The CrossPath Attack: Disrupting the SDN Control Channel via Shared Links

Jiahao Cao, Qi Li, and Renjie Xie, *Tsinghua University;* Kun Sun, *George Mason University;* Guofei Gu, *Texas A&M University;* Mingwei Xu and Yuan Yang, *Tsinghua University*

https://www.usenix.org/conference/usenixsecurity19/presentation/cao

This paper is included in the Proceedings of the 28th USENIX Security Symposium.

August 14–16, 2019 • Santa Clara, CA, USA

978-1-939133-06-9

The CrossPath Attack: Disrupting the SDN Control Channel via Shared Links

Jiahao Cao[1,2], Qi Li[2,3], Renjie Xie[1,2], Kun Sun[4], Guofei Gu[5],
Mingwei Xu[1,2], and Yuan Yang[1,2]

[1]*Department of Computer Science and Technology, Tsinghua University*
[2]*Beijing National Research Center for Information Science and Technology, Tsinghua University*
[3]*Institute for Network Sciences and Cyberspace, Tsinghua University*
[4]*Department of Information Sciences and Technology, George Mason University*
[5]*SUCCESS LAB, Texas A&M University*

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*. Both theoretic analysis and experimental results demonstrate its feasibility and efficiency of identifying the target paths. 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.

1 Introduction

Software-Defined Networking (SDN) becomes increasingly popular and is being widely deployed in data centers [32], cloud networks [13], and wide area networks [11]. 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 [47]. SDN enables diversified packet processing and drives network innovation. A large number of network services and applications [26, 40, 33] 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. In particular, 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 [63, 48], attacks on crashing controllers [60, 49, 65], attacks on disrupting switches [22, 51], and information leakage in SDN [25, 56, 19, 45] , 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 [21, 65], 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 IP networks where almost all links deliver both control traffic (e.g., OSPF or BGP updates [1, 2]) and data traffic at the same time, only a few number of links 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 needs to find a target path that contains the shared links between control and data traffic to send the attack traffic. However, it is difficult to know 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 IP networks due to different forwarding actions in SDN. For example, Traceroute [17] cannot work well because 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 of data traffic that contains the shared links. The key observation 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. Meanwhile, such delays that indicate the path of the current data traffic has shared links with control traffic can be measured by a host. The reason one host can measure the delays is that the first packet of a new flow will be sent to the controller to query forwarding actions, which incurs extra delays of control messages other than that of the following packets directly processed in the data plane. Thus, by crafting timing packets to measure 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. By conducting the above reconnaissances on each possible path, a target path can finally be found.

We note the probing technique may fail to identify a target path in rare cases. We study the conditions of successful probing, and our experiments with 261 real network topologies [4] demonstrate that these conditions can be easily met in practice. Moreover, we analyze the expected number of paths that need to be explored for an attacker to find a target path. Both theoretical analysis and experimental results show the high efficiency of our probing technique. For example, it only needs to explore less than 50 paths on average if there are 1,000 paths and only 2% of them contains shared links. Experimental results in a real SDN testbed show our reconnaissances can achieve more than 90% accuracy.

In order to ensure the stealthiness of the attack, we leverage the low-rate TCP-targeted DoS [41] 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 timeout for TCP connections of the control channel. Compared with direct link flooding on the shared links, it significantly reduces the volume of attack traffic. Note the TCP-targeted DoS cannot effectively disrupt SDN networks without the knowledge of shared links obtained by our probing technique. Moreover, our attack is significantly different from the packet-in flooding attacks [55, 60] that trigger a huge volume of *control traffic* with bogus packets to saturate the SDN control channel. Instead, it 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 [60], FloodDefender [52], and SPHINX [27].

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 targets at disrupting the core services in SDN controllers that support these applications. In order to understand the impacts, we conduct experiments with four typical applications that have been widely deployed in SDN controllers, i.e., ARP Proxy [5], Learning Switch [6], Reactive Routing [9], and Load Balancer [7]. The results show (1) the performance of ARP Proxy can be significantly degraded, such as 10 times increase in the response delays and 95% reduction in the number of the ARP replies; (2) Learning Switch cannot successfully install forwarding decisions in the data plane and thus the throughput of the data plane is reduced to 0 Mbps; (3) Reactive Routing cannot update routing information in time and obtain incorrect topology information, which incurs various routing anomalies, e.g., routing loop, routing blackhole, routing path eviction, and flow table resetting; and (4) Load Balancer generates wrong decisions, resulting in link overloading.

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 prove the conditions of successful probing, analyze the expected number of explored paths to find a target path, and validate our analysis with experiments.
- We perform a systematical study and conduct extensive experiments on four typical SDN applications to demonstrate the impacts of the attack on various SDN network functionalities.

The rest of the paper is organized as follows. Section 2 provides background information about SDN and threat model. Section 3 presents the CrossPath attack along with an effective probing technique. Section 4 evaluates the feasibility and effectiveness of the attack both in large-scale simulations and real SDN testbeds. Section 5 further studies the impacts of the attack on different SDN applications by detailed analysis and extensive experiments. Section 6 discusses defense mechanisms that can be immediately deployed in practice to mitigate the attack. Section 7 reviews related work. Section 8 concludes the paper.

2 Background and Threat Model

2.1 Background

In this section, we briefly review the SDN architecture and a typical protocol of SDN, i.e., the OpenFlow protocol [47]. SDN enables network innovations by decoupling the control and data planes and provide programmability as well as flexibility. The control plane is logically centralized and can be deployed on commodity servers. 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, network monitoring, 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.

The dominant communication protocol between the control and data planes is OpenFlow, which has been standardized by the Open Networking Foundation (ONF) [14]. OpenFlow allows a controller to dynamically 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. OpenFlow also defines how to handle packets in a switch. When a switch receives a packet, it processes the packet based on the rule that matches the packet with the highest priority. If no rules match the packet, the switch sends the packet to the SDN controller through the control channel with a *packet_in* message. Applications running on the controller analyze the packet and make decisions. Once the decisions are made, the packet will be sent back to the switch with a *packet_out* message. The corresponding flow rules will be installed into all switches forwarding the packet with *flow_mod* messages. Such a packet processing procedure is called reactive rule installation, which has been widely used in OpenFlow networks [60, 52]. Moreover, to reduce the cost of building a dedicated control network and operating networks, in particular in large-scale networks [21, 65], OpenFlow allows the control and data traffic to share some links in the network, which is called in-band control.

2.2 Threat Model

In this paper, we consider an SDN network deployed with the OpenFlow protocol. The network uses a reactive approach to install flow rules, which is widely adopted in practice [60, 52], over an in-band control channel [21, 65]. 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.

3 The CrossPath Attack

In this section, we present the CrossPath attack on disrupting the SDN control channel. Particularly, we develop a probing technique called *adversarial path reconnaissance* to accurately find a target path containing shared links.

3.1 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, an attacker can almost break down all network functionalities enabled by SDN by constructing the attack. 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, the attack utilizes the low-rate TCP-targeted DoS (LDoS) [41] to generate periodic on-off "square-wave" traffic, which leads to repeated TCP retransmission timeout for the TCP connections of the control channel.

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

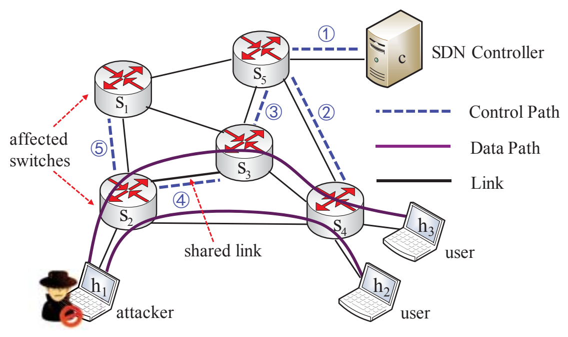


Figure 1: An example of disrupting the SDN control channel.

paths of s_2 and s_1, the control messages delivered between the 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 because SDN has different forwarding behaviors compared to traditional IP networks. For example, Traceroute [17] cannot infer the routing path of the packets, as SDN usually does not decrease the time-to-live (TTL) values in packet headers.

3.2 Adversarial Path Reconnaissance

To address the challenges above, we develop a probing technique called *adversarial path reconnaissance* to find target data paths that have links shared with control paths. The technique inspired by the key observation 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 the key data paths by generating data traffic and measuring the delay variations of the control paths. To achieve the goal, our adversarial path reconnaissance consists of two phases: measuring the delays of control paths and identifying a target data path.

Measuring Delays of Control Paths. In SDN, packets that cannot be matched in a switch will experience long forwarding paths and high delays, since they will be forwarded to the controller to request flow rules. We can analyze the delays of these packets to calculate the delays of control paths that share links with data paths. Assume there are two hosts h_i and h_j, and the data path between them is a sequence of consecutive links $P_d^{i,j} = < l_{h_i \to s_1}, l_{s_1 \to s_2}, \ldots, l_{s_\omega \to h_j} >$. Figure 2a shows the forwarding path and delay for a packet that is sent from h_i to h_j. The packet cannot be matched by flow rules in s_1. We can know the end-to-end delay for the packet is:

$$d_{i,j} = d_{prop}^{h_i} + \sum_{k=1}^{\omega+1} d_{trans}^{k} + \sum_{k=1}^{\omega} (d_{queue}^{k} + d_{proc}^{k}) + \delta_{i,j}, \qquad (1)$$

where $d_{prop}^{h_i}$ is the propagation delay at host h_i, d_{trans}^{k} is the transmission delay at the k^{th} link, d_{queue}^{k} is the queuing delay at the k^{th} switch, and d_{proc}^{k} is the processing delay at the k^{th} switch. $\delta_{i,j}$ is the delay of the control path, which is caused by querying controllers for rule installation. The delay pattern of such packet is shown in Figure 2a. However, if we send the same packet after the rule installation, the path and delay will become shorter, as shown in Figure 2b. The end-to-end delay can be expressed as follows:

$$d'_{i,j} = d_{prop}^{h_i} + \sum_{k=1}^{\omega+1} d_{trans}^{k} + \sum_{k=1}^{\omega} (\hat{d}_{queue}^{k} + d_{proc}^{k}). \qquad (2)$$

Here, we change d_{queue}^{k} to $\hat{d}_{queue}^{k}$ because the queuing delay depends on the current network traffic and is time-varying. Based on equation (1) and (2), the delay of the control path is:

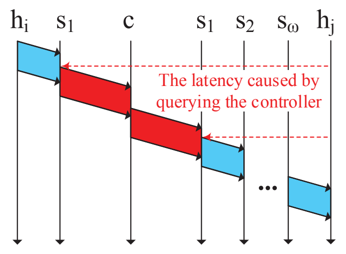
$$\delta_{i,j} = d_{i,j} - d'_{i,j} + \sum_{k=1}^{\omega} (\hat{d}_{queue}^{k} - d_{queue}^{k}) \qquad (3)$$

However, if we send two packets with a short time interval, e.g., sending the same packet immediately once we receive a response to the last packet, the queuing delay d_{queue}^{k} and $\hat{d}_{queue}^{k}$ can be approximately equal. Thus, we have $\delta_{i,j} \approx d_{i,j} - d'_{i,j}$. Similarly, we have $\delta_{j,i} \approx d_{j,i} - d'_{j,i}$. We use δ to denote the sum of $\delta_{i,j}$ and $\delta_{j,i}$. We have the following equation:

$$\delta \approx (d_{i,j} + d_{j,i}) - (d'_{i,j} + d'_{j,i}). \qquad (4)$$

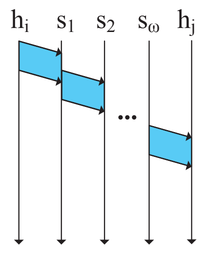
Note that, $d_{i,j} + d_{j,i}$ is the round-trip-time (RTT) of the packet that is not matched by rules, and $d'_{i,j} + d'_{j,i}$ is the RTT of the same packet matched by rules. Thus, we can infer the delay of control paths between two hosts by subtracting RTTs of these two crafted packets.

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 measure the delay δ shown in equation (4). The timing stream must trigger responses from the destination host in the current data path. Fortunately, many types of packets meet the requirement, such as ICMP packets, TCP SYN packets, and HTTP request packets. Moreover, each timing stream must contain a pair of



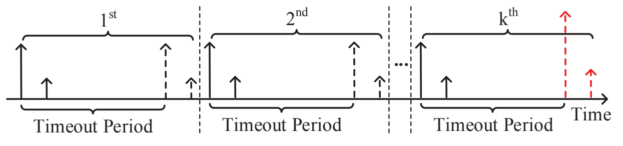
h_i s_1 c s_1 s_2 s_ω h_j
The latency caused by querying the controller
...

(a) A packet matches no rules in s_1.

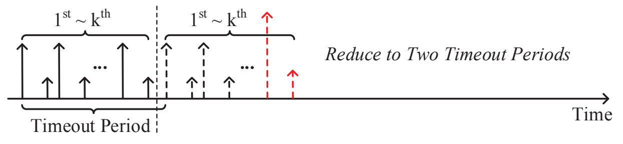


(b) A packet matches rules in all switches.

Figure 2: Different forwarding paths and delays for packets sent from h_i to h_j. c denotes the controller and s_i denotes the i^{th} switch in the packet path.



(a) Serial Reconnaissance.



(b) Parallel Reconnaissance.

Figure 3: Two different reconnaissances of finding a target data path. Each arrow denotes a timing packet and the height of it denotes the RTT of a timing packet. A dashed arrow denotes testing packets are sent at the same time. The red arrows denote a target path is found when conducting a reconnaissance on the k^{th} data path.

packets. The first packet must trigger new rule installation and the second packet must match the newly installed rules. This can be achieved by waiting a long enough time before sending the first timing packet to the destination, and then immediately sending another same packet after receiving a response from the first packet. The first packet can guarantee new rule installation, since old rules will be expired due to timeouts as we mentioned in Section 2. According to the previous study [44], the timeouts are usually configured as small values in order to save space of flow table and waiting for 30 seconds is enough for most cases.

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 [15] 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. After waiting enough time to ensure that old flow rules expire, 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.

Thus, we are able to find a target path by testing each path if it exists.

3.3 Improved Reconnaissance

In order to efficiently and accurately find a target data path, we apply two methods to improve our reconnaissance.

Improving Accuracy with *T-test*. Although our reconnaissance allows an attacker to know whether a data path crosses with control paths by sending only four packets, it may achieve low accuracy in practice. Various network noises can affect the reconnaissance. For example, a burst of benign traffic can also cause high latencies of control paths, which makes a non-target data path misidentified as a target data path. We find that t-test [20] can be a straightforward approach to eliminate the influences of network noise 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 that the two groups of samples belong to the same distribution. Typically, if p is less than a predetermined value, i.e., the significance level α [20], the two groups are considered significantly different. Thus, we can collect two groups of latencies 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.

Improving Efficiency with Parallelization. Basically, an attacker can try to test each data path one-by-one, which is shown in Figure 3a. However, it is time-consuming. An attacker has to wait for at least a timeout value before conducting next round of testing, as obtaining the latencies of control paths with testing stream requires that the old rules have been removed. Suppose that a network has 100 data paths and the timeouts in flow rules are configured to 10s. Moreover, we assume 10 repeated reconnaissances are conducted for each path in order to apply t-test. We can calculate that finding a target path needs approximate 10,000s at the worst case, which is unbearable in practice. Fortunately, different flow rules matching specific packets make up different data paths in SDN, which means the installation and expiration of rules in two different paths are independent. Thus, the reconnaissance can be parallelized to reduce the time. As shown in Figure 3b, an attacker can choose k pending paths. The latencies of their crossed control paths can be measured in turn by sending two timing packets for each data path. After waiting for only one timeout value, an attacker can measure the latencies again in turn while transmitting corresponding testing streams, since the old rules of each data path will expire in turn. The parallel reconnaissance allows an attacker to explore k data paths within two timeout values, which significantly improves efficiency. The maximal k depends on the maximal timeout values of flow rules and the maximal RTT of timing packets. In order to find a target data path as fast as possible, k should be subject to the inequation: $2 \cdot k \cdot RTT_{max} < timeout_{max}$. It ensures that an attacker can check whether there is a target path among k data paths within two timeout periods. If the maximal RTT of the timing packets is 20 ms in the target SDN, the parallel reconnaissance can dramatically reduce the time used by the previous example from 10,000s to less than 100s.

Based on the above designs, the algorithm of improved adversarial path reconnaissance can be easily implemented. Due to space constraints, for further details, we refer the reader to see the pseudo-code in Appendix A.

3.4 Theoretical Analysis

To understand the feasibility and efficiency of the adversarial path reconnaissance in SDN, we perform theoretical analysis to answer the following two questions:

- If there exists target data paths crossing with control paths in the network, which conditions the network must meet so that our reconnaissance can identify a target data path?
- How many data paths should be explored in order to find a target data path?

Firstly, we use an example to illustrate the network conditions that must meet for identifying a target data path before presenting the theory results. Figure 4 shows the target network where an attacker conducts reconnaissances. Each switch connects the controller through the shortest control paths. Switch s_2 and s_3 both have two different shortest control paths that can be chosen. We first consider the case where s_2 connects the controller via $< l_{s_2 \to s_5}, l_{s_5 \to s_6}, l_{s_6 \to c} >$ and s_3 connects the controller via $< l_{s_3 \to s_2}, l_{s_2 \to s_1}, l_{s_1 \to s_6}, l_{s_6 \to c} >$. Obviously, the data path from h_1 to h_2 crosses with the control path of s_3. However, an attacker cannot identify it. Measuring the delay of the crossed control paths is infeasible, since an adversary cannot trigger rule installation into s_3. If we consider another case where s_2 connects the controller via $< l_{s_2 \to s_1}, l_{s_1 \to s_6}, l_{s_6 \to c} >$ and s_3 connects the controller via $< l_{s_3 \to s_2}, l_{s_2 \to s_5}, l_{s_5 \to s_6}, l_{s_6 \to c} >$, the target data path from h_1 to h_2 crossing with the control path of s_2 can be identified. The main difference between

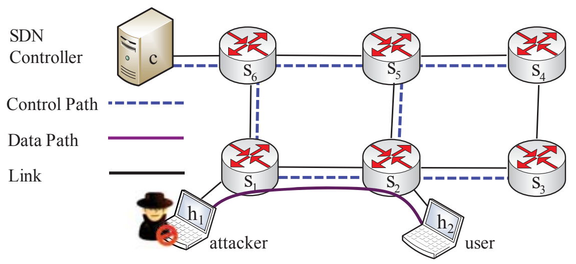


Figure 4: The target network where an attacker conducts reconnaissances.

the two cases is whether the target data path crosses with a control path of a switch belonging to the data path.

We consider a set of all the hosts in the target network $H = \{h_1, h_2, ..., h_n\}$, a set of compromised hosts $\tilde{H} = \{\tilde{h}_1, \tilde{h}_2, ..., \tilde{h}_q\}$, and a set of all the switches in the network $S = \{s_1, s_2, ..., s_m\}$. Let $p_d^{i,j}$ be the data path from host i to host j, let p_c^i be the control path of switch i, and let $S_{i,j} = \{s_1, s_2, ..., s_r\}$ be the set of switches belonging to the data path from host i to host j. Here, $\tilde{H} \subset H$ and $S_{i,j} \subset S$. $p_d^{i,j}$ and p_c^i both is a set that contains a sequence of consecutive links. In fact, we have the following theorem:

Theorem 1. *If and only if the target SDN network meets the condition:* $\exists (p_c^i \cap p_d^{j,k} \neq \emptyset)$, *where* $i \in S_{j,k}$, $j \in \tilde{H}$, $k \in H$ *and* $j \neq k$, *then there exists a target data path which can be identified by the adversarial path reconnaissance.*

Proof. We prove the theorem in two steps. We first prove the sufficient condition, i.e., if the target network meets the conditions in Theorem 1, then a target data path can be identified by the adversarial path reconnaissance. According to the conditions, we can know that a data path $p_d^{j,k}$ from a compromised host $\tilde{h}_j$ to another host h_k crosses with a control path p_c^i. The crossed control paths belong to the switches $S_{j,k}$ along the data path. An attacker can conduct the adversarial path reconnaissance on the data path. Basically, four timing packets will be sent to the data path. The first timing packet will trigger rule installation into all switches along the data path. Only after all switches finished installing rules according to the messages of the controller, the packet can reach the destination and a response packet will be sent to the compromised host. Thus, the RTT of the timing packet contains total latencies of control paths of all switches in $S_{j,k}$. The second timing packet will be sent after rule installation. The total latencies of control paths can be obtained by subtracting the RTTs of these two timing packets. After waiting at least a timeout value, another two timing packets can be sent to the data path with testing stream. The total latencies of control paths can be obtained again in a similar way; however, crossed control path p_c^i will be affected by the test stream. The reconnaissance will notice that the total latencies will be significantly higher than the previous latencies.

Thus, a target data path $p_d^{j,k}$ is identified.

We next prove the necessary condition, i.e., if a target data path can be identified by the adversarial path reconnaissance, then the target network meets the conditions in Theorem 1. We assume that a target data path $p_d^{j,k}$ is identified. Since $p_d^{j,k}$ is a target data path, it at least crosses with a control path p_c^i. Obviously, the reconnaissance can only be launched by the compromised hosts. Thus, $j \in \tilde{H}$, $k \in H$, and $j \neq k$. We only need to prove that the crossed control path belongs to a switch along with the data path $p_d^{j,k}$, i.e., $i \in S_{j,k}$. Let us consider the opposing case $i \notin S_{j,k}$. Note that the timing packets in our reconnaissance trigger rule installation into switches $S_{j,k}$ along the data path. Thus, only the latencies of control paths belonging to the switches in $S_{j,k}$ can be measured. When $i \notin S_{j,k}$, the delay variation of p_c^i cannot be noticed by our reconnaissance. Thus, there must be $i \in S_{j,k}$ if a target data path can be identified. $\square$

Theorem 1 indicates that our reconnaissance can find a target data path only if the network meets the conditions. Fortunately, it only requires at least one data path which crosses with a control path of switches that are in the data path. Such conditions can be easily met in practice. We will show that our reconnaissance can find a target data path with various real network topologies for most cases in Section 4.1.

In order to estimate the average number of explored data paths for finding a target data path, we introduce a parameter γ denoting the total number of target data paths which can be identified in a network. In addition to the notations we used in Theorem 1, let ρ be the total number of data paths between a compromised host in $\tilde{H}$ and a host in H, and let X be a random variable denoting the number of explored data paths for finding a target data path. Obviously, if we find a target data path at the k_{th} exploration, then we have already failed to find a target data path for $k-1$ times. Thus, the probability of finding a target data path at the k_{th} exploration for the first time is:

$$P(X=k) = \frac{\gamma}{\rho-(k-1)} \prod_{j=0}^{k-2} \frac{\rho-\gamma-j}{\rho-j}, \tag{5}$$

where $1 \le k \le \rho-\gamma+1$. Here, we define $\prod_{j=x}^{y} a = 1$, when $x > y$. The average number of explored data paths can be calculated as:

$$\begin{aligned} E(X) &= \sum_{k=1}^{\rho-\gamma+1} k \cdot P(X=k) \\ &= \sum_{k=1}^{\rho-\gamma+1} \frac{k\gamma}{\rho-(k-1)} \prod_{j=0}^{k-2} \frac{\rho-\gamma-j}{\rho-j}. \end{aligned} \tag{6}$$

If we consider the case where there is only one compromised host in the network and each of the data paths between two hosts is different, then $\rho = n-1$. n is the number of hosts in the network. Equation (6) can be simplified as:

$$E(X) = \sum_{k=1}^{n-\gamma} \frac{k\gamma}{n-k} \prod_{j=0}^{k-2} \left(1 - \frac{\gamma}{n-1-j}\right). \tag{7}$$

Equation (7) indicates the average number of explored data paths $E(X)$ totally depends on n and γ. We will show that $E(x)$ gets small values with proper parameters and the theoretical values are consistent with our experimental values in Section 4.1. In reality, our reconnaissance can quickly find a target data path by exploring several data paths (see Figure 6 in Section 4.1).

4 Attack Evaluation

In this section, we perform large-scale simulations to demonstrate that the CrossPath attack can be launched with various network topologies. Moreover, we conduct experiments to evaluate the feasibility and effectiveness of the attack in a real SDN testbed.

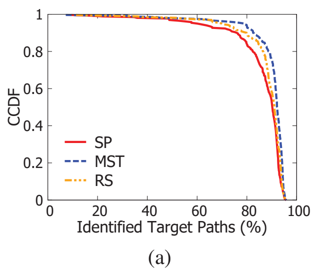
4.1 Large-Scale Simulation Experiments

Simulation Setup. We perform simulations with 261 real network topologies [4] 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 [28] 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 connection 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.

We note that the positions of hosts in a network will affect our experimental results. Thus, we conduct 1,000 experiments for each network topology and randomly changes 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 5a 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.

[1] In reality, we also conduct our experiments with 50, 500, 1000 hosts, respectively. The results are similar to those in Figure 5.



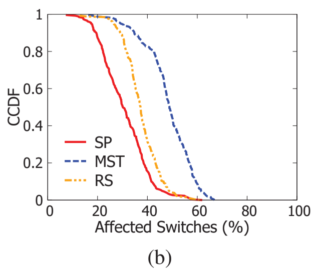


Figure 5: Complementary Cumulative Distribution Function (CCDF). (a) shows the CCDF of the average percentage of identified target paths with 261 real topologies; (b) shows the CCDF of the average percentage of affected switches by attacking a target path with 261 real topologies.

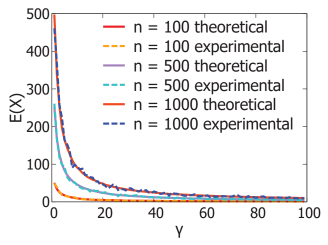


Figure 6: Comparison of theoretical values and experimental values of $E(X)$ with different n and γ.

The results demonstrate that the conditions in Theorem 1 can be easily met. An attacker can use our reconnaissance to find some target data paths to launch the CrossPath attack.

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 5b 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 possible for an attacker to attack multiple target paths to cause damages for the whole switches and incur network-wide DoS.

Average Number of Explored Data Paths. Equation (7) denotes the average number of explored data paths $E(X)$ for finding a target path totally depends on the number of data paths γ containing shared links and the number of hosts in a network n. We draw the theoretical values of $E(X)$ in Figure 6. We can see that $E(x)$ declines quickly when γ increases from 0 to 20. When there are 1,000 hosts and 40 data paths (2% of the 1,000 total data paths) containing shared links, $E(X)$ is less than 50. Moreover, $E(x)$ tends to be the same with the growth of γ. The results demonstrate that our reconnaissance can fast find a target data path and has a good scalability with a different number of hosts in the network. The experimental values of $E(x)$ are also plotted in Figure 6. Each experimental value with different n and γ is obtained by conducting 1,000 experiments to get the average number of explored data paths. The results show that the experimental values are consistent with the theoretical values.

4.2 Experiments in a Real SDN Testbed

Experiment Setup. Our testbed contains a popular SDN controller Floodlight [12], five hardware SDN switches (AS4610-54T [10]), 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. All hosts run Ubuntu 14.04 server LTS. The network topologies, control paths and data paths are illustrated in Figure 1. An attacker first compromises host h_1 to conduct the algorithm of adversarial path reconnaissance (see Appendix A for details) 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 [42] has conducted comprehensive experiments on how different parameters determine the attack impacts of LDoS flows and how to better choose these parameters. As our paper mainly focus 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 approximate 0.33 Gbps on average.

Accuracy of Reconnaissances. We first collect the delay variations in 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. We wait up to 20 seconds for each timing packet to get a response in order to obtain possible maximum delays. Figure 7 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 than 2 ms, while most delay variations are much larger for the target data path. These results illustrate that the discrimination

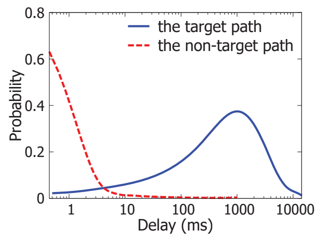


Figure 7: Probability distribution of delay variations.

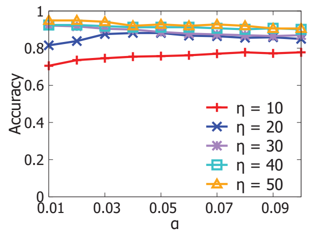


Figure 8: Accuracy of reconnaissances with different parameters.

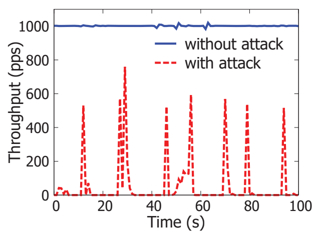


Figure 9: Throughput of control packets.

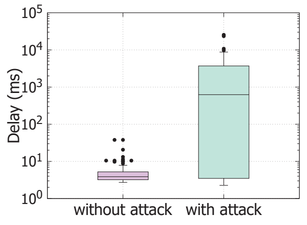


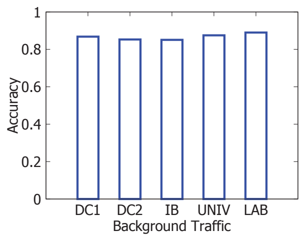
Figure 10: Delays of control packets.

between target data paths and non-target data paths can be easily identified according to the delay variations.

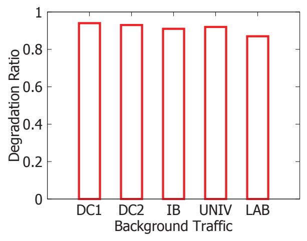
We then calculate the accuracy of our reconnaissance by conducting 1,000 repeated experiments with different settings of η and α. Here, η denotes the number of measured delays for each data path, which is also the size of each group in the t-test used to identify a target path. α is the significance level used in the t-test. As shown in Figure 8, 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 9 shows the throughput of control packets. The throughput can achieve 1,000 packets per second. However, it almost 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 timeout. In this case, no packets will be sent within the retransmission timeout. Figure 10 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. Note that, 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.

[2]There can be thousands of control packets per second [29]. For simplicity but without loss of generality, we choose a practical value, 1,000.



(a) Accuracy of Reconnaissances.



(b) Degradation Ratio of Control Traffic.

Figure 11: Robustness of the attack with different background traffic.

Robustness of the Attack. As background traffic may affect the reconnaissances and attack effects, we inject different background traffic into our network with TCPReplay [16] in turn. Such traffic traces comes from two Data Centers (DC1 and DC2) [3], an Internet Backbone (IB) [8], a University (UNIV) [18] and our Laboratory (LAB). Moreover, due to the limited flow table capacity in switches, we randomly choose flows from the trace to ensure that the number of rules generated by flows do not exceed the table capacity.

Figure 11a shows the accuracy of reconnaissances with different background traffic. The parameters of reconnaissances α and η are set to 0.01 and 50, respectively, which are the best parameters to get the highest accuracy (93% in Figure 8) without background traffic. When the background traffic is injected, the accuracy drops to below 90%, ranging from 85% to 89%. However, such accuracy is still satisfactory for an attacker to conduct reconnaissances. Figure 11b shows the degradation ratio of control packets. The degradation ratio is the fraction of the control packets reduced by the attack over the total control packets without the attack. We can see that the attack always causes more than 90% degradation ratio with different background traffic. Above results demonstrate that our attack achieves high robustness.

5 Attack Impacts on Network Functionalities

In this section, we perform a systematical study on the impacts of the attack on various network functionalities. We first review the common core services enabled in SDN controllers that generate different types of OpenFlow control messages and are used by various SDN applications. We then study four typical SDN applications, which use these common core services, so that we measure the impacts of

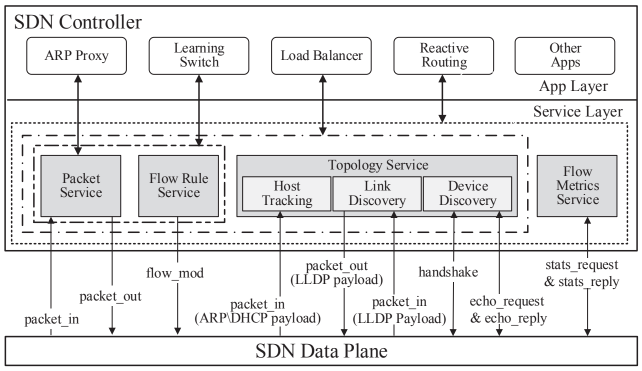


Figure 12: The core services of SDN controllers.

the attack on SDN functionalities.

5.1 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. As shown in Figure 12, 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 dispatch 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. Applications obtain network topologies through the service.

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 four typical applications that are widely deployed in SDN controllers to show the impacts of the attack on various network functionalities. The implementations of the four applications [5, 6, 9, 7] are from Floodlight [12].

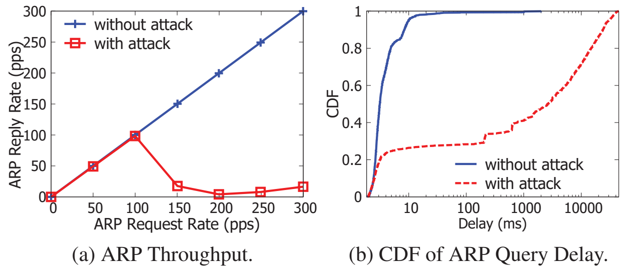
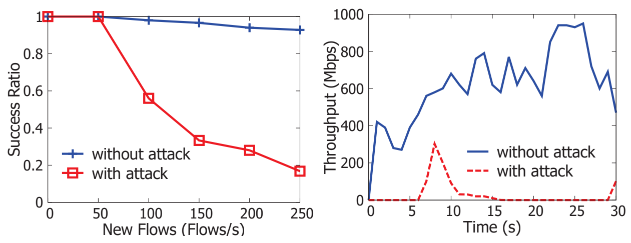


Figure 13: Attack impacts on ARP Proxy.

5.2 ARP Proxy

SDN enables Address Resolution Protocol (ARP) similar to IP networks, which finds the association between a destination IP address and its corresponding hardware (MAC) address so that hosts can correctly send and receive IP packets. In IP networks, layer two switches flood an ARP request sent from a host to get an ARP reply. If the target IP address in the ARP request is not in the local network, a router acts as an ARP proxy to send back an ARP reply with the hardware address of its own interface. In SDN, ARP packets are handled by an *ARP proxy* application [5] in the SDN controller. When an ARP request sent by a host arrives at a switch, it will be sent to the controller via *packet_in* messages. The packet service extracts the ARP request packet from *packet_in* messages and dispatches the packet to the ARP proxy application. The application extracts the sender IP address and the source MAC address to store them into the ARP table. Meanwhile, it finds an entry that the IP address matches the target IP address in the ARP request. A corresponding ARP reply packet is created and will be sent back to the ingress switch via *packet_out* messages. Thus, the original host obtains an ARP reply.

Our attack can completely disrupt the functionality of ARP proxy by interfering with the exchange of the messages between the packet service and switches. Figure 13a shows the ARP throughput. The ARP reply rate is proportional to the ARP request rate in absence of the attack. However, under the attack, the ARP reply rate falls below 10 pps when the ARP request rate exceeds 100 pps. The reason is that the TCP flows of control traffic frequently enter the retransmission timeout phase under the attack due to the congestion. Figure 13b shows the CDF of ARP delays. More than 90% delays are less than 10 ms without the attack, while more than 70% delays are higher than 10 ms and more than 50% delays are higher than 1,000 ms with the attack. Delays under attacks significantly increase. Particularly, some delays exceed 10,000 ms, which can cause connection failures between two hosts because hosts cannot get MAC addresses.



(a) Success Ratio of Rule Installation for a Switch.

(b) Throughput for a Switch with 250 Flows/s.

Figure 14: Attack impacts on Learning Switch.

5.3 Learning Switch

The *learning switch* application [6] allows SDN switches act as normal switches in IP 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. As shown in 12, 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 14 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 14a, 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 per-second, the success ratio reduces to below 20%. Thus, *learning switch* cannot work correctly. As shown in Figure 14b, the throughput of a switch is 0 Mbps for a long time with attack when there are 250 flows/s.

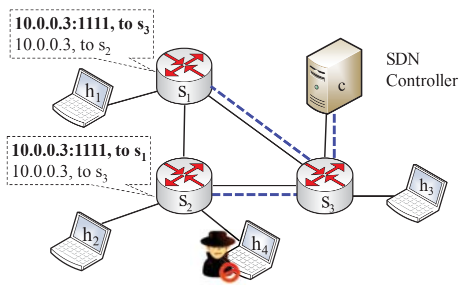
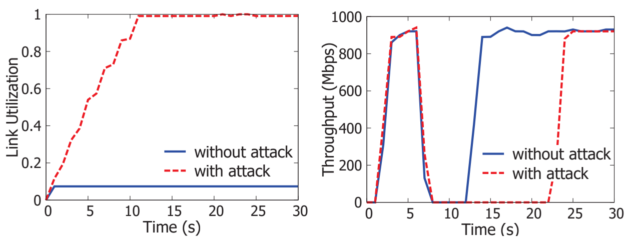
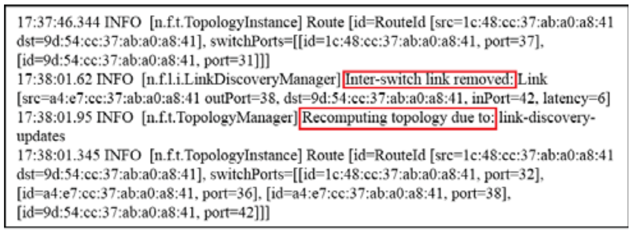


Figure 15: The network topology used in Reactive Routing.



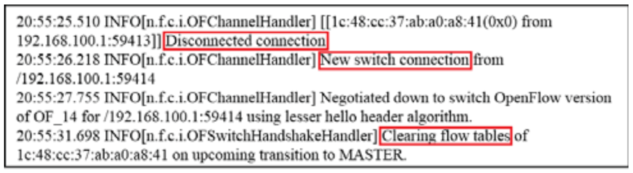
(a) Increasing link utilization due to long-term routing rule inconsistency.

(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.

Figure 16: Attack impacts on Reactive Routing.

5.4 Reactive Routing

The *reactive routing* [9] application enables flexible and fine-grained routing decisions for different flows, which is enabled in almost all controllers. 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 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 15. 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. Here, the compromised host h_4 sends attack (i.e., LDoS) traffic to h_3 in order to exploit the control path of switch s_2.

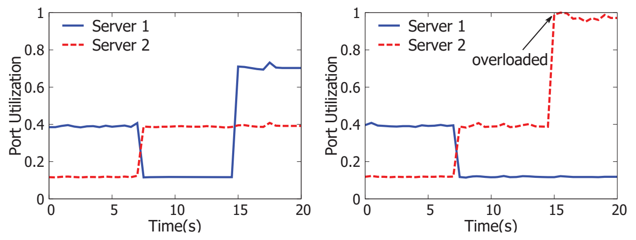
Figure 16 shows the impacts of the attack on *reactive routing*. As shown in Figure 16a, our attack incurs long-term routing rule inconsistency, which makes the link utilization reach 100%. The reason is that SDN exists transient rule inconsistency [36] which can be leveraged by our attack. In the network shown in Figure 15, 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 16b 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 16c 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 16d shows the connection of switch s_2 is reset and all the flow tables are cleaned.

5.5 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* [7] application to achieve the goal. The application in the Floodlight controller can balance requests of clients in two way, i.e., round robin and statistics-based scheduling. Round robin scheduling randomly chooses a server from a server pool to serve a new request each time. The statistics-based scheduling chooses a server that has the lowest utilization to serve a new request, where the utilization is calculated according to the real-time statistics of the switch ports. The *load balancer* application relies on the flow metrics service to collect the statistics.

We configure the *load balancer* application in Floodlight 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 17a 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 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 17b, 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 metrics service and switches, and thus the applications cannot know the port utilization in time. Actually, the application considers that the port utilization of server #2 is still 10% when the new flow comes.



(a) Port Utilization of Servers without Attack. (b) Port Utilization of Servers with Attack.

Figure 17: Attack impacts on Load Balancer for misallocating the workloads across servers.

6 Defense Schemes

In this section, we discuss possible countermeasures that network administrators can be used to mitigate the attack.

Delivering Control Traffic with High Priority. To defend against the attack, one way is to ensure forwarding control traffic with high priority, which thus can protect control traffic from being congested by malicious data traffic. According to our analysis, such a defense scheme can be enforced by carefully configuring Priority Queue (PQ) or Weighted Round Robin Queue (WRR) in switches. We note that many commercial SDN switches support at least one of the two queueing mechanisms (see Appendix C). We implement the defense scheme based on PQ and WRR in our hardware switches to deliver control traffic with high priority. The evaluation shows it can effectively protect control traffic against malicious data traffic. The detailed implementations and evaluations can be found in Appendix B.

Proactively Reserving Bandwidth for Control Traffic.

Another way to defend against the attack is to proactively reserve proprietary bandwidth for control traffic. Such a defense scheme is suitable for SDN switches that do not support PQ and WRR mechanisms. We implement the defense scheme with OpenFlow meter table in our hardware switches. We have demonstrated that control traffic can be well protected by reserving enough bandwidth. We refer the reader to Appendix B for details. The main disadvantage of the defense scheme is that the reserved bandwidth cannot be used by other traffic even there is massive free bandwidth. Our future work will focus on how to dynamically reserve the bandwidth for control traffic to make full use of it.

Disturbing Path Reconnaissances. The necessary condition to successfully launch the CrossPath attack is to find a target path containing shared links. Thus, we can prevent the attack by disturbing path reconnaissances. One way is to deliberately add random delays when installing flow rules, which may result in incorrect delay measurements of control paths when conducting path reconnaissances. Our evaluation shows that the accuracy of path reconnaissances can drops to less than 30% by adding random delays ranging from 100 ms to 1,000 ms. However, adding random delays affects the rule installation of all flows in the network. It is especially harmful to mice flows that are delay-sensitive [30]. Designing a scheme to effectively disturb path reconnaissances and reduce the impacts on network flows is worth more future research.

7 Related Work

In this section, we review related security research in SDN and legacy networks, respectively.

Reconnaissances in SDN. SDN reconnaissances has been extensively studied. Shin et al. [54] designed an SDN scanner to determine whether a network is SDN by measuring response delays of pings. Cui et al. [25] further conducted experiments in real SDN testbed to demonstrate its feasibility. Klöti et al. [39] presented a reconnaissance technique to determine if an SDN has rules for aggregated TCP flows by timing the TCP setup time. Achleitner et al. [19] designed SDNMap to reconstruct composition of flow rules by analyzing probing packets with specific protocols. Liu et al. [45] developed a Markov model to reveal rule distribution among switches. John et al. [56] 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 be applied to find target paths containing shared links with control paths.

Attacks on SDN and Related Defenses. SE-Floodlight [48] and SDNShield [63] are developed to provide permission control for malicious SDN applications. Some studies focus on the security of controllers, including network poisoning [31], identifier binding attacks [35], subverting SDN controllers [49], and exploiting harmful race conditions in SDN controllers [65]. Other studies focus on data plane security, including low-rate flow table overflow attacks [22], SDN teleportation, and detection on abnormal data plane [51]. Our paper focuses on the security of control channel, which is orthogonal to the existing work. Particularly, we uncover a new type of attack, which has not been discovered by existing automatic attack discovery tools [34, 43, 59] in SDN.

The packet_in flooding attack [55, 60] 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 [55, 60, 52, 27] against packet_in flooding attacks since they detect attacks by identifying and throttling malicious control traffic.

LDoS Attacks in Traditional IP Networks. Kuzmanovic et al. [41] developed low-rate TCP-targeted DoS attacks to disrupt TCP connections. Zhang et al. [66] demonstrated the attack has severe impact on the Border Gateway Protocol (BGP) by conducting real experiments. Schuchard et al. [50] extended the attack developed by Zhang et al. and designed the Coordinated Cross Plane Session Termination attack (CXPST) that allows an attacker to attack 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 [42] and complex signal analysis [58, 53, 23, 64, 46, 24] . However, randomizing RTO cannot fully mitigate the attack [66], 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 [50], such as BGP Graceful Restart. However, it is not suitable to protect the SDN control channel with "dumb" SDN switches.

Link Flooding Attacks in Traditional IP Networks. Studer et al. [38] and Kang et al. [57] 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 [67, 61, 62, 37] that detect link flooding attacks, they cannot defend the crosspath attack that disrupts the control channel, which these SDN defense systems depend on.

8 Conclusions

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. We hope this work attract more attention to SDN security, especially the possible attacks on the SDN control channel when deploying SDN to innovate network applications.

Acknowledgments

The research is partly supported by the National Key R&D Program of China under Grant 2017YFB0803202, the National Natural Science Foundation of China (NSFC) under Grant 61625203, 61572278, 61832013, 61872209, and U1736209, the U.S. ONR grants N00014-16-1-3214 and N00014-16-1-3216, and the National Science Foundation (NSF) under Grant 1617985, 1642129, 1700544, and 1740791. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSFC, NSF, and other sponsors. Qi Li and Mingwei Xu are the corresponding authors of the paper.

References

[1] RFC 2328. OSPF Version 2. `https://tools.ietf.org/html/rfc2328/`, 1998. [Online].

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

[3] Data Set for IMC 2010 Data Center Measurement. `http://pages.cs.wisc.edu/~tbenson/IMC10_Data.html`, 2010. [Online].

[4] The Internet Topology Zoo. `http://www.topology-zoo.org/dataset.html`, 2011. [Online].

[5] Floodlight ARP Proxy. `https://github.com/mbredel/floodlight-proxyarp/`, 2013. [Online].

[6] Floodlight Learning Switch. `https://github.com/floodlight/floodlight/blob/master/src/main/java/net/floodlightcontroller/learningswitch/`, 2014. [Online].

[7] Floodlight Load Balancer. `https://github.com/floodlight/floodlight/tree/master/src/main/java/net/floodlightcontroller/loadbalancer`, 2014. [Online].

[8] CAIDA Passive Monitor: Chicago B. `http://www.caida.org/data/passive/trace_stats/chicago-B/2015/?monitor=20150219-130000.UTC`, 2015. [Online].

[9] Floodlight Reactive Routing. `https://github.com/floodlight/floodlight/tree/master/src/main/java/net/floodlightcontroller/routing/`, 2016. [Online].

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

[11] AT&T SD-WAN. `https://www.business.att.com/solutions/Family/network-services/sd-wan/`, 2018. [Online].

[12] Floodlight Controller. `http://www.projectfloodlight.org/`, 2018. [Online].

[13] Microsoft Azure and Software Defined Networking. `https://docs.microsoft.com/en-us/windows-server/networking/sdn/azure_and_sdn/`, 2018. [Online].

[14] Open Networking Foundation (ONF). `https://www.opennetworking.org/`, 2018. [Online].

[15] Raw Sockets. `https://en.wikipedia.org/wiki/Network_socket#Raw_socket`, 2018. [Online].

[16] TCPReplay. `http://tcpreplay.synfin.net`, 2018. [Online].

[17] Traceroute. `https://en.wikipedia.org/wiki/Traceroute/`, 2018. [Online].

[18] Traffic and Tools. `http://traffic.comics.unina.it/Traces/ttraces.php`, 2018. [Online].

[19] ACHLEITNER, S., LA PORTA, T., JAEGER, T., AND MCDANIEL, P. Adversarial network forensics in software defined networking. In *Proceedings of the Symposium on SDN Research* (2017), ACM, pp. 8–20.

[20] Box, J. F., et al. Guinness, gosset, fisher, and small samples. *Statistical science 2*, 1 (1987), 45–52.

[21] Braun, W., and Menth, M. Software-defined networking using openflow: Protocols, applications and architectural design choices. *Future Internet 6*, 2 (2014), 302–336.

[22] Cao, J., Xu, M., Li, Q., Sun, K., Yang, Y., and Zheng, J. Disrupting sdn via the data plane: a low-rate flow table overflow attack. In *Proceedings of International Conference on Security and Privacy in Communication Systems* (2017), Springer, pp. 356–376.

[23] Chen, Y., Hwang, K., and Kwok, Y.-K. Collaborative defense against periodic shrew ddos attacks in frequency domain. *ACM Transactions on Information and System Security 30* (2005).

[24] Chen, Z., Yeo, C. K., Lee, B. S., and Lau, C. T. Power spectrum entropy based detection and mitigation of low-rate dos attacks. *Computer Networks 136* (2018), 80–94.

[25] Cui, H., Karame, G. O., Klaedtke, F., and Bifulco, R. On the fingerprinting of software-defined networks. *IEEE Transactions on Information Forensics and Security 11*, 10 (2016), 2160–2173.

[26] Deng, J., Li, H., Hu, H., Wang, K.-C., Ahn, G.-J., Zhao, Z., and Han, W. On the safety and efficiency of virtual firewall elasticity control. In *Proceedings of Network and Distributed System Security Symposium* (2017).

[27] Dhawan, M., Poddar, R., Mahajan, K., and Mann, V. Sphinx: Detecting security attacks in software-defined networks. In *Proceedings of Network and Distributed System Security Symposium* (2015).

[28] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[29] Dixit, A., Hao, F., Mukherjee, S., Lakshman, T., and Kompella, R. Towards an elastic distributed sdn controller. In *ACM SIGCOMM computer communication review* (2013), vol. 43, ACM, pp. 7–12.

[30] He, K., Rozner, E., Agarwal, K., Felter, W., Carter, J., and Akella, A. Presto: Edge-based load balancing for fast datacenter networks. In *ACM SIGCOMM Computer Communication Review* (2015), vol. 45, ACM, pp. 465–478.

[31] Hong, S., Xu, L., Wang, H., and Gu, G. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Proceedings of Network and Distributed System Security Symposium* (2015), vol. 15, pp. 8–11.

[32] Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review 43*, 4 (2013), 3–14.

[33] Jang, R., Cho, D., Noh, Y., and Nyang, D. Rflow+: An sdn-based wlan monitoring and management framework. In *Proceedings of IEEE Conference on Computer Communications* (2017), IEEE, pp. 1–9.

[34] Jero, S., Bu, X., Nita-Rotaru, C., Okhravi, H., Skowyra, R., and Fahmy, S. Beads: automated attack discovery in openflow-based sdn systems. In *Proceedings of International Symposium on Research in Attacks, Intrusions, and Defenses* (2017), Springer, pp. 311–333.

[35] Jero, S., Koch, W., Skowyra, R., Okhravi, H., Nita-Rotaru, C., and Bigelow, D. Identifier binding attacks and defenses in software-defined networks. In *Proceedings of USENIX Security Symposium* (2017), USENIX Association, pp. 415–432.

[36] Jin, X., Liu, H. H., Gandhi, R., Kandula, S., Mahajan, R., Zhang, M., Rexford, J., and Wattenhofer, R. Dynamic scheduling of network updates. *ACM SIGCOMM Computer Communication Review 44*, 4 (2014), 539–550.

[37] Kang, M. S., Gligor, V. D., Sekar, V., et al. Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. In *NDSS* (2016).

[38] Kang, M. S., Lee, S. B., and Gligor, V. D. The crossfire attack. In *Proceedings of Symposium on Security and Privacy* (2013), IEEE, pp. 127–141.

[39] Klöti, R., Kotronis, V., and Smith, P. Openflow: A security analysis. In *Proceedings of International Conference on Network Protocols* (2013), IEEE, pp. 1–6.

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

[41] Kuzmanovic, A., and Knightly, E. W. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications* (2003), ACM, pp. 75–86.

[42] KUZMANOVIC, A., AND KNIGHTLY, E. W. Low-rate tcp-targeted denial of service attacks and counter strategies. *IEEE/ACM Transactions on Networking 14*, 4 (2006), 683–696.

[43] LEE, S., YOON, C., LEE, C., SHIN, S., YEGNESWARAN, V., AND PORRAS, P. Delta: A security assessment framework for software-defined networks. In *Proceedings of Network and Distributed System Security Symposium* (2017), vol. 17.

[44] LENG, J., ZHOU, Y., ZHANG, J., AND HU, C. An inference attack model for flow table capacity and usage: Exploiting the vulnerability of flow table overflow in software-defined network. *arXiv preprint arXiv:1504.03095* (2015).

[45] LIU, S., REITER, M. K., AND SEKAR, V. Flow reconnaissance via timing attacks on sdn switches. In *Proceedings of International Conference on Distributed Computing Systems* (2017), IEEE, pp. 196–206.

[46] LUO, J., YANG, X., WANG, J., XU, J., SUN, J., AND LONG, K. On a mathematical model for low-rate shrew ddos. *IEEE Transactions on Information Forensics and Security 9*, 7 (2014), 1069–1083.

[47] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[48] PORRAS, P. A., CHEUNG, S., FONG, M. W., SKINNER, K., AND YEGNESWARAN, V. Securing the software defined network control layer. In *Proceedings of Network and Distributed System Security Symposium* (2015).

[49] RÖPKE, C., AND HOLZ, T. Sdn rootkits: Subverting network operating systems of software-defined networks. In *Proceedings of International Workshop on Recent Advances in Intrusion Detection* (2015), Springer, pp. 339–356.

[50] SCHUCHARD, M., MOHAISEN, A., FOO KUNE, D., HOPPER, N., KIM, Y., AND VASSERMAN, E. Y. Losing control of the internet: using the data plane to attack the control plane. In *Proceedings of the conference on Computer and communications security* (2010), ACM, pp. 726–728.

[51] SHAGHAGHI, A., KAAFAR, M. A., AND JHA, S. Wedgetail: An intrusion prevention system for the data plane of software defined networks. In *Proceedings of the Asia Conference on Computer and Communications Security* (2017), ACM, pp. 849–861.

[52] SHANG, G., ZHE, P., BIN, X., AIQUN, H., AND KUI, R. Flooddefender: protecting data and control plane resources under sdn-aimed dos attacks. In *Proceedings of IEEE Conference on Computer Communications* (2017), IEEE, pp. 1–9.

[53] SHEVTEKAR, A., ANANTHARAM, K., AND ANSARI, N. Low rate tcp denial-of-service attack detection at edge routers. *IEEE Communications Letters 9*, 4 (2005), 363–365.

[54] SHIN, S., AND GU, G. Attacking software-defined networks: A first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (2013), ACM, pp. 165–166.

[55] SHIN, S., YEGNESWARAN, V., PORRAS, P., AND GU, G. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 413–424.

[56] SONCHACK, J., DUBEY, A., AVIV, A. J., SMITH, J. M., AND KELLER, E. Timing-based reconnaissance and defense in software-defined networks. In *Proceedings of Conference on Computer Security Applications* (2016), ACM, pp. 89–100.

[57] STUDER, A., AND PERRIG, A. The coremelt attack. In *European Symposium on Research in Computer Security* (2009), Springer, pp. 37–52.

[58] SUN, H., LUI, J. C., AND YAU, D. K. Defending against low-rate tcp attacks: Dynamic detection and protection. In *Proceedings of International Conference on Network Protocols* (2004), IEEE, pp. 196–205.

[59] UJCICH, B. E., THAKORE, U., AND SANDERS, W. H. Attain: An attack injection framework for software-defined networking. In *Proceedings of International Conference on Dependable Systems and Networks* (2017), IEEE, pp. 567–578.

[60] WANG, H., XU, L., AND GU, G. Floodguard: A dos attack prevention extension in software-defined networks. In *Proceedings of International Conference on Dependable Systems and Networks* (2015), IEEE, pp. 239–250.

[61] WANG, J., WEN, R., LI, J., YAN, F., ZHAO, B., AND YU, F. Detecting and mitigating target link-flooding attacks using sdn. *IEEE Transactions on Dependable and Secure Computing*, 1 (2018), 1–1.

[62] WANG, L., LI, Q., JIANG, Y., JIA, X., AND WU, J. Woodpecker: Detecting and mitigating link-flooding attacks via sdn. *Computer Networks 147* (2018), 1–13.

[63] Wen, X., Yang, B., Chen, Y., Hu, C., Wang, Y., Liu, B., and Chen, X. Sdnshield: Reconciliating configurable application permissions for sdn app markets. In *Proceedings of International Conference on Dependable Systems and Networks* (2016), IEEE, pp. 121–132.

[64] Xiang, Y., Li, K., and Zhou, W. Low-rate ddos attacks detection and traceback by using new information metrics. *IEEE Transactions on Information Forensics and Security 6*, 2 (2011), 426–437.

[65] Xu, L., Huang, J., Hong, S., Zhang, J., and Gu, G. Attacking the brain: Races in the sdn control plane. In *USENIX Security Symposium* (2017), USENIX Association, pp. 451–468.

[66] Zhang, Y., Mao, Z. M., and Wang, J. Low-rate tcp-targeted dos attack disrupts internet routing. In *Proceedings of Network and Distributed System Security Symposium* (2007), Citeseer.

[67] Zheng, J., Li, Q., Gu, G., Cao, J., Yau, D. K., and Wu, J. Realtime ddos defense using cots sdn switches via adaptive correlation analysis. *IEEE Transactions on Information Forensics and Security 13*, 7 (2018), 1838–1853.

A The Algorithm of Adversarial Path Reconnaissance

Algorithm 1 shows the pseudo-code of improved adversarial path reconnaissance, which can be performed by any host in the 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 t_{wait} is the waiting time for rules to expire. The input t_{max} is the maximal waiting time for each timing packet to get a response in the target network, and α is the significance level used in the t-test. Here, t_{wait} must be larger than the timeouts of flow rules and t_{max} must be large enough so that most RTTs in the network do not exceed it. Step 1 gets all hosts in the network in order to explore the data paths between the compromised host and them. Step 2 initializes the maximal number of data paths that can be explored within two timeout values. The main loop is from Step 4 to Step 29. In each loop iteration, the algorithm tests k_{max} data paths. Step 5 to Step 20 collects 2η latencies of the crossed control paths for each of the k_{max} data paths. The delay of crossed control paths when the testing stream is not transmitted is obtained in Step 7 to Step 10. Step 13 to Step 18 obtain the delay while transmitting the testing stream. Step 11 and Step 19 both make the program paused for enough time so that old rules can expire before conducting the next reconnaissance. After obtaining all the latencies of possible crossed control paths for the k_{max} data

Algorithm 1 Adversarial Path Reconnaissance

```
Input: $\eta$, $t_{wait}$, $t_{max}$, $\alpha$
Output: $h$;
1:  $H \leftarrow ScanAllHosts()$
2:  $k_{max} \leftarrow t_{wait}/(2 \cdot t_{max})$
3:  $i \leftarrow 0$
4:  while $i < |H|$ do
5:      for $j = 0 \rightarrow \eta - 1$ do
6:          $t_{start} \leftarrow time()$
7:          for $k = i \rightarrow min(i + k_{max}, |H|)$ do
8:              $d_1 \leftarrow$ sendTimingStreamTo($H[k]$)
9:              $\delta_1[k].add(d_1)$
10:         end for
11:         $sleep(t_{wait} - (time() - t_{start}))$
12:         $t_{start} \leftarrow time()$
13:         for $k = i \rightarrow min(i + k_{max}, |H|)$ do
14:             startSendTestingStreamTo($H[k]$)
15:             $d_2 \leftarrow$ sendTimingStreamTo($H[k]$)
16:             stopSendTestingStreamTo($H[k]$)
17:             $\delta_2[k].add(d_2)$
18:         end for
19:         $sleep(t_{wait} - (time() - t_{start}))$
20:     end for
21:     for $k = i \rightarrow min(i + k_{max}, |H|)$ do
22:         if $tTest(\delta_1[k], \delta_2[k]) < \alpha$ and $sum(\delta_1[k]) < sum(\delta_2[k])$ then
23:             /* The data path from the compromised host to $H[k]$ crosses with control paths. */
24:             $output(H[k])$
25:             $exit()$
26:         end if
27:     end for
28:     $i \leftarrow i + k_{max}$
29: end while
```

paths, the t-test is applied to determine whether a data path crosses with control paths in Step 21 to Step 27. If the group of latencies with testing stream is dramatically higher than the other group, the algorithm outputs the destination host of the data path and terminates. Otherwise, the algorithm prepares for the next round of iteration in Step 28.

In our experiments on a real SDN testbed, t_{wait} is set as 30s which is larger than the default values of timeouts in the Floodlight controller in order to leave enough time for rules to be expired, and t_{max} is set to 1s for each timing packet to get a response. The settings of η and α are varied.

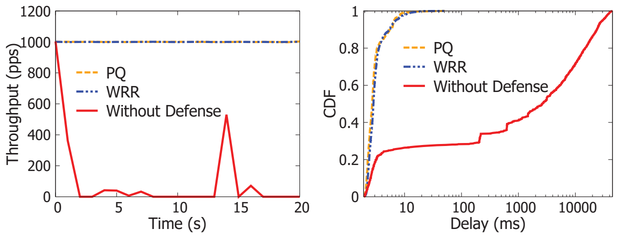
B Defense Against the CrossPath Attack

We explore two defense schemes to mitigate the CrossPath attack. The first is delivering control traffic with high priority. Hence, any malicious data traffic cannot disturb the delivery of control traffic. Such a defense scheme can be enforced with Priority Queue (PQ) or Weighted Round Robin

Table 1: The Settings for Flow Rules to Enforce Defense Strategies.

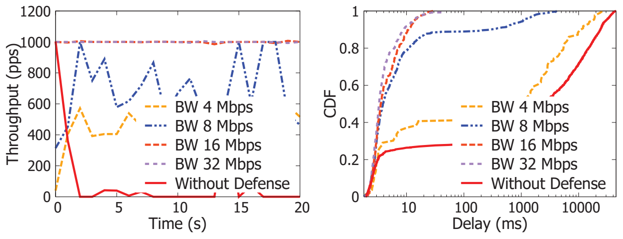
Defense Strategy	Rule	Match	Actions
Control traffic delivery with high priority [1]	#1	control flows	OutPort(x), ..., *SetQueue(ID=highPriQueue)*
	#2	data flows	OutPort(x), ..., *SetQueue(ID=lowPriQueue)*
Proactive bandwidth reservation for control traffic [2]	#1	data flows	OutPort(x), ..., *SetMeter(ID=RateLimit)*

[1] It requires SDN switches to support PQ or WRR queuing mechanism.
[2] It is used when SDN switches fail to enable PQ or WRR mechanism.



(a) Throughput of Control Packets with Queuing Policies. (b) CDF of Delays of Control Packets with Queuing Policies.

Figure 18: Evaluation on the defense scheme of delivering control traffic with high priority via PQ or WRR mechanism.



(a) Throughput of Control Packets with Bandwidth Reservation. (b) CDF of Delays of Control Packets with Bandwidth Reservation.

Figure 19: Evaluation on protecting control traffic with the defense scheme of proactive bandwidth (BW) reservation.

(WRR) [3] scheduling mechanism in SDN switches. Specifically, network administrators can inform controllers to add *SetQueue* actions to flow rules associated with switch ports in the control paths. Packets matching a flow rule with the *SetQueue* action will be directed to a queue with an *ID* set by the action. As shown in Table 1, we can set a flow rule matching control flows with a high priority queue and set a flow rule matching data flows with a low priority queue. In this way, the control traffic will always be forwarded in advance with no disturbances of other traffic.

We note that some switches in the market do not support PQ or WRR mechanisms. However, we can still mitigate the CrossPath attack by proactive bandwidth reservation for control traffic with OpenFlow meter table. A meter entry belonging to a meter table associates with various flow rules so that it can measure the total rate of packets matching the flow rules and enforce rate limiting. We can assign each flow rule matched by the data traffic a meter entry with the *SetMeter* action (see Table 1). Therefore, by limiting the maximal rate of the total data traffic, we reserve proprietary bandwidth for control traffic.

We evaluate above two defense schemes with AS4610-54T commercial hardware SDN switches in our testbed. For simplicity, we trigger 1,000 new flows per second to generate the control traffic and generate the attack traffic to disrupt the transmission of control packets. Figure 18a and 18b shows that defense schemes with PQ or WRR mechanism effectively protect the control traffic. The throughput always reaches approximate 1,000 pps over time even with the attack. The delays of more than 99% of the control packets are less than 10 ms with either of the two queuing mechanisms. Figure 19a and 19b show that proactive bandwidth reservation with meter table can also protect control traffic. The larger the reserved bandwidth is, the higher the throughput is, and also the better the delay is. In our experiments, 16 Mbps reserved bandwidth is enough to ensure forwarding control traffic. Note that compared with the queuing mechanism, it requires proactively reserving enough bandwidth for control traffic. In a large network, it may require reserving bandwidth in the order of several Gbps.

[3] By configuring different weighted values to queues with WRR, similar results like PQ can be achieved.

Table 2: SDN Switches with PQ or WRR Support.

Brand	Model	Queue Support	
		PQ	WRR
Pica8	All switches loaded with PicOS	√	√
Cisco	Catalyst 4500 Series Switches	√	×
Brocade	NetIron XMR Series, MLX Series, CES 2000, and CER 2000 Series	×	√
Dell	S4810, S4820T, S6000, Z9000, Z9500, and MXL switches	×	√
Huawei	CloudEngine 8800 Series	√	√

C SDN Switches with Queue Support

We investigate mainstream SDN switches and find that many switches support PQ or WRR mechanism. Table 2 shows the switches with PQ or WRR support.