Constructing SDN Covert Timing Channels Between Hosts With Unprivileged Attackers

Yixiong Ji, *Graduate Student Member, IEEE*, Jiahao Cao, *Member, IEEE*, Qi Li, *Senior Member, IEEE*, Yan Liu, Tao Wei, Ke Xu, *Fellow, IEEE, Member, ACM*, and Jianping Wu, *Fellow, IEEE*

***Abstract*— Software-defined networking (SDN) has been widely deployed due to its centralization and programmable features. However, these new features bring new threats at the same time. Previous studies have shown that SDN covert channels can be built with a privileged adversary that controls SDN key components, such as controller applications or SDN switches. In this paper, we propose new SDN covert timing channels between hosts without controlling applications, controllers, or having access to switches. Experiments in a real SDN testbed demonstrate the feasibility and effectiveness of our covert channels. To defend against the covert timing channels, we design a defense system named CovertGuard, which utilizes the timing characteristics of the covert channels' delays to detect and eliminate covert channels effectively.**

***Index Terms*— Covert channels, software-defined networking (SDN), defense.**

I. Introduction

SOFTWARE-DEFINED networking (SDN) [1], [2] separates data and control planes. As SDN standardizes southbound and northbound interfaces, network administrators typically do not need to design and implement different management and control software for different types of network devices [3]. Hence, the cost of network management and control is significantly reduced. Large companies such as Microsoft, Google, Amazon, and Alibaba Cloud have deployed SDN networks in their cloud networks on a large scale [4], [5]. They have developed various SDN applications for traffic engineering [6], QoS policing and management [7], load balancing [8], and security [9], etc. Moreover, SDN enables researchers to conduct experiments and test new network protocols [10].

Received 14 December 2023; revised 28 August 2024; accepted 5 November 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor N. Zhang. This work was supported in part by NSFC under Grant 62132011 and Grant 62202260, in part by China National Funds for Distinguished Young Scientists under Grant 62425201, in part by Ant Group, and in part by the Leading Innovative and Entrepreneur Team Introduction Program of Hangzhou under Grant TD2020001. *(Yixiong Ji and Jiahao Cao contributed equally to this work.) (Corresponding author: Qi Li.)*

Yixiong Ji, Ke Xu, and Jianping Wu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: ji-yx20@mails.tsinghua.edu.cn; xuke@tsinghua.edu.cn; jianping@cernet.edu.cn).

Jiahao Cao and Qi Li are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: caojh2021@tsinghua.edu.cn; qli01@tsinghua.edu.cn).

Yan Liu and Tao Wei are with Ant Group, Hangzhou 310023, China (e-mail: bencao.ly@antgroup.com; lenx.wei@antgroup.com).

Digital Object Identifier 10.1109/TNET.2024.3496997

Although SDN brings many advantages and enables innovations with centralized control, it may also introduce emerging security threats. Particularly, some studies [11], [12], [13], [14], [15] have discovered covert channels in SDN. Attackers can secretly disclose sensitive information to remote collaborators with covert channels and launch targeted attacks. Covert channels cause significant harm to SDN since they can bypass existing SDN security checks [16] and network access control policies [17], [18] to transmit messages.

Recent studies [19], [20], [21], [22] discover that covert channels can be built with different SDN mechanisms. SDN proxy and rule expiry mechanisms can be exploited to build covert timing and storage channels between controllers and switches [19]. Moreover, critical information can be secretly transmitted between switches through out-of-bound forwarding when malicious switches encode information into packet-in and packet-out messages [20]. Flow re-configuration and switch identification mechanisms enable secret communications between switches by declaring a fake connection to trigger path update or path reset operations [20], [21]. Covert storage channels can be built among hosts if an adversary controls SDN applications to add conflicting flow rules into switches [22]. However, successfully building these covert channels requires a privileged adversary that can control SDN key components, such as switches, controllers, or SDN applications. It is challenging in practice since there are many advanced countermeasures [23], [24], [25] to prevent adversaries from compromising SDN key components.

In this paper, we present new covert timing channels between hosts in SDN with unprivileged attackers. The key insight is to utilize the limited processing capacity of switches and the controller for proxy requests. Specifically, a host can quickly generate many proxy requests, such as ARP request packets, to increase packet processing delay on the controller or remain idle to signal a "1" or "0", respectively. Meanwhile, another host can sense the timing signal by generating proxy packets to measure the round-trip time (RTT) since the controller also processes its proxy packets. Hence, the two hosts originally isolated and unable to communicate have now established a covert timing channel.

We develop a systematic four-step approach to construct an efficient covert timing channel capable of transmitting continuous messages consisting of multiple bits. We start by probing three parameters that are crucial for constructing a covert timing channel: (1) the minimum speed of generating

SDN Proxy Request (SPR) packets to cause higher-than-usual RTTs, (2) the duration during which a sender host continuously sends SPR packets to increase the RTTs, and (3) the duration that the sender host remains idle to wait for the decrease of RTTs. In the second step, we synchronize the start time and period of the covert channel between the sender and the receiver hosts so that the message can be correctly encoded and decoded. In the last two steps, the sender encodes the message and transmits it by chronologically controlling the delay, while the receiver detects the delay sequence and decodes the message.

We build a real SDN testbed consisting of commercial SDN switches and controllers to evaluate our covert timing channels. Experimental results demonstrate that our covert channels can achieve an accuracy exceeding 98.5% when transmitting multiple bits without any error-correcting code. During the covert information transmission, CPU usage and control traffic throughput do not change significantly, which shows the acceptable overhead of our covert channels. Moreover, we demonstrate the successful transmission of a 512-byte TLS key within 10 minutes when the two hosts are under surveillance and unable to communicate.

To defend against the covert timing channels between hosts in SDN, we propose a defense system named CovertGuard. It exploits latency and timing characteristics to detect and eliminate covert channels. To reduce detection overhead, CovertGuard initiates a coarse-grained detection process to assess the presence of covert timing channels from a large volume of flows. Subsequently, it analyzes a subset of relevant traffic, identifying attackers by calculating the similarity of various flows and covert channel delay sequences. Finally, it eliminates the covert channels by installing flow rules that block the flows associated with covert timing channels. We conduct experiments to demonstrate that CovertGuard can effectively defend SDN covert timing channels. Our results show that CovertGuard can detect covert channels of different speeds in real time with 98% accuracy.

We highlight our key contributions as follows:

- We present new covert timing channels between hosts in SDN with unprivileged attackers.
- We comprehensively evaluate the covert timing channels in a real SDN testbed.
- We design and prototype CovertGuard, which utilizes delay and timing characteristics of covert channels to effectively detect and eliminate covert channels.

The rest of this paper is organized as follows: Section II illustrates our threat model. Section III presents our key insights and the design of covert timing channels between hosts in SDN and evaluates our covert channels. Section IV describes our design and implementation of CovertGuard. Section V discusses the limitations of CovertGuard. Section VI reviews related work. Section VII concludes this paper.

II. Threat Model

We assume that an attacker compromises some hosts in the target SDN. It may either be insiders who own hosts or users who rent hosts in the target network. For example, an attacker can rent hosts through SDN-based cloud services [26]. In addition, an attacker may compromise hosts through credential stuffing attacks [27], phishing attacks [28], etc. We assume that the SDN switches, the OpenFlow controller, and its applications are trusted. We also assume that the control channel is trustworthy and may be encrypted. Since there are strict access control policies and advanced security enhancements for controllers and switches [23], [24], [25], we do not assume that an attacker can compromise SDN controllers or switches.

We do not assume that the location of the compromised hosts in the network can be controlled by the attacker. These hosts may be connected to the same switch or connected to different switches. They may even be physically disconnected or logically disconnected. However, these hosts are all located in the target SDN domain. We assume that communication between these hosts is monitored and blocked by advanced security countermeasures, such as firewalls and intrusion detection systems. The goal of an attacker is to covertly transmit sensitive information between these hosts even in the presence of security mechanisms. The sensitive information may include private keys [29], confidential metadata [30], attack coordination messages [20], etc.

III. Covert Timing Channel Design and Evaluation

In this section, we introduce the design of our covert timing channel. Moreover, we conduct experiments in a real SDN testbed to evaluate our covert channels' accuracy, efficiency, stealthiness, and overhead.

A. Key Insight

In SDN, the network proxy functionalities are provided by running applications on controllers that leverage SDN proxy mechanisms, such as ARP Proxy [31], [32], [33], NDP Proxy [34], DHCP Proxy [35]. For simplicity but without loss of generality, we use ARP Proxy as an example to illustrate the SDN proxy mechanisms: (1) a host generates an ARP request packet for address resolution; (2) an SDN ingress switch receives and encapsulates the packet into a packet-in message and sends it to the controller; (3) the controller sends back an ARP reply packet that is encapsulated into a packet-out packet; (4) the ingress switch decapsulates the packet-out packet and sends the ARP reply packet to the host.

As the SDN proxy mechanism enforces all the ARP request/reply packets to be processed by the controller,[1] it can be utilized to transmit messages between hosts in SDN covertly. Specifically, as shown in Fig. 1, a host can generate a burst of ARP requests to increase packet processing delay on the controller or remain idle to signal a "1" or "0", respectively. Meanwhile, another host can sense the timing signal by generating ARP packets to measure the round-trip time (RTT) since the controller also processes its ARP packets. For example, if a host receives three ARP response packets with RTTs of 1 ms, 10 ms, and 3 ms, respectively, 3 bits of covert messages, represented as "010", can be interpreted.

To verify our crucial insight, we conduct experiments in a real SDN testbed consisting of commercial hardware

[1] It is similar to the NDP or DHCP request/reply packets.

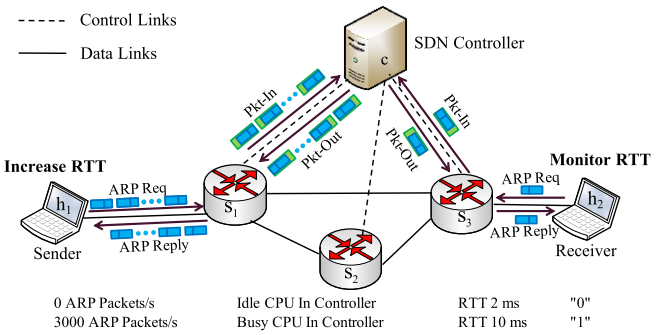


Fig. 1. The sender increases RTTs by generating ARP packets at a high speed while the receiver monitors the RTTs.

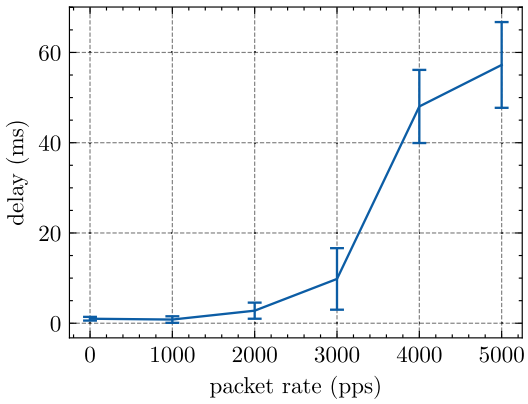


Fig. 2. The processing delay on the controller for different ARP packet generation rates.

SDN switches, EdgeCore AS4610-54T, and a popular open-source controller, RYU [36]. The controller runs on a server with 12 GB RAM and an Intel Xeon Quad-Core CPU E5504. We run the ARP proxy application [33] on the controller. We use a host in the network to generate ARP request packets at different rates. The experimental results are shown in Fig. 2. The delay fluctuates slightly when the generation rate of ARP request packets is small. For example, the delay stays below 5 ms when the rate is smaller than 1000 pps. However, the delay increases significantly when the speed exceeds a threshold. For example, the delay exceeds 10 ms when the rate reaches 3000 pps. Thus, a host can change its ARP generation rate and leverage the low and high RTTs to signal "0" and "1", respectively.

B. Covert Channel Construction

Based on our crucial insight that high-speed request packets triggering the SDN proxy mechanism would increase the processing delay, we can construct our covert timing channel between two hosts in SDN. To simplify, we will refer to the packets triggering the SDN proxy mechanism as *SDN Proxy Request packets*, specifically abbreviated as ***SPR packets***.

The whole process of constructing the covert channel can be divided into four steps. The first step is to probe feasible construction parameters. For example, we will measure the feasible rate at which SPR packets should be generated to effectively and consistently increase the processing delay. This

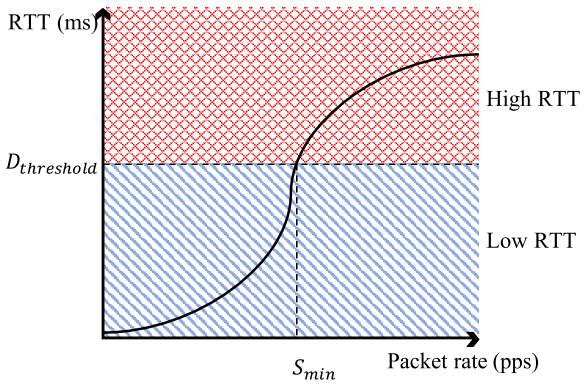


Fig. 3. The illustration of S_{min} and $D_{threshold}$. As the packet rate increases, the RTT increases. When the packet rate is below S_{min}, the RTT remains lower than $D_{threshold}$, indicating a low RTT. Conversely, when the packet rate exceeds S_{min}, the RTT surpasses $D_{threshold}$, signifying a high RTT.

step ensures that we can control the processing delays of the controller to transmit one-bit messages covertly. In the second step, we synchronize necessary information, such as the start time of the covert channel between the sender and the receiver host. Thereby, covert message can be encoded and decoded correctly. The sender encodes and sends the message in the last two steps by controlling the processing delay. Meanwhile, the receiver detects the delay sequence and decodes the message. Each step will be described in detail in the following subsections.

1) Probing Feasible Construction Parameters: In our covert channel, the RTTs of SPR packets serve as the message carrier. To ensure that the sender can generate higher-than-usual RTTs that represent "1", we need to know the rate at which we should generate SPR packets to archive this (as shown in Fig. 3). To ensure that the receiver can identify consistent and detectable increases in RTTs denoting "1", we must determine the duration for maintaining a continuous generation of SPR packets (as shown in Fig. 4). Hence, we can maintain an elevated RTT level for a brief period. Moreover, to transmit a sequence of covert messages, we need to ensure that the transmission of different bits of message does not interfere with each other. In this context, the generated high delays do not impact the subsequent message transmission. Consequently, we must determine how long we should wait for the delay to decrease from a high level to a normal range.

In summary, before we can transmit messages between hosts through the covert channel, there are mainly three parameters that must be probed: the minimum speed of generating SPR packets to cause a high delay denoted as S_{min}, the duration for which the sender generates SPR packets to ensure that messages can be received, denoted as δ_{data}, and the duration for which the sender remains idle to ensure that the current one-bit message's transmission does not affect the following one-bit message's transmission, denoted as δ_{idle}. Table I lists the related notations and meanings.

First, the sender needs to find S_{min} (as shown in Fig. 3), the minimum speed of generating SPR packets to cause higher-than-usual RTTs in the current network environment consistently. To increase the maximum transmission rate of

TABLE I
NOTATIONS USED IN THE COVERT CHANNEL

Notation	Meaning
S_{min}	The minimum speed of generating SPR packets to cause high delay
$D_{threshold}$	The threshold for SPR packet processing delay, above which the delay is considered "1", otherwise the delay is considered "0"
δ_{data}	The time for the sender to send a one-bit message by changing the RTTs
δ_{idle}	The duration the sender remains idle after changing the RTTs to ensure the correct sending of the next bit message

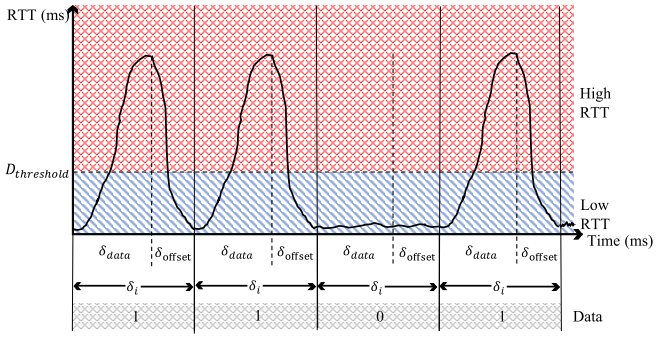


Fig. 4. The illustration of δ_{data} and δ_{idle}. The sender would increase the RTT for the duration of δ_{data} if the one-bit message is "1"; otherwise, the sender would not increase the RTT. Afterward, the sender will enter an idle state for the duration of δ_{idle}, during which it patiently waits for the RTT to revert to a low RTT.

the covert channel and to make the covert channel as covert as possible to avoid being detected by defense systems, the higher-than-usual RTTs, which represent "1", should be as low as possible. Based on our finding in Fig. 2 that a higher sending rate led to a higher delay, to make the higher-than-usual RTTs as low as possible, the sending rate S_{min} should be as low as possible.

The sender measures how the RTT changes as the SPR packet generating speed rises and finds S_{min}, the minimum speed of generating SPR packets to cause higher-than-usual RTTs consistently. First, the attacker detects the RTTs of SPR packets when there is no attack. Then, to determine the threshold for RTT, we employ a statistical approach based on the mean and standard deviation of the RTT distribution. The threshold $D_{threshold}$ is set at $\mu + 3\sigma$, where μ is the mean and σ is the standard deviation. This method is commonly known as the three-sigma rule in statistics. Approximately 99.7% of the data falls within three standard deviations of the mean in a normal distribution. Finally, the sender generates SPR packets at different speeds. Meanwhile, the sender counts the RTTs of SPR packets. Based on how the RTT changes as the SPR packet generating speed raises, the sender finds S_{min}. When generating SPR packets at speed S_{min}, the RTT is $D_{threshold}$. By setting the threshold $D_{threshold}$ at $\mu + 3\sigma$, we aim to capture the vast majority of delay values within an acceptable range, considering deviations beyond this threshold as potential outliers. This selection is reasonable as it provides a balance between accommodating the majority of delay variations expected in normal operation and identifying instances that deviate significantly from the norm. It helps in distinguishing delays that may indicate unusual network behavior or potential security threats.

Moreover, δ_{data} (as shown in Fig. 4), how long the sender keeps sending SPR packets at speed S_{min}, is vital for the covert channel. The δ_{data} should be big enough to increase the RTTs. Moreover, it takes time for the receiver to detect the high delays, which means the δ_{data} can not be too small. Meanwhile, a smaller δ_{data} led to a smaller δ_i, thus leading to a higher transmission rate. To determine an appropriate value for δ_{data}, the sender continuously transmits SPR packets at a rate of S_{min} for varying durations. Meanwhile, the sender starts a new procedure to detect how long the RTT of SPR packets remains bigger than $D_{threshold}$. We select the duration δ_{data} as the minimum duration which can ensure that the RTT remains bigger than $D_{threshold}$ for a duration exceeding S_{min}, making sure that the receiver can detect the RTTs steadily.

Finally, δ_{idle} (as shown in Fig. 4), the time that the sender remains idle to wait for the decrease of RTTs, should be chosen as an appropriate value. After the sender sends SPR packets at rate S_{min} for time δ_{data}, the sender should remain idle for time δ_{idle} to wait for the decreasing of RTTs. Otherwise, the transmission of a certain bit would be affected by the transmission of the preceding bits, and the covert channel's accuracy would decrease. To find a suitable δ_{idle}, the sender keeps sending SPR packets at speed S_{min} for time δ_{data}. Then, the sender starts a new procedure to detect how the RTTs of SPR packets decrease. The duration δ_{idle} is selected as the minimum duration that ensures the RTTs consistently decrease to a level below $\mu + \sigma$.

2) Time Synchronization: Time synchronization is critical for covert timing channels. If the time is not synchronized, the received covert timing messages may be decoded incorrectly.

For example, the sender may send messages one bit per second, and the receiver may try to detect the delays and receive messages in each second. However, there may be a significant offset between the sending and receiving times because of a lack of time synchronization. Due to significant deviations in sending time and receiving time, the receiver can not correctly receive the messages from the sender.

We assume both sender and receiver can access Network Time Protocol (NTP) servers to synchronize the clocks to achieve at least millisecond accuracy. Because it takes time for the sender to increase the RTTs and for the receiver to detect the high RTTs, there is still a short time offset between the sender sending a one-bit message and the receiver receiving it. We still need to make a fine-grained correction for the time δ_{offset}. To achieve this, the sender keeps sending "1" and the receiver tries to detect them with a receiving rate higher than the sending rate. The following example shows how synchronization works. During each second, the sender generates SPR packets from 0 ms to 10 ms at a high speed and remains idle for the remaining 990 ms. The receiver tries to detect the delay at a rate of 200 times per second and finds that the delay is higher than usual from the fifth to the fifteenth millisecond of each second ($\delta_{offset} = 5$ ms). Therefore, the receiver shifts the receive window by 5 ms and

receives messages 5 ms after the sender starts generating SPR packets at a high speed every time.

3) Encoding and Sending: In this step, according to the configuration synchronized in the previous steps (what the SPR packets generating speed is, what the sending rate is, etc.), the sender encodes the message to form the bitstream data and sends the bitstream data chronologically. We transmit messages on a bit-by-bit basis, where the transmission of each bit requires a duration denoted as δ_i, which we term a "time slot". This time slot encompasses two segments: δ_{data}, representing the time taken by the sender to transmit a one-bit message by altering the RTTs, and δ_{idle}, signifying the period during which the sender remains inactive after adjusting the RTTs. If the current bit message is "1", the sender would generate SPR packets at speed S_{min} that is high enough for a short duration δ_{data} to cause a delay higher than $D_{threshold}$, which can be easily detected by the receiver. Otherwise, the sender would remain idle for a short duration δ_{data}. Then, the sender remains idle for δ_{idle} to wait to send the following bit message. The meaning of these parameters and how to probe them are described in detail in subsection III-B.1.

After we finish probing these parameters, we can transmit as follows. The sender sends messages one bit by one bit. The sending of each bit costs duration δ_i, which consists of duration δ_{data} (the time for the sender to send a one-bit message by changing the RTTs) and duration δ_{idle} (the time the sender remains idle after changing the RTTs, to ensure the correct sending of the following bit message). If the current bit message is "1", the sender would generate SPR packets at speed S_{min} that is high enough for a short duration δ_{data} to cause a delay higher than $D_{threshold}$, which can be easily detected by the receiver. Otherwise, the sender would remain idle for a short duration δ_{data}. Then, the sender would remain idle for δ_{idle} to wait to send the following one-bit message.

4) Receiving and Decoding: In this step, the receiver needs to keep sending SPR packets to measure the current processing delay of SPR packets and record it for subsequent analysis of covert message. After recording the delay sequence, the receiver decodes it into a message based on the configuration in the previous steps. For each delay d_i in the delay sequence, if $d_i \geq D_{threshold}$, the decoded message $m_i = 1$; otherwise, the decoded message $m_i = 0$.

C. Evaluation

We conduct experiments in a real SDN testbed to evaluate our covert channels' accuracy, efficiency, stealthiness, and overhead.

1) Setup: Our SDN testbed consists of commercial hardware SDN switches, namely EdgeCore AS4610-54T, and a popular open-source controller, namely RYU. The controller runs on a server with 12 GB RAM and an Intel Xeon Quad-Core CPU E5504. The network topology of the testbed is shown in Fig. 5. The controller runs an ARP proxy application, which an attacker can exploit to build covert timing channels. A host named h_3 in the testbed applies Tcpreplay [37] to replay real traffic traces as background traffic, aiming at emulating a real-world network environment. As illustrated in Fig. 8b, when the transmission speed is set to 0, the packet-in

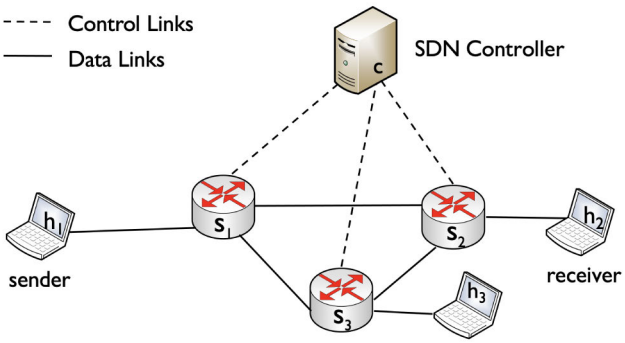


Fig. 5. The network topology of our SDN testbed. Host h_1 serves as the sender, host h_2 functions as the receiver and host h_3 generates background traffic using Tcpreply.

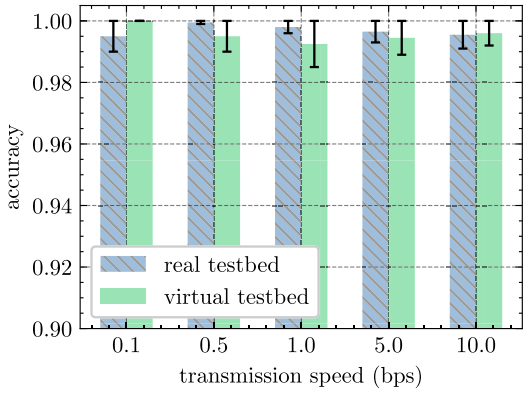


Fig. 6. The accuracy of the covert timing channel with different transmission rates.

speed fluctuates dynamically between 700 pps and 950 pps. These packet-in messages are generated in response to the varying background traffic, primarily comprising ARP traffic, that we introduced. This implies that the background traffic has introduced varying noise levels to the covert channel.

2) Accuracy: In our experiments, We probe these parameters as described in subsection III-B.1. When there is no covert channel, the RTTs of SPR packets do not exceed 5 ms. When we generate about 30 SPR packets in 10 ms, the RTTs would exceed 10 ms. The outcomes of the experimentation reveal that $S_{min} = 3000$ pps, $\delta_{data} = 10$ ms, and $D_{threshold} = 10$ ms. Since the parameter δ_{idle} is notably influenced by the maximum RTT and is directly responsible for determining the transmission rate, we have opted not to set a fixed value for δ_{idle}.

Under the condition that the host h_3 applies Tcpreplay to replay real traffic traces as background traffic, the message is transmitted through the covert channel at different transmission rates. The original message is compared with the message parsed by the receiver to calculate the transmission accuracy, and the experimental results are shown in Fig. 6. The results of our experiments indicate that when the transmission rate remains below the maximum transmission rate where the transmission of different bits of message would interfere with each other, the accuracy of the covert channel is not less than 98.5%. When the transmission rate is lower than the maximum transmission rate, the high delay caused by sending

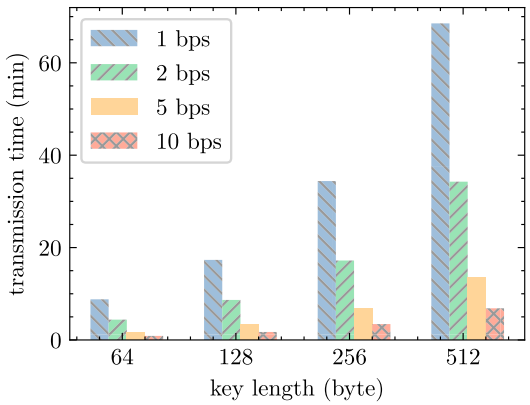
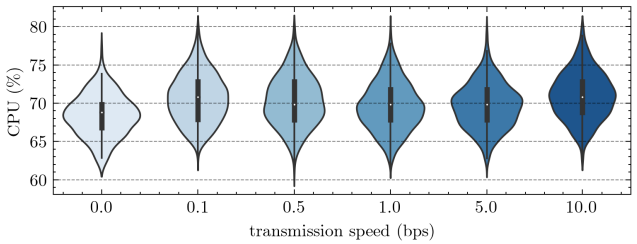
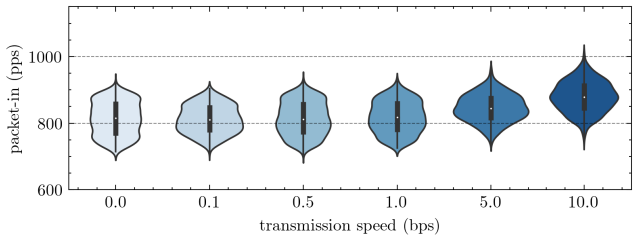


Fig. 7. The efficiency of our covert timing channels. We conduct experiments using covert channels of different rates to transmit TLS keys of different lengths.



(a) CPU usage



(b) Packet-in speed

Fig. 8. CPU utilization and control traffic throughput in the presence of covert channels of different rates.

a "1" can be reduced to a low delay within a time slot, and the transmission of messages in different time slots will not affect each other. Therefore, when the transmission rate is lower than the maximum transmission rate, the accuracy of the covert channel is irrelevant to the transmission rate. The background traffic essentially does not cause high delays as background traffic does not exceed the processing capacity. The experimental results demonstrate that in scenarios where only background traffic is present without any covert channels, there is approximately one high-delay sample point for every 300 delay sample points. Considering that only delays within a short duration, denoted as δ_{data}, carry the message and are minimally influenced by background traffic, it can be inferred that background traffic does not significantly impact the accuracy of covert channels.

However, when the transmission rate is higher than the maximum transmission rate, the delay cannot be reduced to a normal delay within a time slot after sending several "1", and the subsequent "0" will inevitably cause bit errors. Therefore, when the transmission rate is higher than the maximum transmission rate, the accuracy rate is largely affected by the continuous "1" in the message, which is unacceptable. Therefore, to ensure the accuracy of the covert channel, we should ensure that the transmission rate is not higher than the highest transmission rate. In our experiments, the RTTs of SPR packets may be increased to 50 ms sometimes so that the transmission rate cannot be higher than 20 bps.

3) Efficiency: To evaluate the efficiency of our covert channels, we perform experiments using covert channels of different rates to transmit TLS keys of different lengths. The result is shown in Fig. 7. The lengths of TLS keys are 64 bytes, 128 bytes, 256 bytes, and 512 bytes. The transmission rates of our covert channels are 1 bps, 2 bps, 5 bps, and 10 bps. The experiment result shows that when the transmission rate is constant, the transmission time is positively related to the length of the TLS keys. When the length of the TLS key is constant, the transmission time is negatively related to the transmission rate. By setting the transmission rate to 10 bps, we can successfully transmit a 512-byte TLS key within 10 minutes.

4) Stealthiness: For covert channels, stealthiness is relatively more important than the transmission rate. Since the communication between the two parties is monitored or restricted, and the message transmitted through the covert channel is often relatively private and of higher value, the two parties hope to use the covert channel to bypass the monitoring and restrictions for communication and do not want the communication to be detected. The controller's control flow throughput and CPU utilization are two indicators for judging whether the current working status is abnormal. Therefore, we conducted experiments to observe control traffic throughput and CPU utilization for the covert timing channels we constructed to evaluate their stealthiness. In the experiments, h_3 generates background traffic, and at the same time, a covert channel with different transmission rates is constructed between h_1 and h_2.

The results are shown in Fig. 8. As you can see, when the transmission rate is lower than 5 bps, the control flow throughput and CPU utilization are almost the same as they would be without the attack, and the covert channel is very well hidden. The transmission rate can be increased selectively and appropriately for SDN networks with low detection efforts.

The stealthiness of the covert channel is also related to the network environment. When the processing capacity of the controller is bigger, and the corresponding background traffic bandwidth is larger, the traffic generated by the covert channel accounts for a lower proportion of the total traffic. The covert channel is more difficult to detect. Our covert channels can hardly be detected by analyzing the controller's control flow throughput and CPU usage.

5) Overhead: The cost to the receiver of receiving one bit is to send an SPR packet to detect the RTT. The overhead of receiving one bit is approximately 42 bytes when we use ARP packets as SPR packets. The main overhead of the covert channel is on the sender. Although the sender must generate SPR packets at a high speed, we don't need to do this for

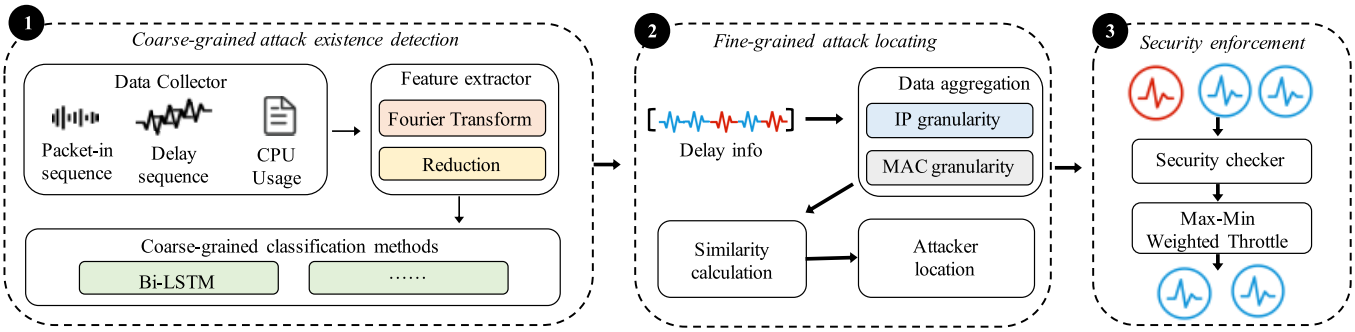


Fig. 9. Overview of CovertGuard.

too long. Experiments have shown that generating dozens of packets instantly can cause a significant increase in delay. So the overhead of sending one bit is not as significant as you might think, just dozens of packets. In our testbeds, the overhead of sending one bit is approximately 1 KB, about 25 SPR packets.

IV. Defense

To defend against covert timing channels between hosts in SDN, we have designed and implemented our defense system, CovertGuard, which exploits the delay and timing characteristics of covert channels to detect the covert channel, identify malicious hosts, and eliminate covert channels. In this section, we will first introduce the overall framework of CovertGuard and then introduce each module of it in detail.

A. Overview

There are severe challenges in the defense. First, in a real-world network environment, the background traffic is several orders of magnitude higher than the traffic associated with the covert timing channel. If we directly perform fine-grained detection of all traffic, the high-speed background traffic will introduce significant overhead to the detection, making it impossible to perform online detection. Second, the attackers can transfer information at different speeds. They can transfer several bits in a second, and they can also transfer one bit in a few minutes. We need to make the detection effective for detecting covert channels at any transmission rate. In addition, the attacker may compromise multiple hosts or forge the contents of the packet header. So, there is a big challenge in identifying the attacker accurately. Finally, we should not interfere with normal traffic when defending covert channels.

To address these challenges and make the defense efficient, we design CovertGuard as three main modules: detection, identification, and enforcement. The overall framework of CovertGuard is shown in Fig. 9. In the detection module, we collect the delay sequence and utilize machine learning methods to detect whether there are covert channels. Once a covert channel is detected, we identify the relevant flows and hosts based on finding the flows' cluster with the highest similarity to the channel among all the flows' clusters in the identification module. Finally, we dynamically restrict the relevant traffic from the relevant host to the controller. The detection module handles the first challenge and the second challenge, the identification module handles the third challenge, and the enforcement module handles the fourth challenge. We will illustrate how they work in detail in the following subsections.

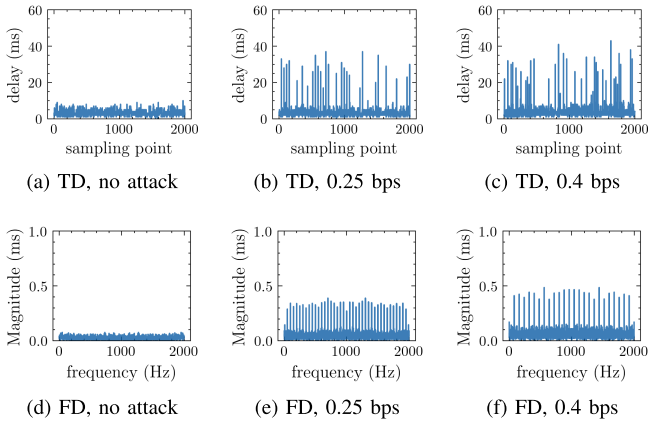
B. Detection

Compared to applying detection directly to all the traffic, filtering out most of the irrelevant traffic before detection can greatly reduce the overhead of detection. Instead of analyzing all the traffic directly, we can first perform a coarse-grained attack existence detection and perform fine-grained attack identification when covert channels are detected. The covert timing channels constructed in this paper have a timing effect on the RTTs of SPR packets. Therefore, we can judge whether there is a covert timing channel by whether the RTTs of SPR packets are sequential and sometimes exceed the normal delay range. Some samples are shown in Fig. 10. Fig. 10a and Fig. 10d show that when there is no covert channel, the RTTs fluctuate in small increments within the normal range (sometimes there may be occasional high RTTs due to bursts of traffic), with no significant peaks in the corresponding frequency domain. Fig. 10b, Fig. 10e, Fig. 10c, and Fig. 10f show that when there is a covert channel, the delay exceeds the normal delay range more frequently and has a timing pattern, with regular and distinct peaks in the corresponding frequency domain.

In the detection module, our solution is to collect the delay sequence at the controller and utilize bi-LSTM to detect the time domain information and frequency domain information of the delay sequences. The detection module provides a coarse-grained attack existence detection. If the detection result is that there are no covert channels, then further detection is not needed. Otherwise, we use corresponding traffic for fine-grained attack identification.

C. Identification

Once our detection module detects the presence of a covert timing channel, we need to identify the attacker efficiently. Although, thanks to the detection module, we only need to analyze packets when there is a covert timing channel, there is still too much background traffic. To make matters worse, the attacker may use multiple types of flows, use multiple hosts, or modify the contents of the packet headers.



(a) TD, no attack (b) TD, 0.25 bps (c) TD, 0.4 bps

(d) FD, no attack (e) FD, 0.25 bps (f) FD, 0.4 bps

Fig. 10. The time domain and frequency domain of delay samples. "TD" means time domain, "FD" means frequency domain. When there is no covert channel, the RTTs fluctuate in small increments within the normal range. When there is a covert channel, the delay exceeds the normal delay range more frequently and has a timing pattern, with regular and distinct peaks in the corresponding frequency domain.

To solve these challenges, we cluster flows and find the cluster with the biggest similarity with the covert channel. Since it is difficult to directly distinguish between normal traffic and malicious traffic based on a single packet, we cluster the flows at different granularities based on the incoming port, outgoing port, protocol type, etc. Then, we calculate the similarity between the delay sequence and the flow clusters. If the traffic within a cluster frequently appears when the delay exceeds the normal range and rarely appears when the delay is within the normal range, this cluster is highly correlated with the covert channel. Based on this idea, we first measure and record the delays within each time slot, forming the delay sequence $\mathbf{D}$. Then, we calculate the packet counts within a cluster for each time slot, forming the sequence $\mathbf{C}$. We define the similarity function [38] between the delay sequence $\mathbf{D}$ and the packet count sequence $\mathbf{C}$ of a cluster as follows:

$$S_{\boldsymbol{D},\boldsymbol{C}} = \frac{\sum_{i=0}^{n} D_i C_i}{\sqrt{\sum_{i=0}^{n} D_i^2}\sqrt{\sum_{i=0}^{n} C_i^2}} \tag{1}$$

Then, we can identify the attacker based on the cluster with the highest similarity to the covert channel. If there are multiple types of packets in the cluster, the attacker may utilize multiple types of SPR packets for the covert channel. If the packets in the cluster have different ingress ports, the attacker may control a few hosts for the covert channel. If the attack traffic has different IPs but has the same ingress port, it means the attacker may modify the contents of the packet header or control a few hosts that connect to the same switch. Based on the above information, we can find the hosts compromised by the attackers.

D. Enforcement

In this module, we destroy the covert channel with the least amount of effort. Enforcement should be as lightweight as possible and avoid adversely affecting normal traffic. The key to eliminating the covert channel is to disrupt the channels' timing or avoid causing high delays. The information cannot be transmitted accurately if the sender cannot cause distinguishably high delays. And the information cannot be received if the timing of the covert channel is disturbed. In our enforcement, we start from the source and try to avoid the emergence of high delays. Due to the limited resources of switches and the controller, we dynamically allocate the upstream(from the data plane to the controller) bandwidth of SPR packets to prevent attackers from maliciously causing high delays. Due to the possibility of misjudgment and the possibility of normal traffic from the attacking host, we do not directly and permanently block all traffic from the relevant hosts.

We have developed a dynamic bandwidth allocation algorithm. We use the Max-Min Weighted Fairness algorithm [39] to allocate the upstream bandwidth of each type of SPR packet on each port of edge switches. If covert timing channels are detected, we continuously halve the weight of the types of SPR packets related to the covert channel and reuse the Max-Min Weighted Fairness algorithm for bandwidth allocation until the high latency no longer occurs. For example, the weights of the SPR packet and the benign packet are 2 and 2, and the bandwidth requirements of the SPR packet and the benign packet are 80 Mbps and 40 Mbps while the bandwidth of the control path is 80 Mbps in total. In this case, we halve the weight of the SPR packet to 1. Since the weight of the benign packet is larger than the SPR packet, we would allocate 40 Mbps bandwidth to the benign packet and the SPR packet can only be allocated with 40 Mbps bandwidth. To ensure normal usage of the compromised hosts, we will remove the restrictions when covert channels disappear long enough or when we kick attackers out of the compromised hosts. We do not limit flows of other types from these compromised hosts, as these flows may be benign.

E. Experimental Result

1) Setup: The topology of our testbed is shown in Fig. 5. The host h_3 replays real traffic traces as background traffic, and host h_1 transmits messages to host h_2 through covert channels at different rates. We collect a few delay sequences (each delay sequence contains 200 delay samples) when there is no attack and when the attacker transmits messages at different rates. For each rate, we have collected hundreds of samples. Then, we get the frequency-domain dataset by Fourier transform. We combine the time-domain information and frequency-domain information and then split them into train datasets and test datasets. After that, we use these data for training our Bi-LSTM detection models and test these models on different rate data and full data.

2) Detection Result: As shown in Tabel II, Bi-LSTM performs well even when there is a significant difference in the transmission rate of train data and test data. We utilize precision, recall, and f1-score to evaluate these models. If the train data is 0.4 bps data, 1 bps data, or all the data, when testing on all the data, the precision, the recall, and the f1-score are all above 0.998. If the train data is 5 bps data, the result is good when testing on the same rate data, but the result

TABLE II
THE DETECTION RESULT OF UTILIZING BI-LSTM ON DIFFERENT TRAINING DATA FOR DETECTION

Train Data (bps)	Test Data (0.4 bps)			Test Data (1 bps)			Test Data (5 bps)			Test Data (Full)		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
0.4	1.0	1.0	**1.0**	1.0	1.0	**1.0**	1.0	1.0	**1.0**	0.995	0.995	0.995
1	0.958	0.955	0.955	0.995	0.995	0.995	0.995	0.995	0.995	0.985	0.984	0.984
5	0.235	0.485	0.317	0.235	0.485	0.317	1.0	1.0	**1.0**	0.886	0.281	0.286
Full	1.0	1.0	**1.0**	1.0	1.0	**1.0**	1.0	1.0	**1.0**	0.998	0.998	**0.998**

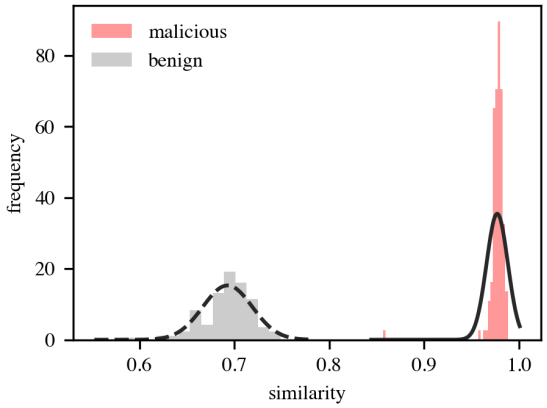


Fig. 11. The distribution of cosine similarity between covert channels and malicious or benign flows.

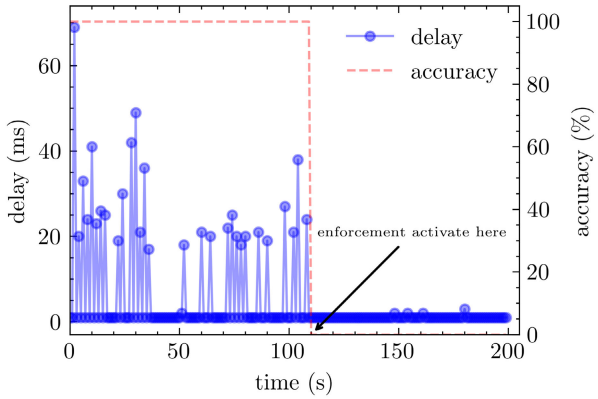


Fig. 12. The change of delay and transmitting accuracy.

is unacceptable when testing on 0.4 bps data, 1 bps data, or all the data. The result means training on higher rate data and testing on data of lower rate channels would get bad detection results. However, we can certainly get perfect detection results when we use data from low-rate channels or all the data for training.

3) Identification Result: We log the packet information (timestamp, ip src, ip dst, in port, etc.) on the controller and collect the delay sequence. Then, we cluster the data according to ip src, etc. Finally, we calculate the cosine similarity between the delay sequence and these clusters. This process has been done hundreds of times. The results show that we can clearly distinguish malicious flows from benign flows based on cosine similarity. As is shown in Fig. 11, the similarity between malicious flows and the covert channel is very close to 1, while the similarity between benign flows and the covert channel is smaller than 0.8. The similarity between malicious flows and the covert channel is concentrated and distributed between 0.95 and 1. The distribution of the similarity between malicious flows and covert channels is roughly a normal distribution. The similarity between benign flows and the covert channel is relatively scattered and distributed around 0.7. It should be noted that if the bandwidth of a flow is very low, the similarity between it and the covert channel will be even lower, close to 0. We can distinguish malicious flows from benign flows with 100% accuracy based on cosine similarity (even simply setting the threshold as 0.8 can achieve it).

4) Enforcement Result: After we detect the covert channel and identify the associated malicious traffic, we use the Max-Min Weighted Fairness algorithm to limit the bandwidth of the malicious flows. As shown in Fig. 12, when the enforcement is activated, the delay can not be increased by the malicious flows, and the transmitting accuracy of the covert channel drops to 0 instantly (the transfer does not work properly, and the accuracy is meaningless). Before we activate our enforcement, the delay of the sample points would be influenced by malicious flows, and the covert channel can transmit information accurately. At around 110 seconds, we activate our enforcement. The delays of the sample points did not change significantly anymore, and information could not be transmitted by changing the delay, which means that the covert channel was cut off.

5) Overhead: We evaluated the overhead introduced by CovertGuard across its key components: monitor, detection, location, and enforcement. The results, detailed in Table III, illustrate both CPU utilization and memory consumption for each component. The CPU utilization across CovertGuard components shows a relatively modest impact, with the detection component being the most resource-intensive at 13.23%. Furthermore, employing a GPU to execute the detection algorithm can significantly reduce CPU utilization, as GPUs are more adept at handling our Bi-LSTM detection algorithm. The monitor, location, and enforcement components exhibit lower CPU utilizations of 1.73%, 2.30%, and 0.56%, respectively. In terms of memory consumption, the detection component records the highest usage at 235.91 MB, largely due to runtime memory demands for intermediate computations, dependencies on deep learning libraries, and data handling. Given that the Bi-LSTM model and its parameters consume 32.4 MB, there is substantial potential to reduce the memory footprint of the detection component. Meanwhile,

TABLE III
THE OVERHEAD OF COVERTGUARD

Metric	CovertGurad			
	Monitor	Detection	Location	Enforcement
CPU Usage	1.73%	13.23%	2.30%	0.56%
Memory Usage	56.06 MB	235.91 MB	113.27 MB	0.13 MB

the monitor and location components use 56.06 MB and 113.27 MB, respectively, and the enforcement component maintains a minimal footprint of 0.13 MB. These findings demonstrate that although CovertGuard imposes some overhead, particularly through the detection component's reliance on the Bi-LSTM model, the overall impact on system resources remains manageable, supporting its deployment in large-scale SDN environments.

V. DISCUSSION

A. Limitations of CovertGuard

Covert timing channels operating at extremely low transmission rates may evade detection by CovertGuard. Consider a scenario where a covert channel transmits only one bit of information per second. In such cases, the resulting delay sequence may resemble that of normal traffic within a short detection window, potentially preventing CovertGuard from triggering an alert. However, this limitation could be addressed by implementing multiple detection windows in CovertGuard, each with varying granularities, to enhance detection accuracy. In addition to covert timing channels, there are covert storage channels [20], [40] in SDN. They embed messages into specific fields of packets without intentionally altering the round-trip times (RTTs) or inter-packet delays (IPDs). Thus, covert storage channels can not be detected by analyzing the timing characteristics of the RTTs or the IPDs. The detection of such covert storage channels, however, falls outside the scope of our current research. This area has been extensively covered in related studies [41], [42], [43], which explore various aspects and defenses.

B. Defense Against Adaptive Attackers

It is possible that attackers dynamically change the transmission rate during runtime within covert timing channels. Despite these modifications, attackers are still required to control the delays chronologically to transmit messages. Consequently, the frequency domain of the delay sequence will display periodic peaks, aiding in the differentiation of covert channels from normal traffic. CovertGuard has been trained to recognize patterns in both the frequency and time domains of delay sequences originating from normal and covert channel traffic. As a result, CovertGuard can effectively detect covert channels that dynamically adjust their transmission rates at runtime.

VI. RELATED WORK

A. Covert Channels in SDN

SDN proxy mechanisms and SDN rule expiry mechanisms are exploited to transmit information from the controller to switches [19]. SDN information, such as TLS Keys and Certifications, Routing Policies, and Network Topology, can be leaked from the controller to switches by being embedded in packets generated by the controller, and timing affects packet-out delay. With the help of the controller, information can be secretly transmitted between switches by being embedded in the packet-in messages [20]. Flow (re-)configurations and switch identification also provide opportunities for secret communication between switches [20], [21]. Switches can get information through whether a specific DPID has been used or there are specific flow (re-)configurations. Communication restrictions between hosts can be broken by adding a few OF flow rules [22]. As you can see, these works require the attacker to have more or less access to the switches or controller. Moreover, there is no work aimed at constructing covert timing channels between hosts in SDN. Our covert timing channels do not need any support from the switches or controller and help transmit information between hosts.

B. Covert Channels in Traditional Networks

Unused header fields may be utilized for transforming information in storage channels [40], [44], [45], [46]. The Type of Service (ToS) bits in the IP header are possible for carrying messages [47], as many networks never use them. An IP Identification field (IP ID) generation scheme based on the toral automorphism is designed to ensure that the modified field is random and embedding data in the IP ID field [45]. IP Fragment Offset field can transmit information covertly by modulating the size of the fragments [40]. Even TCP timestamps are used for storage channels [48]. The presence or absence of an IP packet in an interval conveys a one-bit message [49]. Zillien and Wendzel [50] modify the covert channel's behavior by dynamically choosing the inter-packet times, to escape from some recent covert channel detection methods [49], [51], [52], [53]. The attacker can increase the router's packet response time by interacting with a single shared router in a specific manner, thereby leaking sensitive information between two logically separated (or isolated) networks hosted by a single router [54]. Wang et al. [55] propose a blockchain covert communication model that utilizes a label tree to form a chained relationship between transactions and output addresses. Cloak [56] uses the different combinations of N packets sent over X flows in each round to represent a covert message. Luo et al. [57] proposes a method for designing TCP covert timing channels and discusses corresponding detection mechanisms.

C. Defense Against Covert Channels

Aiming at the regularity of covert channels, Serdar Cabuk et al. developed and tried two methods to detect covert timing channels in IP traffic [49]. The IPDs corresponding to simple covert timing channels tend to be concentrated around a few values, while the arrival time of normal traffic is often unlimited and may arrive at any time. In addition, the higher-order entropy and distribution of IPD can also be used for channel detection [41]. The corrected conditional entropy test can detect the covert timing channels with abnormal regularity,

while the entropy test can detect the covert timing channels with abnormal shape [58]. Fuzzing time is added to the delay to interfere with the timing of covert channels and affect the transmission of information [59].

Safety checks of SDN applications and control messages censorship on SDN controllers are two directions for mitigating the covert channels in SDN [19]. Covert channels constructed by the DPID can be deterred if OpenFlow connections are secured via a hardened authentication scheme [21]. Packet-in-Packet-out watcher, audit-trails and accountability, and enhanced IDS with waypoint enforcement are possible countermeasures [20]. However, our covert channel does not rely on changing the DPID or modifying the packet-in message or the packet-out message, thus these defenses can not discover or cut off our covert channel. Our defenses do not need a flow-by-flow inspection because we first do a coarse-grained detection. Subsequent identification modules can identify malicious flows that cause high delays in a more targeted manner.

VII. Conclusion

In this work, we present new covert timing channels between hosts in SDN, by utilizing the key insight that the delay of processing new SDN proxy packets would exceed the normal delay variation range when encountering flow processing bottlenecks. Our covert channels do not require any access to switches and controllers. They can help covertly transfer information between hosts even if traffic between the two hosts is monitored or blocked. We implement a prototype of our covert channels and present an extensive evaluation of the accuracy, efficiency, stealthiness, and overhead of it. Experimental results show that we can maintain more than 95% accuracy. To defend against our covert channels, we design and implement our defense system, named CovertGuard, which utilizes the latency and timing characteristics of covert channels to detect covert channels, identify the malicious hosts, and eliminate the covert channels. Our experimental results show that CovertGuard can detect and eliminate covert channels of different speeds in real time.

References

[1] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

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

[3] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Commun. Mag.*, vol. 51, no. 2, pp. 114–119, Feb. 2013.

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

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

[6] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in SDN-openflow networks," *Comput. Netw.*, vol. 71, pp. 1–30, Oct. 2014.

[7] A. O. Adedayo and B. Twala, "QoS functionality in software defined network," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2017, pp. 693–699.

[8] Y. Zhou et al., "A load balancing strategy of SDN controller based on distributed decision," in *Proc. IEEE 13th Int. Conf. Trust, Secur. Privacy Comput. Commun.*, Sep. 2014, pp. 851–856.

[9] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: A survey," in *Proc. IEEE SDN Future Netw. Services (SDN FNS)*, Nov. 2013, pp. 1–7.

[10] M. Cicioglu and A. Çalhan, "HUBsFLOW: A novel interface protocol for SDN-enabled WBANs," *Comput. Netw.*, vol. 160, pp. 105–117, Sep. 2019.

[11] C. G. Girling, "Covert channels in LAN's," *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 2, p. 292, Feb. 1987.

[12] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-based survey and categorization of network covert channel techniques," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 1–26, Apr. 2015.

[13] J. C. Wray, "An analysis of covert timing channels," *J. Comput. Secur.*, vol. 1, nos. 3–4, pp. 219–232, Oct. 1992.

[14] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: High-bandwidth and reliable covert channel attacks inside the cloud," *IEEE/ACM Trans. Netw.*, vol. 23, no. 2, pp. 603–615, Apr. 2015.

[15] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Commun. Surveys Tuts.*, vol. 9, no. 3, pp. 44–57, 3rd Quart., 2007.

[16] Z. Hu, M. Wang, X. Yan, Y. Yin, and Z. Luo, "A comprehensive security architecture for SDN," in *Proc. 18th Int. Conf. Intell. Next Gener. Netw.*, Feb. 2015, pp. 30–37.

[17] S. T. Yakasai and C. G. Guy, "FlowIdentity: Software-defined network access control," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2015, pp. 115–120.

[18] J. Matias, J. Garay, A. Mendiola, N. Toledo, and E. Jacob, "FlowNAC: Flow-based network access control," in *Proc. 3rd Eur. Workshop Softw. Defined Netw.*, Sep. 2014, pp. 79–84.

[19] J. Cao et al., "Covert channels in SDN: Leaking out information from controllers to end hosts," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Cham, Switzerland: Springer, 2019, pp. 429–449.

[20] K. Thimmaraju, L. Schiff, and S. Schmid, "Outsmarting network security with SDN teleportation," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS P)*, Apr. 2017, pp. 563–578.

[21] R. Krösche, K. Thimmaraju, L. Schiff, and S. Schmid, "I DPID it my way! A covert timing channel in software-defined networks," in *Proc. IFIP Netw. Conf. (IFIP Netw.) Workshops*, May 2018, pp. 217–225.

[22] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw.*, Aug. 2012, pp. 121–126.

[23] H. Hu, G.-J. Ahn, W. Han, and Z. Zhao, "Towards a reliable SDN firewall," in *Open Networking Summit 2014*. Santa Clara, CA, USA: USENIX Association, 2014.

[24] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with SDN: A feasibility study," *Comput. Netw.*, vol. 85, pp. 19–35, Jul. 2015.

[25] Z. Abaid, M. Rezvani, and S. Jha, "MalwareMonitor: An SDN-based framework for securing large networks," in *Proc. CoNEXT Student Workshop*, Dec. 2014, pp. 40–42.

[26] D. Firestone, "VFP: A virtual switch platform for host SDN in the public cloud," in *Proc. NSDI*, vol. 17, 2017, pp. 315–328.

[27] S. Rees-Pullman, "Is credential stuffing the new phishing?" *Comput. Fraud Secur.*, vol. 2020, no. 7, pp. 16–19, Jan. 2020.

[28] S. Gupta, A. Singhal, and A. Kapoor, "A literature survey on social engineering attacks: Phishing attack," in *Proc. Int. Conf. Comput., Commun. Autom. (ICCCA)*, Apr. 2016, pp. 537–540.

[29] J.-H. Lam, S.-G. Lee, H.-J. Lee, and Y. E. Oktian, "Securing distributed SDN with IBC," in *Proc. 7th Int. Conf. Ubiquitous Future Netw.*, Jul. 2015, pp. 921–925.

[30] M. Sneps-Sneppe and D. Namiot, "Metadata in SDN API for WSN," in *Proc. 7th Int. Conf. New Technol., Mobility Secur. (NTMS)*, Jul. 2015, pp. 1–5.

[31] T. Alharbi and M. Portmann, "SProxy ARP–efficient ARP handling in SDN," in *Proc. 26th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Dec. 2016, pp. 179–184.

[32] *Onos ARP Proxy*. Accessed: Mar. 25, 2023. [Online]. Available: https://github.com/opennetworkinglab/onos/tree/master/apps/proxyarp

[33] *Floodlight ARP Proxy*. Accessed: Mar. 25, 2023. [Online]. Available: https://github.com/mbredel/floodlight-proxyarp

[34] *Onos Neighbour Resolution Service*. Accessed: Mar. 25, 2023. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Neighbour+Resolution+Service

[35] *Floodlight DHCP Proxy*. Accessed: Mar. 25, 2023. [Online]. Available: https://github.com/floodlight/floodlight/tree/master/src/main/java/net/floodlightcontroller/dhcpserver

[36] *RYU Controller*. Accessed: Mar. 25, 2023. [Online]. Available: https://ryu.readthedocs.io/en/latest/

[37] K. Fred and AppNeta. *Tcpreplay—PCAP Editing and Replaying Utilities*. Accessed: Mar. 25, 2023. [Online]. Available: https://tcpreplay.appneta.com/

[38] A. Huang, "Similarity measures for text document clustering," in *Proc. 6th New Zealand Comput. Sci. Res. Student Conf.*, Christchurch, New Zealand, vol. 4, 2008, pp. 9–56.

[39] P. Marbach, "Priority service and max-min fairness," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Societies*, vol. 1, Jun. 2002, pp. 266–275.

[40] S. J. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Proc. Int. Workshop Inf. Hiding*. Cham, Switzerland: Springer, 2005, pp. 247–261.

[41] J. Xing, A. Morrison, and A. Chen, "NetWarden: Mitigating network covert channels without performance loss," in *Proc. 11th USENIX Workshop Hot Topics Cloud Comput. (HotCloud)*, 2019, pp. 1–7.

[42] C.-R. Tsai, V. D. Gligor, and C. S. Chandersekaran, "On the identification of covert storage channels in secure systems," *IEEE Trans. Softw. Eng.*, vol. 16, no. 6, pp. 569–580, Jun. 1990.

[43] S. Cabuk, "Network covert channels: Design, analysis, detection, and elimination," Ph.D. dissertation, Purdue Univ., West Lafayette, IN, USA, 2006.

[44] K. Ahsan, "Covert channel analysis and data hiding in TCP/IP," M.S. thesis, Univ. Toronto, Toronto, ON, Canada, 2002.

[45] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Proc. Workshop Multimedia Secur. ACM Multimedia*, vol. 2, no. 7. New York, NY, USA: ACM Press, 2002, pp. 1–8.

[46] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, "Eliminating steganography in internet traffic with active wardens," in *Proc. Int. Workshop Inf. Hiding*. Cham, Switzerland: Springer, 2002, pp. 18–35.

[47] T. G. Handel and M. T. Sandford, "Hiding data in the OSI network model," in *Proc. Int. Workshop Inf. Hiding*. Cham, Switzerland: Springer, 1996, pp. 23–38.

[48] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibetts, "Covert messaging through TCP timestamps," in *Proc. Int. Workshop Privacy Enhancing Technol.* Cham, Switzerland: Springer, 2002, pp. 194–208.

[49] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert timing channels: Design and detection," in *Proc. 11th ACM Conf. Comput. Commun. Secur.*, Oct. 2004, pp. 178–187.

[50] S. Zillien and S. Wendzel, "Weaknesses of popular and recent covert channel detection methods and a remedy," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 6, pp. 5156–5167, Feb. 2023.

[51] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert channel detection," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 4, pp. 1–29, Apr. 2009.

[52] H. Li, T. Song, and Y. Yang, "Generic and sensitive anomaly detection of network covert timing channels," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 5, pp. 4085–4100, Sep. 2022.

[53] S. Al-Eidi, O. Darwish, Y. Chen, and G. Husari, "SnapCatch: Automatic detection of covert timing channels using image processing and machine learning," *IEEE Access*, vol. 9, pp. 177–191, 2021.

[54] O. Shvartzman et al., "Characterization and detection of cross-router covert channels," *Comput. Secur.*, vol. 127, Apr. 2023, Art. no. 103125.

[55] Z. Wang et al., "A covert channel over blockchain based on label tree without long waiting times," *Comput. Netw.*, vol. 232, Aug. 2023, Art. no. 109843.

[56] X. Luo, E. W. W. Chan, P. Zhou, and R. K. C. Chang, "Robust network covert communications based on TCP and enumerative combinatorics," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 6, pp. 890–902, Nov. 2012.

[57] X. Luo, E. W. W. Chan, and R. K. C. Chang, "TCP covert timing channels: Design and detection," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. FTCS DCC (DSN)*, Jun. 2008, pp. 420–429.

[58] S. Gianvecchio and H. Wang, "An entropy-based approach to detecting covert timing channels," *IEEE Trans. Dependable Secure Comput.*, vol. 8, no. 6, pp. 785–797, Nov. 2011.

[59] W.-M. Hu, "Reducing timing channels with fuzzy time," *J. Comput. Secur.*, vol. 1, nos. 3–4, pp. 233–254, Oct. 1992.

Yixiong Ji (Graduate Student Member, IEEE) received the B.Eng. degree from Tsinghua University in 2020, where he is currently pursuing the Ph.D. degree in computer science and technology. His research interests include network security, software-defined networking, and machine learning for security.

Jiahao Cao (Member, IEEE) received the B.Eng. degree from Beijing University of Posts and Telecommunications in 2015 and the Ph.D. degree from Tsinghua University in 2020. He was a Visiting Scholar with George Mason University. He is currently an Assistant Research Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His current research interests include network protocol security, network attack detection, and network traffic analysis.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. He is an Editorial Board Member of IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and ACM TOPS.

Yan Liu is currently a Researcher and the Director of Ant Group. Before that, he was a Senior Researcher with Baidu, in charge of the AI Security Team. His research interests include trusted AI, networking security, and private computing. He works on broad applications of machine learning and privacy computing technologies in enterprise security.

Tao Wei received the B.S. and Ph.D. degrees from Peking University, Beijing, China, in 1997 and 2007, respectively. He is currently the Vice President of Ant Group and an Adjunct Professor with Peking University. He is in charge of foundational security with Ant Group. His work has helped Windows, Android, iOS, and other operating systems improve their security capabilities.

Ke Xu (Fellow, IEEE) received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has published more than 200 technical articles and holds 11 U.S. patents in the research areas of next-generation internet, blockchain systems, the Internet of Things, and network security. He is a member of ACM. He was the Steering Committee Chair of IEEE/ACM IWQoS. He has guest-edited several special issues in IEEE and Springer journals. He is an Editor of IEEE INTERNET OF THINGS JOURNAL.

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