Kim, and E. Y. Vasserman, "Losing control of the internet: Using the data plane to attack the control plane," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2010, pp. 726–728.

[61] H. Sun, J. C. Lui, and D. K. Yau, "Defending against low-rate TCP attacks: Dynamic detection and protection," in *Proc. Int. Conf. Netw. Protocols*, 2004, pp. 196–205.

[62] A. Shevtekar, K. Anantharam, and N. Ansari, "Low rate TCP denial-of-service attack detection at edge routers," *IEEE Commun. Lett.*, vol. 9, no. 4, pp. 363–365, Apr. 2005.

[63] Y. Chen, K. Hwang, and Y.-K. Kwok, "Collaborative defense against periodic shrew DDoS attacks in frequency domain," *ACM Trans. Inf. Syst. Secur.*, vol. 30, pp. 1–30, May 2005.

[64] Y. Xiang, K. Li, and W. Zhou, "Low-rate DDoS attacks detection and traceback by using new information metrics," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 2, pp. 426–437, Jun. 2011.

[65] J. Luo, X. Yang, J. Wang, J. Xu, J. Sun, and K. Long, "On a mathematical model for low-rate shrew DDoS," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 7, pp. 1069–1083, Jul. 2014.

[66] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, "Power spectrum entropy based detection and mitigation of low-rate DoS attacks," *Comput. Netw.*, vol. 136, pp. 80–94, May 2018.

[67] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *Proc. Symp. Secur. Privacy*, 2013, pp. 127–141.

[68] A. Studer and A. Perrig, "The coremelt attack," in *Proc. Eur. Symp. Res. Comput. Secur.* Cham, Switzerland: Springer, 2009, pp. 37–52.

[69] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu, "Realtime DDoS defense using COTS SDN switches via adaptive correlation analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 7, pp. 1838–1853, Jul. 2018.

[70] J. Wang, R. Wen, J. Li, F. Yan, B. Zhao, and F. Yu, "Detecting and mitigating target link-flooding attacks using SDN," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 6, pp. 944–956, Nov./Dec. 2018.

[71] M. S. Kang *et al.*, "SPIFFY: Inducing cost-detectability tradeoffs for persistent link-flooding attacks," in *Proc. NDSS*, vol. 1, Feb. 2016, pp. 53–55.

Renjie Xie received the B.Eng. degree from the Beijing University of Posts and Telecommunications in 2016 and the M.Sc. degree from Tsinghua University in 2019, where he is currently pursuing the Ph.D. degree. His current research interests include networks and machine learning security.

Jiahao Cao received the B.Eng. degree from the Beijing University of Posts and Telecommunications in 2015 and the Ph.D. degree from Tsinghua University in 2020. He has been a Visiting Scholar at George Mason University. He is currently a Post-Doctoral Researcher at the Department of Computer Science and Technology, Tsinghua University. His current research interests include SDN security, routing security, and network traffic analysis.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University, Beijing, China. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network and system security, particularly in internet and cloud security, mobile security, and big data security. He is an Editorial Board Member of IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and *DTRAP* (ACM).

Kun Sun (Member, IEEE) received the Ph.D. degree from the Department of Computer Science, North Carolina State University. He is currently an Associate Professor with the Department of Information Sciences and Technology, George Mason University. He is also the Associate Director of the Center for Secure Information Systems and the Director of the Sun Security Laboratory, George Mason University. He has more than 15 years working experience in both industry and academia on systems 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 an Associate Professor with the Department of Computer Science and Engineering, Texas A&M University. He is directing the SUCCESS—Secure Communication and Computer Systems Laboratory, TAMU. He was a recipient of the 2010 NSF CAREER Award, the 2013 AFOSR Young Investigator Award, the Best Student Paper Award from Oakland'10, the Best Paper Award from ICDCS'15, and the Google Faculty Research Award.

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

Yuan Yang (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees from Tsinghua University. He was a Visiting Ph.D. Student with The Hong Kong Polytechnic University from 2012 to 2013. He is currently an Assistant Researcher with the Department of Computer Science and Technology, Tsinghua University. His major research interests include computer network architecture, routing protocol, and green networking.