Received 23 November 2022, accepted 13 December 2022, date of publication 15 December 2022, date of current version 20 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3229899

RESEARCH ARTICLE

LINK-GUARD: An Effective and Scalable Security Framework for Link Discovery in SDN Networks

ISMAIL AL SALTI AND NING ZHANG
Department of Computer Science, The University of Manchester, M13 9PL Manchester, U.K.
Corresponding author: Ismail Al Salti (ismail.alsalti@postgrad.manchester.ac.uk)

This work was supported by The University of Manchester.

ABSTRACT Software-Defined Networking (SDN) is an emerging networking paradigm that creates new opportunities for future generations of networks. The main characteristic of SDN is its ability to centralise control through the decoupling of control decisions from the network switches to make the network more flexible, programmable, and scalable. As part of this centralised control management, the SDN controller maintains a holistic view of the underlying network. Therefore, topology discovery in SDN is an essential service for topology-aware applications, such as routing, load balancing, mobility, and tracking. However, during the SDN topology discovery process, the controllers, without proper protection, are vulnerable to topology poisoning attacks, most notably Link Fabrication Attacks (LFAs). LFAs may be mounted due to a leak of packet source authentication, the lack of packet integrity checks, or the reuse of static packets. In this paper, we describe an effective and scalable security framework, LINK-GUARD, used for facilitating secure link discoveries in an SDN network. LINK-GUARD is designed to detect and thwart LFAs, thus reducing the risks of network topology poisoning. The framework has been implemented and evaluated on a Mininet emulator with an RYU controller. The security analysis indicates that LINK-GUARD can effectively and efficiently secure topology discoveries against both host-based and switch-based link fabrication attacks. Performance evaluation results show that the legitimacy of new links can be verified nearly real-time, taking approximately 30 milliseconds, and fake links can be detected within as low as 6 milliseconds, with a negligible runtime overhead. These results show that LINK-GUARD is a scalable solution for dynamic and large SDN networks.

INDEX TERMS Software-defined networking (SDN), topology discovery, OpenFlow protocol, topology poisoning, link fabrication attacks.

I. INTRODUCTION

In recent years, with the rapid development of mobility, cloud computing, virtualisation, and multi-tenant networks, it has become increasingly challenging to manage traditional networks. To address the challenges, the concept of a programmable network has been proposed [1] and this leads to the advent of the Software-Defined Networking (SDN) architecture which uses centralised control and allows open programming.

The SDN paradigm has a number of characteristics [2] such as its ability to decouple control decisions from the forwarding plane to make networks more flexible, programmable, scalable, and subject to centralised control [3]. The SDN technology is increasingly being adopted in Data Centre (DC) networks and Wide Area Networks (WANs). For instance, it has been adopted in Google data centres [4] to interconnect their data centre networks around the globe.

As the brain of an underlying network, the SDN logically centralises the control plane in an entity called the SDN controller, which interacts with and manages the underlying network infrastructure. The controller maintains a holistic view of the network by collecting topology information from the SDN switches. In this way, it is possible to retain a global view of the entire underlying network topology. The global view entails hosts, switches, and links between any pair of

The associate editor coordinating the review of this manuscript and approving it for publication was Ghufran Ahmed.

switches. To ensure the proper functioning of the SDN core services and applications, including routing, host migration tracking, load balancing and topology-based slicing, this global view must be up-to-date [5]. For this reason, the SDN controller uses a topology discovery mechanism for network topology discoveries and maintenance [6].

The Link Discovery Service (LDS) is one of the SDN controller core services that play a vital role in network topology discoveries. During the link discovery process, the controllers, without proper protection, are vulnerable to topology poisoning attacks, most notably, Link Fabrication Attacks (LFAs). LFAs are mounted by exploiting LDS vulnerabilities through compromised hosts or switches. If an LDS fails to guarantee the authenticity of the link discovery packets or if the propagation path taken by link discovery packets cannot be verified, the risks of LFA and other types of topological poisoning attacks will be high [7].

A. MOTIVATION AND CONTRIBUTIONS

Currently, most mainstream SDN controllers adopt the OpenFlow Discovery Protocol (OFDP) to perform network link discoveries [8]. OFDP leverages the traditional Link Layer Discovery Protocol (LLDP) with a few modifications to the packet format and operations to be compatible with the SDN architecture [9]. Each controller implements a variant of an LLDP packet. A link discovery process is vulnerable to LFAs if the link discovery packets are not authenticated or if the propagation paths taken by the link discovery packets are not verified. To reduce the risk of attacks, some security extensions can be used, e.g. a hash field which is added to each packet for integrity check of the packet [10]. Similar methods have also been proposed in the research domain. For example, TopoGuard [7] and [11] propose the use of a keyed hash in an LLDP packet to protect its authenticity. There are also other proposals. For example, in SLDP [10] and ESLD [12], LLDP packets should only be sent to non-host ports,i.e. ports connected to an OpenFlow switch, thus reducing the number of LLDP packets and preventing the relaying of LLDP packets from compromised host ports. The sOFTDP [13] is a novel security protocol designed to reduce the control load and improve the security of the topology discovery mechanism in an SDN controller. More detailed discussions of related SDN topology security solutions are given in section V.

However, these prior works have limitations. First, they are largely designed to counter only some of the security threats seen in a network topology discovery process, e.g., LLDP fabrication and LLDP relay attacks. When faced with other threats, e.g., LLDP flooding, and port amnesia attacks, they are not as effective. The LLDP flooding attack exhausts controller resources and consumes the bandwidth of the switch to the controller channel. Therefore, it negatively affects benign packet delivery, resulting in the link's removal from the topology database. Port amnesia attacks, on the other hand, enable the attacker to reset the port behavioural profiling based on the first seen packet. Thus, an attack can lead to relay-type LFAs. Second, adding a unique hash value to every LLDP packet incurs a non-negligible amount of processing overhead and time consumption in the SDN controller. Specifically, in large-scale networks with tens of thousands of active ports, every discovery cycle requires the SDN controller to generate an LLDP packet with a unique hash value for each active port in the network. Additionally, the controller must track and match every unique hash value for the purpose of packet authentication. Third, most existing solutions are designed under the assumption that only the hosts may be compromised. However, in reality, switches are also vulnerable; they may also be compromised. A compromised switch can also poison the network topology view. Therefore, how to systematically thwart multiple types LFAs effectively, but with as low overloads as possible, is still an open research issue.

As part of our efforts on tackling this issue, this paper describes the design and evaluation of LINK-GUARD, a novel security framework for link discovery in SDN networks. The novelty of LINK-GUARD lies in that it does not assume that all network switches are trustworthy. It can counter LFAs from both compromised hosts and compromised network switches systematically. It achieves this by employing three novel detection methods. First, the Bidirectional Link Verification (BLV) method based on two-way link direction verification is used to detect LLDP injection attacks using compromised hosts. Second, the Link Latency Measurement (LLM) method with a statistical analysis technique detects host- and switch-based LLDP relay attacks. Third, for LLDP flooding attacks detection, we used the Per-port LLDP Packet Counter (PLPC) method, which is based on counting the number of LLDP packets received from each port in each discovery round. In addition, LINK-GUARD provides a mitigation method in case of the detection of fake links based on port blocking. This study makes the following significant contributions:

- We implemented LFAs (fake LLDP injection, LLDP relay, and LLDP flooding attacks) over the current version of the mainstream controllers. In addition, we analysed these controllers' security measures to detect and prevent these attacks.
- We propose an effective and scalable security framework called LINK-GUARD to improve the security of the topology discovery mechanism. LINK-GUARD presents simple and effective novel detection methods for host and switch-based LFAs. In addition, it provides a mitigation method to prevent relaunching LFAs.
- We implemented LINK-GUARD on the Mininet emulator with the RYU controller. We conducted a security analysis to evaluate LINK-GUARD effectiveness against LFAs under different attack scenarios. In addition, we evaluated the performance overhead introduced by LINK-GUARD over different network scales. The experimental results were compared with those of the latest related works.

The rest of the paper is organised as follows. Section II presents an overview of OpenFlow-based Software Defined Networks (SDN) and their topology discovery mechanism.

In Section III, we analyse the LFAs in different scenarios. Requirements specification described in section VI. Section V comprehensively analyses the existing solutions and their limitations. In Section VI, we present the details of our proposal (LINK-GUARD). The experimental setup is presented in Section VII. In Section VIII, we evaluate the effectiveness of LINK-GUARD against LFAs. In Section IX, the performance of LINK-GUARD is evaluated in terms of the link validation delay, resource consumption, and detection rate. In Section X, we discuss the study limitations and future research directions. Finally, the conclusions are drawn in Section XI.

II. BACKGROUND

This section provides an overview of OpenFlow-based Software Defined Networks (SDN) and their topology discovery mechanism, known as the OpenFlow Discovery Protocol (OFDP).

A. SOFTWARE-DEFINED NETWORKING

Software-Defined Networking (SDN) is a new programmable network framework that separates the control plane from the data plane, allowing a single control plane to handle multiple devices. Plane separation allows networking devices to become simple forwarding units governed by a logically centralised controller, which serves as the network's operating system.

The SDN architecture consists of three layers: application, control, and infrastructure [2], [14], [15], [16]. Each has its own functional sub-layers and communication interfaces, as shown in Figure 1.

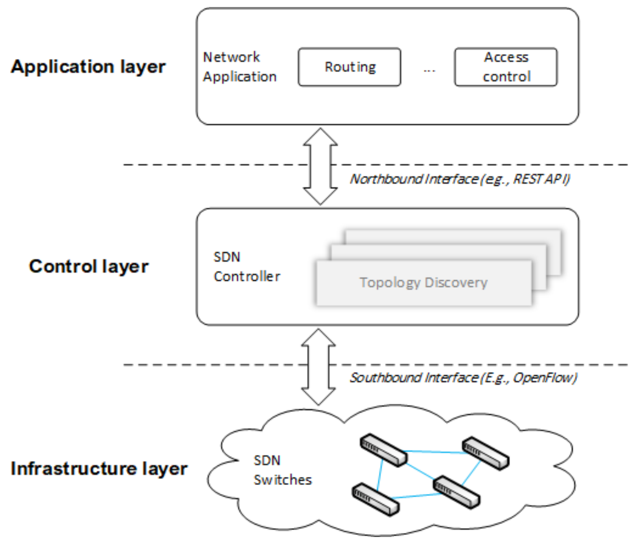


FIGURE 1. General SDN layer architecture.

The application layer is comprised of several applications that manage the entire data plane via a control layer. These applications include routing, access control, load balancing, and topology-based slicing. The control layer consists of the centralised SDN controller software, which serves as the network's brain. The infrastructure layer consists of the data-forwarding devices (such as switches and routers) that route the data packets based on the forwarding instructions received from the SDN controller.

Moreover, the application and control layers communicate via an unstandardised northbound API. Currently, the Representational State Transfer (REST) protocol appears to be the most popular northbound interface [17]. The OpenFlow protocol, on the other hand, is the most widely used southbound API for communications between the controller and infrastructure plane devices [18].

B. OpenFlow

OpenFlow protocol, as specified by ONF [19], is a communication protocol between OpenFlow controllers and Openflow forwarding devices. The OpenFlow protocol permits the SDN controller to instruct OpenFlow-enabled switches on how to handle different types of incoming packets. In addition, it provides a secure communication channel between the controllers and switches for events and statistical information.

The OpenFlow switch consists of two components: (1) An OpenFlow channel that is used to communicate with the SDN controller over a secure channel Transport Layer Security (TLS) on Transmission Control Protocol (TCP) port 6653 [20]; and (2) An OpenFlow flow table, composed of flow entries that specify packet match conditions and resulting actions. The SDN controller collects the topology information and forms a global view of the underlay network topology by exchanging Packet-In and Packet-out messages with OpenFlow switches.

C. TOPOLOGY DISCOVERY SERVICE

One of the essential functions of the SDN controller is to provide an accurate, near real-time view of the underlying network topology to the application plane services. A routing service, for example, requires the network topology to route the network traffic to its destination.

Topology discovery is a process used by the controller to learn about the three main network entities: hosts, network equipment (e.g., switches), and the inter-connected links between the switches. The SDN controller discovers the actual location of the hosts within the network by utilising the Host Tracking Service (HTS) [7]. OpenFlow switches are discovered during the initial handshake process with the controller. The links between switches are discovered and tracked by a Link Discovery Service (LDS) [7]. LDS can dynamically discover network links by leveraging the OpenFlow Discovery Protocol (OFDP).

OFDP is considered to be a de facto protocol for link discovery in current mainstream SDN controllers [9], [21]. The OFDP adopts the layer 2 Link Layer Discovery Protocol (LLDP) with a few modifications to the protocol operation for compatibility with the SDN architecture. Figure 2 shows the format of the LLDP packet, which is divided into the header and payload. The header consists of a destination address, source address, and Ethernet type. The payload of the LLDP packet consists of a different set of Type-length value (TLV)

Header			Payload				
Destination MAC	**Source MAC**	**Ethernet Type**	**Chassis ID TLV**	**Port ID TLV**	**TTL TLV**	**Optional TLVs**	**End TLV**
01:80:C2:00:00:0E	Outgoing Port MAC	0X88CC	Local switch ID	Sending Port ID	Time to Live	E.g., System Description	End Signal of LLDP

FIGURE 2. The format of LLDP packets.

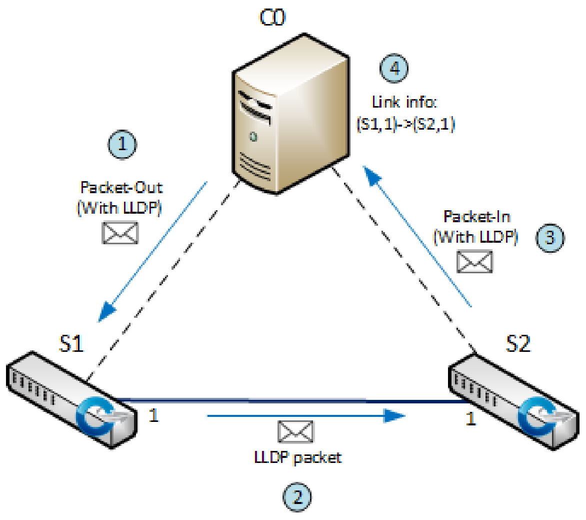


FIGURE 3. Discovering a unidirectional link from S1 to S2 using OFDP.

fields. Some controllers maintain a distinct set of TLVs. The Chassis ID, Port ID, and Time To Live (TTL) TLVs are used to store the switch data path id (dpid), port number, and timestamp, respectively. The Optional TLVs store additional information that is not required for the topology discovery process.

Figure 3 illustrates the discovery process of the unidirectional link between the two OpenFlow switches (denoted by S1 and S2). The discovery process can be divided into four steps.

Step 1: The SDN controller C0 requests all the switch S1 active ports. Subsequently, controller C0 encapsulates the LLDP packet inside a Packet-Out message for each active port in S1. After that, the controller sends them to switch S1.

Step 2: After the Packet-Out message reaches switch S1, the LLDP packet is forwarded to a specific output port (port 1).

Step 3: Upon receiving the LLDP packet, switch S2 encapsulates the LLDP packet as a payload into a Packet-In message and forwards it to controller C0.

Step 4: Controller C0 receives a Packet-In message with the meta-data of the destination dpid and destination port number. Based on the LLDP payload and meta-data, the LDS can discover a unidirectional link from switch S1 to switch S2.

Most SDN controllers can discover a bidirectional link (two-way link direction) by simply repeating the same process in the reverse direction [22].

III. TOPOLOGY POISONING ATTACKS

During the SDN topology discovery process, controllers, without proper protection are vulnerable to topology poisoning attacks, most notably Link Fabrication Attacks (LFAs). An attacker's aim with LFAs is to mislead the controller into adding fake links to the network topology. An LFAs is mounted by compromising the switches or hosts that exploit LDS vulnerabilities. For instance, if an SDN controller fails to guarantee the authenticity of link discovery packets to prevent any modification. In addition, if the controller is unable to verify the propagation path of the link discovery packets to avoid any host port involved in the process [7].

LFAs are classified into three types: fake LLDP injection, LLDP relay, and LLDP flooding attacks. An adversary can create a fake link by injecting malicious LLDP packets into the network to mislead the controller into adding a fake link to the topology view. An adversary can relay LLDP packets between two switches by taking advantage of the controller LLDP broadcast to create fake links. In addition, an adversary aims to use an LLDP flooding attack to consume the SDN controller's computing resources.

This section describes LFAs and possible attack implementation scenarios. In addition, we analysed the security measures taken by different mainstream controllers to avoid LFAs. Furthermore, we analysed the effects of LFAs on the SDN architecture layers. Our study mainly focused on the LFAs initiated by compromised hosts and switches. We separately analysed the three attack types: fake LLDP injection, LLDP relay, and LLDP flooding attacks.

A. FAKE LLDP INJECTION ATTACK

Attackers can create fake links by injecting malicious LLDP packets into the network. Simultaneously, the controller can not verify the legitimacy of the LLDP packet. There are two possible scenarios for creating fake links by injecting forged LLDP packets, as follows.

Scenario 1 (Injecting Fake LLDP Packets Using a Single Compromised Host): In this scenario, we assume that only a single compromised host generates and injects a modified LLDP packet into an interface, which is directly connected to an OpenFlow switch. To simulate a fabricated LLDP injection attack using a single compromised host as shown in Figure 4 (a), the attack can be divided into the following steps:

1) The compromised host H1 connected to switch S1 via port 1 sniffs the LLDP packet sent from the controller. Different third-party open-source tools are used to capture the network traffic, such as TCPdump [23].

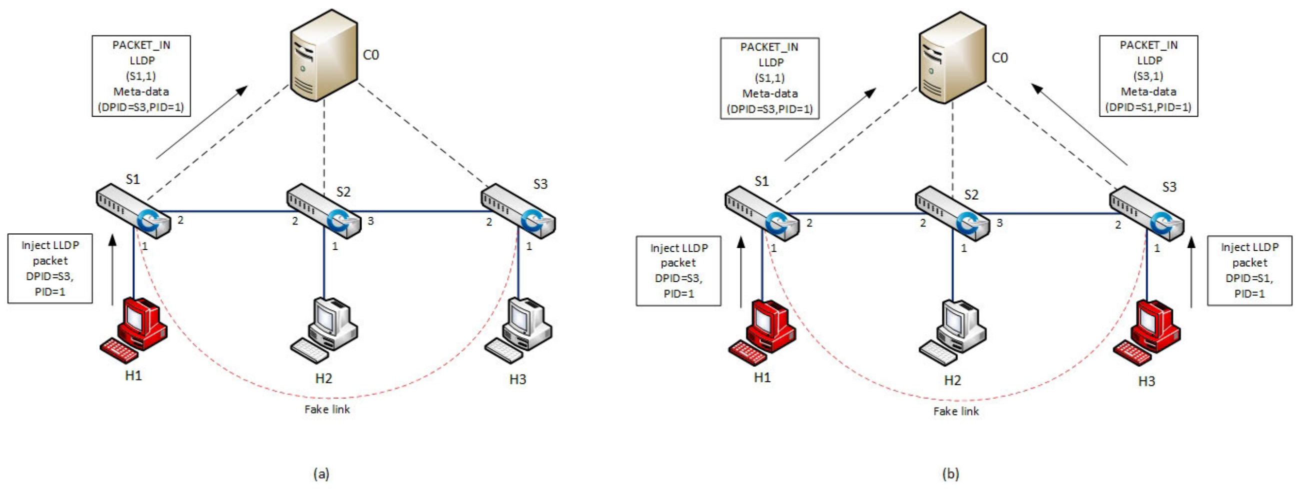


FIGURE 4. Fake LLDP injection attacks. (a) injecting fake LLDP packets using a single compromised host. (b) injecting fake LLDP packets using two compromised hosts.

2) The attacker changes the Chassis ID TVL to S3 and Port ID to 1. The malicious LLDP packet follows the same format as the given controller. Host H1 then injects a modified LLDP packet into port 1 of switch S1.
3) When switch S1 receives the forged LLDP packet, it forwards it to the controller without noticing any abnormal activity in the network. Switch S1 treats this packet as if it comes from a switch connected to port 1 and forwards it to the controller after encapsulating it in a Packet-In message with the ingress information as (S1,1).
4) After the controller receives the forged LLDP packet, it determines that a unidirectional link exists between S1 and S3. Subsequently, the controller updates the overall network topology.

Scenario 2 (Injecting a Fake LLDP Packet Using Two Compromised Hosts): In this scenario, we assume that there are two compromised hosts that generate and inject modified LLDP packets into an interface directly connected to an OpenFlow switch. The difference between this scenario and scenario 1 is that two compromised hosts are involved in the attack. For instance, as shown in Figure 4 (b), a compromised H3 forges the LLDP packets using the Chassis ID TLV set to S1 and Port ID set to 1. H3 then injects them via port 1, which is connected to OpenFlow switch S3. Switch S3 forwards the packets to the controller as Packet-In messages. The controller receives the Packet-In messages and assumes an existing direct unidirectional link between S3 and S1. Using the two compromised hosts, an attacker can mislead the controller to build a bidirectional fake link between S1 and S3.

Each controller constructs and generates a different variant of an LLDP packet. Some controllers add security features (such as cryptic hash values) to LLDP fields. We divided controllers into three categories based on the security features added to the LLDP packet TLVs. The first category comprises controllers that generate LLDP messages without hash values. These controllers are vulnerable to fake LLDP injection attacks, including RYU, POX, and Beacon controllers. The second category includes controllers that add a unique hash value to any of the TLVs of the LLDP packet. A static hash value is added once and repeated for all the topology discovery rounds. As authors [7], [8] have mentioned, by performing reverse engineering, it is possible to obtain the necessary details to reconstruct an LLDP packet with a valid hash value to create a poisoned packet. Floodlight, Opendaylight, and HPEVAN controllers are examples of such controllers. The third category comprises controllers that generate a dynamic hash value for each active port in each topology discovery cycle. ONOS controllers use SHA256 for the hash function over the port number and the timestamp as a single cryptographic key in each discovery cycle. As a result, ONOS is resistant to fake LLDP injection attacks.

B. LLDP RELAY ATTACK

An LLDP relay attack is a type of LFAs. Instead of injecting fabricated LLDP packets, an attacker just relays LLDP packets from one port to another, taking advantage of the controller LLDP broadcast. Two possible network entities are used to successfully create fake links by relaying LLDP packets: host-based and switch-based.

1) HOST-BASED LLDP RELAY ATTACKS

We assume that there are at least two compromised hosts in the network that are under adversary control. The adversary must create a communication channel between the two compromised hosts to relay the LLDP packets. There are two methods used to build a communication channel: an out-band channel (physical links) and an in-band channel (logical tunnel).

Scenario 1 (Relaying LLDP Packets Using an Out-Band Channel): An attacker is required to create an out-band communication channel to bridge the two network interfaces. Two network mediums can be used to create out-band

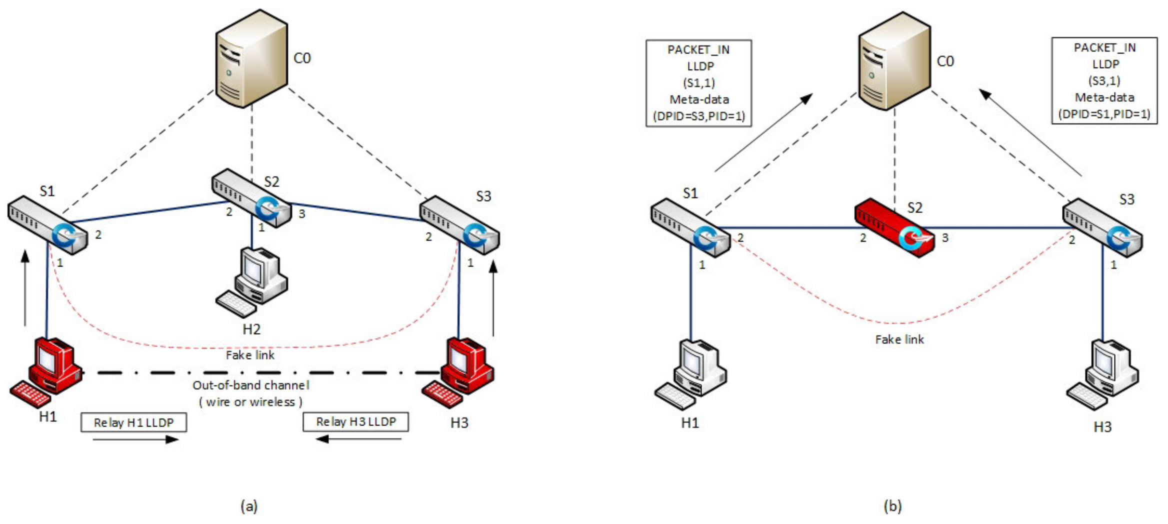


FIGURE 5. **LLDP relay attacks. (a) host-based LLDP relay attacks. (b) switch-based LLDP relay attacks.**

channels: wired and wireless connections. Creating a direct wire connection (coaxial or fibre cable) between two hosts has some physical restrictions, such as distance, which limits the capability to build the channel in all network environments [24]. An out-band wireless channel is a better choice for avoiding these physical restrictions. A wireless channel can be built in either infrastructure or ad hoc mode. Infrastructure mode requires the use of an access point to connect two hosts. In the ad hoc mode, each host communicates directly with each other within the communication range [25].

Figure 5 (a) illustrates the relaying of LLDP packets using two compromised hosts via an out-band channel. Upon receiving the LLDP packet from switch S3 via port 1, compromised host H3 instantly forwards the packet over an out-band channel to an associate, host H1. Thereafter, H1 sends the packet to the directly connected OpenFlow switch S1 via port 1, which adds its ingress information and forwards it to the controller. The controller assumes a unidirectional link between (S3,1) and (S1,1). Thus, the topology information is updated, resulting in a unidirectional fake (non-existing) link in its network topology view. A bidirectional link can be easily built by making host H1 send the received LLDP packets via port 1 to host H3. Host H3 forwards the packet to switch S3 via port 1.

Scenario 2 (Relaying LLDP Packets Using an In-Band Channel): In this scenario, the attacker uses an in-band channel to relay LLDP packets to create a fake link between the two switches. The attacker must have a network connection between the two compromised hosts to relay LLDP packets. The network connection can be verified using different tools such as the ping tool. Two major protocols are used to communicate between hosts: the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Unlike UDP, TCP requires a virtual connection before exchanging data between the hosts. The controller is less likely to identify the ports of the two connected hosts as host ports when using the UDP client-server mode. This is because the client does not need to form a connection with the server.

Using a UDP client/server connection, an attacker can send an LLDP packet from H1 to H3, as shown in Figure 5 (a). Both the compromised hosts have an executable program with three threads. The first thread is used to sniff the LLDP packet from the port directly connected to the switch (e.g., port 1 of host H1). Whenever the first thread receives an LLDP packet, it triggers a second thread. The second thread is used to create a UDP socket and send the received LLDP packet to another compromised host (e.g., host H3). The LLDP packet is encapsulated as a datagram in the IP packet. The third thread extracts an LLDP packet from the IP packet received from the UDP socket. It then relays the LLDP packet to the port directly connected to the switch (e.g., port 1 of host H3). Both compromised hosts performed these three threads to create a bidirectional fake link between S1 and S3.

2) SWITCH-BASED LLDP RELAY ATTACKS

In a switch-based LLDP relay attack, we assume at least one compromised switch in the topology. Attacks via compromised switches not only have the same capabilities as host-based LLDP relay attacks but also have significant consequences and severity in the network [26].

The idea behind a switch-based LLDP relying attack is to use compromised switches rather than compromised hosts. The lack of security in the SDN design makes it possible for attackers to compromise switches, either by exploiting some vulnerability on the switch or by modifying the switch configuration by physically accessing it. According to the author, [27], an attacker can modify the switch configuration and flow tables if they compromise the OpenFlow switch. For flow table modification, the attacker can add, modify, or remove flow entries from the flow tables of the compromised switch. However, the attacker cannot perform other actions on the compromised switch, such as changing the way the switch processes the control messages received from the controller. We assume that the attacker only has remote read and write privileges on the flow tables in the compromised switches.

To illustrate the attack, we assume that S2 is a compromised switch, as shown in Figure 5 (b). In this scenario, the attacker creates a non-existent fake link between S1 and S3. The attacker simply implements three actions on the flow table of compromised switch S2. The first is to remove the flow rule that instructs the received LLDP packets from any port to be sent to the controller. The second is to add a flow rule instructing the switch S2 to forward the received LLDP packet from switch S1 via port P2 to port P3. The third action is to add a flow table rule to instruct the received LLDP packet from S3 via port P3 to be forwarded to port P2. As a result of the flow table modification by these three actions, the attacker can mislead the controller by wrongly discovering a direct link connecting switch S1 via P2 to switch S3 via P2.

To prove the applicability of the attack implementation, we considered the attack impact on the routing function on the application layer of the SDN design. We installed a multipath routing application on top of the RYU controller. The application is responsible for finding available routes to the destination in the network topology and calculating the cost of each discovered route [28]. As shown in Figure 6, without an attack, the application finds one path to connect S1 to S3 via S2. After the attack implementation, the attacker successfully adds a fake link to the topology database. The routing application selects the fabricated link because it has fewer hops to the destination. Thus, traffic is directed through the fake link.

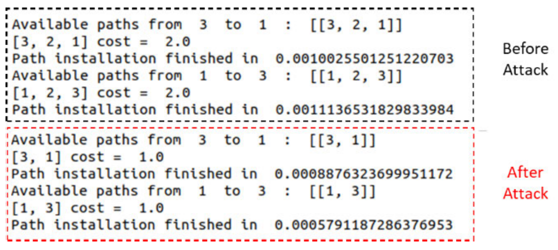

```
Available paths from  3  to  1  :  [[3, 2, 1]]
[3, 2, 1] cost =  2.0
Path installation finished in  0.0010025501251220703
Available paths from  1  to  3  :  [[1, 2, 3]]
[1, 2, 3] cost =  2.0
Path installation finished in  0.0011136531829833984
Available paths from  3  to  1  :  [[3, 1]]
[3, 1] cost =  1.0
Path installation finished in  0.0008876323699951172
Available paths from  1  to  3  :  [[1, 3]]
[1, 3] cost =  1.0
Path installation finished in  0.0005791187286376953
```

FIGURE 6. **Routing application before and after the attack.**

Suppose that multiple switches are connected to a compromised switch. In this case, removing the flow rule that instructs the switch to send the received LLDP packet to the controller can cause the removal of all the switch links connected to the compromised switch. An attacker can solve this problem by modifying the priority of the flow rule to be less than the flow rules that are used to forward the received LLDP packet to the egress ports. As a result, the compromised switch can create a fabricated link between two normal switches without affecting other links with other switches.

C. LLDP FLOODING ATTACK

An LLDP flooding attack is used to flood the controller with a massive amount of fake crafted LLDP packets generated by compromised hosts. The attacker aims to exhaust the controller resources and consume the bandwidth of the channel that connects the switches to the controller.

To illustrate the attack, we assume that there is at least one compromised host in the topology used to flood the controller with fake LLDP packets. The attacker uses packet crafting tools such as Scapy [29] to create fabricated LLDP packets. The attacker sends a massive number of LLDP packets (e.g., 50,000 packets per second). The controller is required to parse every received LLDP packet to extract vital information used for building a topology view. As a result, the controller's resources are rapidly consumed, affecting the genuine packet service rate. Furthermore, the bandwidth of the control channel is consumed because of malicious traffic.

D. IMPACT ANALYSIS OF LFAs ON SDN LAYERS

Each layer of the SDN architecture is negatively affected by the LFAs. The impact of the attack on the application layer appears through a variety of topology-dependent services, for instance, routing, load balancing and topology-based slicing applications. For example, the routing application requires information regarding the network topology to calculate the shortest path to route the network traffic to its destination. Therefore, the poisoning of topology information by creating fake links can lead to route network traffic to a malicious route. In addition, this affects the legitimate shortest path towards the destination.

The impact of the attack on the control layer is reflected by targeting controller functionality to compute and create an abstract of the network topology. The control layer is the core part of the SDN technology. Misleading the control layer to obtain and maintain the network status and topology information affects the supplementing application layer due to the false information. As a result, false topology information deceives the decisions of the application layer services, as discussed previously.

The infrastructure layer, which consists of switches that support the OpenFlow protocol, is responsible for processing and forwarding the traffic from the source to the destination. In addition, OpenFlow switches rely on SDN controllers for traffic forwarding decisions. While the attacker successfully maintains a fake link, switches direct traffic to fake links based on the flow rules installed by the controller. This leads to communication failures and poor switch performance.

IV. REQUIREMENT SPECIFICATIONS

Based on the threat analysis above, the following gives the requirements for the design of LINK-GUARD.

A. FUNCTIONAL REQUIREMENTS

As mentioned earlier, the novel method should be able to detect fake links created in the network. There are three different ways through which fake links may be created or the attacks may be mounted. To detect or thwart these attacks, the novel method should satisfy the following functional requirements.

(FR1) It should be able to detect the fake links created by LLDP injection attacks.

(FR2) It should be able to distinguish between the links created by relaying normal LLDP packets between a pair of SDN switches and the links created by relaying normal LLDP packets by a pair of compromised hosts.

(FR3) It should be able to prevent compromised switches from creating a fake link by forwarding normal LLDP packets between connected SDN switches.

B. SECURITY REQUIREMENTS

(SR1) It should be more resilient to LLDP flooding attacks.

C. PERFORMANCE REQUIREMENTS

(PR1) The bandwidth consumed should be as low as possible.

(PR2) The computational cost incurred should be as low as possible.

(PR3) The time required to verify the legitimacy of a new link should be as short as possible.

V. EXISTING SOLUTIONS AND ANALYSIS

There have been numerous studies that address various security aspects of SDN. However, only a few studies have addressed the security of SDN topology discovery mechanisms. A recent study [30] provided a systematic security analysis of state-of-the-art countermeasures against topology attacks and their vulnerabilities. The following paragraphs comprehensively review the existing studies that have proposed security solutions to the topology discovery process in SDN. In addition, we have exploited the weaknesses of each proposed approach.

The concept of an LFAs was introduced by Hong et al. [7]. The authors designed a defence mechanism called TopoGuard, which assumes that adversaries can control one or more hosts, implying that the controller and switches are trustworthy. TopoGuard is an OpenFlow-based SDN controller extension which uses port classification and LLDP authentication to prevent LFAs. The controller categorised switch ports as a HOST, a SWITCH, or ANY, based on the first packet received from the port. Therefore, the controller stops sending LLDP packets to any port classified as a HOST. In addition, TopoGuard adds an optional TLV Hash-based Message Authentication Code (HMAC) to authenticate LLDP packets, thus ensuring packet integrity and origin. However, their proposed method computes HMAC using a static secret key, which is vulnerable to relay-type LFAs. Furthermore, the LLDP flooding attack was ignored in this solution.

Alharbi et al. [11], [31] proposed using an HMAC with a dynamic key attached to each LLDP packet to provide integrity and authentication. However, this approach requires the controller to track the keys used in each discovery round. Furthermore, it adds 8% to the CPU overhead. Nevertheless, this study does not address relay-type LFAs. On the other hand, [24] proposed a statistical analysis of link latencies to detect LLDP relay attacks by measuring the link latency of receiving LLDP packets. However, an attack cannot be detected if an attacker relays the LLDP packet at a high speed as low as 100 milliseconds [32].

Alimohammadifar et al. [33] proposed a Stealthy Probing-based Verification (SPV) defence to detect link fabrication attacks. The SPV sends probing packets that are indistinguishable from standard packets toward the switches to find potential fake links. While most of the studies assumed that the switch could be trusted, this study claimed to work even if a few switches were compromised. However, this approach may result in bandwidth consumption and scalability issues in large-scale networks.

Skowyra et al. [34] proposed an improved version of TopoGuard, called TopoGuard+. In addition, two new attacks were introduced: port probing and port amnesia. Port Amnesia Attack (PAA) refers to a technique that enables an attacker to reset the port's behavioural profiling. Consequently, defences that use port profiling (such as host, switch, or any) based on the first seen packet are vulnerable to PAA. Thus, it opens up the system to LFAs. Therefore, TopoGuard+ includes a Link Latency Inspector (LLI) module, which is used to differentiate between genuine and fake switch links. In addition, it resists the PAA. LLI measures the switch-internal link latency during all LLDP propagations and flags anomalies that could indicate a forged link. A similar approach was proposed by Wang et al. [35]. However, in the study by [30], the author discovered an attack that allowed for the removal of genuine links between switches, taking advantage of how LLI works. In addition, the method used to measure link latency is not scalable for large-scale networks and unsuitable for low-rate networks.

Furthermore, the authors of [12] proposed an Efficient and Secure Link Discovery Scheme (ESLD). The ESLD only generates and sends LLDP packets to the non-host ports. Thus, it reduces the number of LLDP Packet-Out messages that are unnecessary for topology discovery. In addition, ESLD uses a time-marked HMAC (tHMAC) verification technique to prevent LFAs. However, the port classification approach is based on profiling the behaviour of a given port, that is vulnerable to PAA. As a result, the system is open to LFAs.

Researchers [13] have presented a Secure and Efficient Topology Discovery Protocol (sOFTDP) that shifts a part of the link discovery to the SDN switch. The topology discovery mechanism is performed only when there is a link-state change in the network. Thus, the sOFTDP eliminates the possibility of LLDP flooding attack packets by removing periodic LLDP packet broadcasts. In addition, the content of the LLDP packets is encrypted with hash values to prevent further spoofing attacks and controller fingerprinting. Despite these advantages, the study has not been assessed in a large network environment. In addition, it does not resist LLDP relay-type attacks when the attacker changes the state of the port before launching the attack.

Nehra et al. [8], [10] showed that most SDN controllers lack security mechanisms to protect against LFAs (e.g., poison, replay, and flooding attacks). In addition, the authors proposed a lightweight, efficient, and secure approach called

TABLE 1. Comparative table of the existing security solutions to link discovery in SDN.(✓: satisfy the requirement, ✗: not satisfy the requirement).

References	FR1	FR2	FR3	SR1	PR1	PR2	PR3
[7]	✓	✗	✗	✗	✓	✗	✗
[11],[30]	✓	✗	✗	✗	✓	✗	✗
[23]	✗	✓	✗	✗	✓	✗	✗
[32]	✓	✗	✓	✗	✗	✗	✗
[33]	✓	✓	✗	✗	✓	✗	✗
[34]	✗	✓	✗	✗	✓	✗	✗
[12]	✓	✗	✗	✗	✓	✗	✗
[13]	✓	✗	✗	✓	✓	✓	✓
[8],[10]	✓	✗	✗	✓	✓	✓	✓
[31]	✓	✓	✗	✗	✗	✗	✗
[35]	✓	✓	✗	✗	✓	✗	✓
[36]	✓	✗	✗	✓	✓	✓	✓
[25]	✗	✓	✗	✗	✓	✓	✓
[37]	✓	✗	✗	✗	✓	✓	✓
[38]	✓	✓	✗	✗	✓	✓	✓
LINK-GUARD	✓	✓	✓	✓	✓	✓	✓

Secure and Lightweight Link Discovery Protocol (SLDP). For link discovery, the SLDP employs a new packet structure that uses minimal frame features by removing unnecessary fields from the standard LLDP frame. In addition, the SLDP creates an eligible port list and only sends SLDP packets to eligible ports to avoid attacks. However, the SLDP is vulnerable to PAA, which leads to an LLDP relay-type LFA.

Chou et al. [32] proposed a Correlation-based Topology Anomaly Detection mechanism (CTAD). CTAD uses Spearman's rank correlation to analyse the time difference between each LLDP packet round-trip delay in order to determine the existence of relay-type link fabrication attacks. Also, a dynamic authentication key in the LLDP frame is used to prevent LLDP injection attacks. Moreover, a counting mechanism is used to calculate the number of LLDP packets received at each port in each topology discovery cycle to detect LLDP flooding attacks. However, CTAD requires sending many LLDP packets over the network to calculate the correlation coefficient. Thus, it causes bandwidth consumption and scalability issues in large enterprise networks.

Huang et al. [36] developed a lightweight and efficient SDN topology verification scheme called TrustTopo. The scheme uses a chaotic model and dynamic password generation to ensure the unforgeability and integrity of links. However, this scheme is not suitable for low latency networks. In addition, the password generation method causes scalability issues.

Jia et al. [37] introduced the Lightweight Automatic Discovery Protocol (LADP) to discover the network topology. The controller generates a random number added to the AUTH TLV of the LDAP frame, which is used for authorisation to pass the controller. Also, the controller creates a blocked ports list to avoid forwarding the LADP frames to non-switch ports. On the other hand, a meter table is used to prevent flooding attacks. However, suppose that an attacker launches the attack directly after resting port states (e.g., port amnesia attack). In that case, their approach does not counter LLDP relay-type LFA.

Sonali et al. [26] proposed a simple defence mechanism that detects host and switch-based LFAs using active ports. This mechanism is based on monitoring the active ports on every switch. An active port connected to multiple links simultaneously is considered to be a fake link. However, an attacker can forge an LLDP packet.

Hauser et al. [38] proposed a novel secure link discovery mechanism called P4-MACsec. This automated deployment mechanism provisions IEEE 802.1AE (MACsec) on the detected links between P4 switches. By encrypting payloads and sequence numbers, P4-MACsec enhances LLDP. However, the authentication-based mechanisms could not defend against the LLDP relay-type LFAs.

Kumar et al. [39] proposed a novel defence method against relay-based poisoning attacks called the Topology Validator. The defence solution is based on monitoring the number of port status messages received by the controller after administratively turning down the link port. For example, suppose the controller does not receive two status messages from either switch of a discovered link after shutting down one of the link ports. In this case, it would consider the newly discovered link to be a fake link because receiving one port status means the port is a host port. However, this solution is designed under the assumption that the switches and controller are trustworthy and only hosts may be compromised. Also, this solution will cause the removal of normal links in the hybrid SDN.

Overall, the existing security solutions for SDN topology discovery have limitations in coping with host-based and switch-based LFAs. Based on the above-mentioned related research, most studies assume that only hosts may be compromised. However, this assumption is untrue; a switch may be compromised. Moreover, a compromised switch can poison the network topology view.

Approaches that use the HMAC with a dynamic key for LLDP packet authentication have non-negligible processing overhead and time consumption for the controller. In large-scale networks with tens of thousands of active ports, the SDN controller is required to generate an LLDP packet for each active port with a unique hash value in every discovery round.

On the other hand, approaches that use the LLDP packet propagation delay to measure link latency are unsuitable for

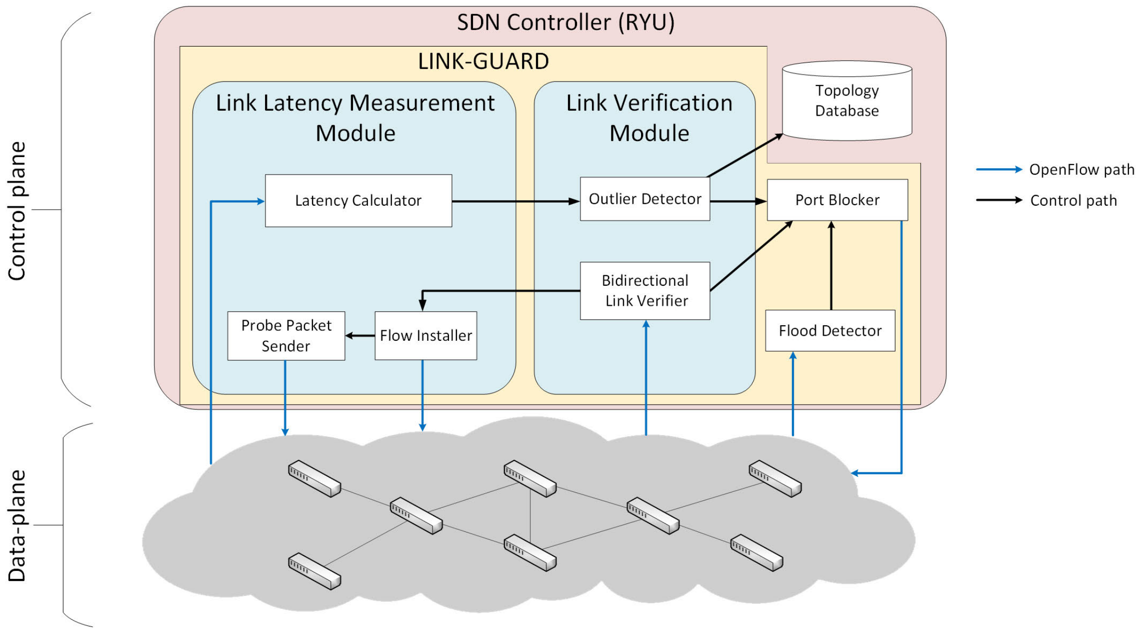


FIGURE 7. LINK-GUARD architecture.

large-scale and low-rate networks. Increasing the scale of the SDN networks increases the link latency measurement errors. Because adding more switches will increase the controllers' overhead, leading to a longer delay in processing LLDP packets.

Table 1 compares existing security solutions based on the requirements specified in section IV. As seen from the table, none of the existing solutions satisfies all requirements. Overall, none of these studies shows an effective and scalable security solution for both host-based and switch-based LFAs. This study proposes an effective and scalable security solution called LINK-GUARD. The proposed solution attempts to cover the limitations mentioned above. The authors assume that both hosts and switches may be compromised.

VI. LINK-GUARD ARCHITECTURE

A. OVERVIEW

LINK-GUARD is a security solution framework for the three types of LFAs. The attacks are (Attack 1) LLDP injection attacks using compromised hosts, (Attack 2) host-based and switch-based LLDP relay attacks, and (Attack 3) LLDP flooding attacks using compromised hosts. This framework provides a detection method for each type of attack. For attack 1 detection, we used the Bidirectional Link Verification (BLV) method based on two-way link direction verification. For attack 2 detection, we used the Link Latency Measurement (LLM) method with a statistical analysis technique to detect outliers in the data distribution of links latency. For attack 3 detection, we used the Per-port LLDP Packet Counter (PLPC) method, which is based on counting the number of LLDP packets received from each port in each discovery round. In addition, LINK-GUARD provides a mitigation method in the case of a detected fake link, which is based on port blocking.

The LINK-GUARD framework was implemented as an extension of the topology discovery service in the RYU controller. The LINK-GUARD architecture is comprised of two major modules and seven submodules, as shown in Figure 7. The modules are the link verification module and link latency measurement module. In the following section, we explain each module and submodules.

B. LINK VERIFICATION MODULE

The Link Verification (LV) module classifies newly discovered links as either normal or fake. The LV has two submodules, a bidirectional link verifier and an outlier detector. The details of each submodule are discussed below.

The Bidirectional Link Verifier (BLV) module examines newly detected links to confirm the existence of a bidirectional link connection. The BLV module performs two sequential tasks: new link detection and bidirectional link verification.

The first task is to detect new links added to the network. The BLV follows three steps to discover new links in a network. The first step is intercepting a Packet-In message sent from OpenFlow switches that contain an LLDP packet. The second step disassembles the Packet-In message and extracts the link information. The link information is formed by combining the chassis ID and the ingress port of the Packet-In message with the chassis ID and port ID TLVs of the LLDP packet payload. The final step is to match the link information with the existing links information stored in the database. If that link information does not match any existing links, is considered a new unidirectional link. Upon detecting a unidirectional link, BLV moves to the second task.

The second task is bidirectional link verification. A bidirectional link is constructed by receiving a two-way LLDP

Algorithm 1 BLV Algorithm

```
1:  Input: M (Incoming LLDP packet)
2:  Procedure Link Validation (new links)
3:      Src_dp, Src_port, Dst_dp, Dst_port ⟵ EXTRACT(M)
4:      Link_ID = (Src_dp, Src_port, Dst_dp, Dst_port)
5:      ReverseLink = (Dst_dp, Dst_port, Src_dp, Src_port)
6:      If Link_ID or Reverse_Link in Verified_Links:
7:        Return
8:      Else if Link ID or Reverse Link in Blocked Links then
9:        Return
10:     Else if Link_ID in Unidirectional_links then
11:       If Reverse Link in Unidirectional links then
12:           Rasie "Bidirectional link detected"
13:           FlowInstaller(Link_ID)
14:       Else
15:           Rasie "Unidirectional link detected"
16:           Rasie "Checking for reverse link"
17:           Wait(timeout)
18:           If Link_ID in Unidirectional_links and
                    Reverse_Link not in
                    Unidirectional_links then
19:              Rasie "Timeout exceeded for checking
                  Bidirectional link"
20:              Rasie " ATTENTION! Abnormal link"
21:             Port_Blocker(Dst_dp, Dst_port)
22:       Else
23:             Unidirectional_links ⟵ ADD(Link_ID)
```

packet. Based on the unidirectional link information of the new link, the BLV waits for the LLDP packet, which represents reverse link information. If the model discovers a bidirectional link, then it communicates with the link latency measurement module to validate this link. Otherwise, after a time-out, this unidirectional link is considered an attempt at an LLDP injection attack using a single compromised host. As a result, BLV interacts with the port blocker module to block the port and send a notification for further investigation. Algorithm 1 formally illustrates the BLV module.

The outlier detector module classifies a given link latency value for a bidirectional link as an outlier or as normal. This module receives the link latency value from the latency calculator module. The link latency value is considered to be an outlier if it deviates from other latency values. This module uses a univariate outlier detection method called boxplot [40]. The boxplot computes the first quartile (Q1), median, third quartile (Q3), maximum, and interquartile range (IQR) of the link latency data distribution. If the given link latency is greater than Q3+3*IQR, it is considered an outlier. In addition, this module report links information to the port blocker module if the link latency is classified as an outlier. If it is not classified as an outlier, a link is added to the topology database.

C. LINK LATENCY MEASUREMENT MODULE

The link latency measurement (LLM) module is responsible for generating, sending and collecting probe packets to/from OpenFlow switches. This module has three submodules: a flow installer, a probe packet sender, and a latency calculator. The details of each submodule are discussed below.

The flow installer module is used to construct and send the flow rules to the targeted switches. These rules are used to forward the probe packets between the two switches ports of the detected bidirectional link. It also sends probe packets back to the controller. The flow rules are injected into the source and destination switches. The source switch is a switch that receives a probe packet from the controller. The destination switch is the switch that sends the probe packets back to the controller. The controller randomly selects the source and destination switches.

There are two types of generated flow rules: group table and match-action rules. The group table rules are designed to execute two separate lists of actions. Each action list is called a bucket. The first bucket contains one action: forward the received probe packet to the controller. The second bucket contains two actions: decreasing the Time To Live (TTL) value of the probe packet by one and forwarding the packet back to the incoming port. The group table rule is installed in every OpenFlow switch in the network.

On the other hand, the match-action rules are used to forward probe packets between a source and destination switch. Two matching-action rules exist. The first rule is installed in the source switch to forward the probe packet to the destination switch. The second rule is installed in the destination switch to forward probe packets to the group table. Figure 8 summarises the rules installed in the source and destination switches.

The probe packet sender module is responsible for generating and sending probe packets. More specifically, this module generates an Internet Control Message Protocol (ICMP) packet and sends it to the source switch of the new bidirectional link. The controller predefines the TTL field of the ICMP packet header to limit the number of probe packets used to measure link latency. The probe packet Maximum Transmission Unit (MTU) equals 1,500 bytes. After installing the flow rules, this module is triggered to generate and sends a single ICMP probe packet to the source switch.

The latency calculator module is responsible for collecting the probe packets and calculating the link latency of the bidirectional link. This module employs the following steps to calculate the link latency. Initially, it records a timestamp for every probe packet received from the destination switch. Then, it subtracts the current link delay between the controller and destination switch from the recorded timestamp. The results are stored in a list. These two steps are repeated for all of the received probe packets. After receiving all the probe packets, the time difference between the recorded timestamps is calculated. Finally, the median of the stored time difference of the timestamps is calculated to produce the link latency of the bidirectional link. The link latency is reported to the

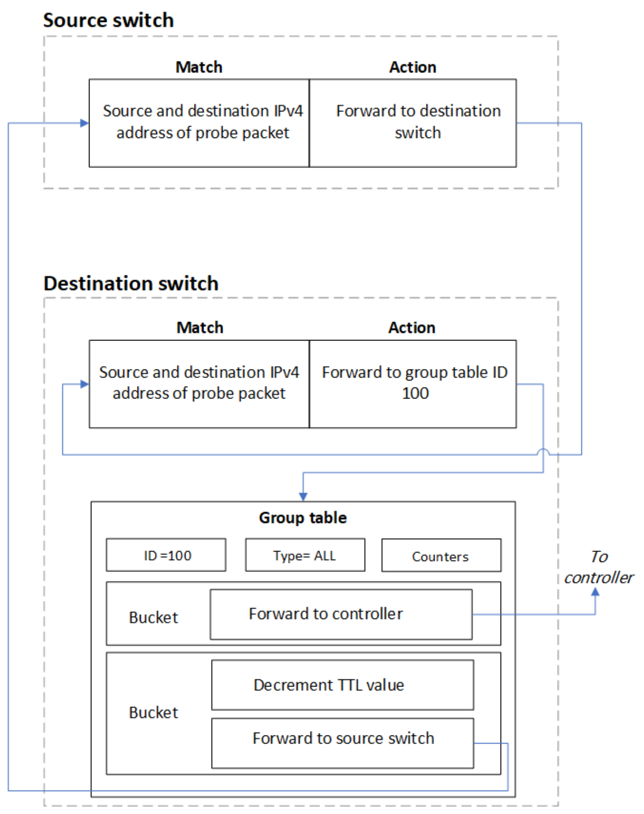


FIGURE 8. **Group table and match-action rules installation.**

Algorithm 2 Link Latency Calculation Algorithm

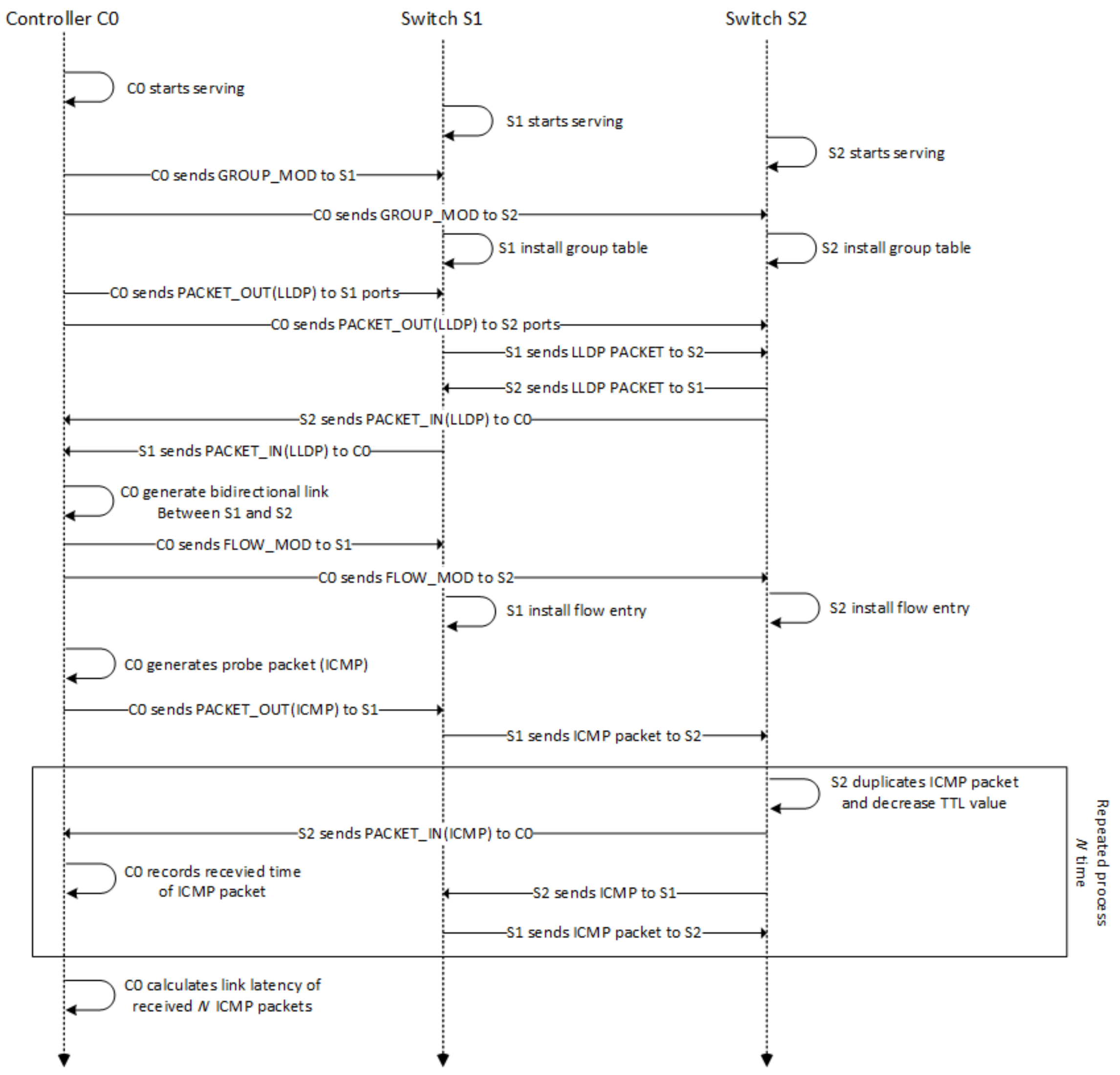
```
1:  Input: P (ProbePacket), TTL (Time-To-Live value),
     Ed (Egress switch link delay)
2:  Procedure  LinkLatencyMeasurment(Bidirectional
     new   link)
3:     Initial values: x =  1,y = 0
4:     While true do
5:        $e_{delay} \longleftarrow$ current link latency between controller
               and destination switch Ed
6:        $t_{probe} \longleftarrow$ record probe packet receiving time
7:        $l_{delay} \longleftarrow t_{probe}$ - $e_{delay}$
8:        ProbeDelayList $\longleftarrow$ ADD($l_{delay}$)
9:        If length(ProbeDelayList) == TTL then
10:          For i in (TTL-1) do
11:             di = ProbeDelayList [value index (x)
                 - value index (y)]
12:             TotalDelaysList   $\longleftarrow$ ADD(Jz)
13:             If length(TotalDelaysList == (TTL -1)
                   then
14:               LinkLatency = median (TotalDelaysList)
15:               Return LinkLatency
16:             x+ =1
17:             y+ =1
```

outlier detector module for further analysis. Algorithm 2 shows the process of the link latency calculation.

The event sequence of the LLM module is illustrated in Figure 9. To understand this in more detail, note that we use three network entities, a controller C0 and two switches, S1 and S2. There is a vertical timeline for each network entity.

Initially, controller C0 starts serving before switches, S1 and S2. After controller C0 establishes a successful connection with switches S1 and S2, it sends the group table rule to both switches. Then, each of the switches installs the group table rule. Next, C0 generates and sends LLDP packets in PACKET_OUT to all the active ports of S1 and S2. Upon receiving the PACKET_OUT messages, S1 and S2 resolve the LLDP packets and send them to the designated ports. After S2 receives the LLDP packet from S1, it sends it to C0 as a PACKET_IN message. Similarly, S1 sends an LLDP packet received from S2 to C0 as a PACKET_IN message. After C0 receives PACKET_IN from S1 and S2, it creates a bidirectional link between S1 and S2 without adding it to the topology database. C0 randomly selects the source and destination switch of the bidirectional link. Therefore, we denote S1 as the source switch and S2 as the destination switch. Next, C0 generates and sends match-action flow entries to S1 and S2 as FLOW_MOD PACKET_OUT messages. Then, each of the switches installs a match-action flow entry. After installation, C0 generates a single ICMP packet with a custom TTL count (e.g., 10). Then, C0 sends the ICMP packet as PACKET_OUT to the source switch S1 port. S1 unwraps the ICMP packet from the PACKET_OUT message. Then, S1 sends the ICMP packet to destination switch S2. S2 mirrors the ICMP packet and sends one copy to C0 as a PACKET_IN. The TTL of the second copy of the ICMP packet is decreased and forwarded to S1. The probe packet bounces between S1 and S2 until the TTL count equals zero. Then the packet is discarded. Next, C0 records the ICMP packet's received time. After receiving all the probe packets (ICMP), C0 calculates the time difference between the received time for all ICMP packets. C0 then calculates the median of the recorded time differences to obtain the link latency. The event sequence diagram surrounding area, denoted by a rectangle, depicts the repeatedly executed instructions. This process is repeated N times, where N is equal to the TTL count.

Other essential modules in the LINK-GUARD framework include flood detectors and a port blocker. A flood detector module is used to detect LLDP flooding attacks. This module monitors and records the number of LLDP packets received from each port during each discovery cycle. In this matter, we can determine whether any port suffers from an LLDP flooding attack in real-time. By default, the controller sends a single LLDP packet to each active port in every discovery cycle. Thus, relaying more than one packet in each discovery cycle from a specific port to the controller is considered abnormal behaviour. It is detected as a flooding attack from the second packet received by the controller in the same discovery cycle from the same port. Furthermore, this module communicates with the port blocker module to block the attack sources and notify the controller for



Controller C0
Switch S1
Switch S2
C0 starts serving
S1 starts serving
S2 starts serving
C0 sends GROUP_MOD to S1
C0 sends GROUP_MOD to S2
S1 install group table
S2 install group table
C0 sends PACKET_OUT(LLDP) to S1 ports
C0 sends PACKET_OUT(LLDP) to S2 ports
S1 sends LLDP PACKET to S2
S2 sends LLDP PACKET to S1
S2 sends PACKET_IN(LLDP) to C0
S1 sends PACKET_IN(LLDP) to C0
C0 generate bidirectional link Between S1 and S2
C0 sends FLOW_MOD to S1
C0 sends FLOW_MOD to S2
S1 install flow entry
S2 install flow entry
C0 generates probe packet (ICMP)
C0 sends PACKET_OUT(ICMP) to S1
S1 sends ICMP packet to S2
S2 duplicates ICMP packet and decrease TTL value
S2 sends PACKET_IN(ICMP) to C0
C0 records recevied time of ICMP packet
S2 sends ICMP to S1
S1 sends ICMP packet to S2
Repeated process N time
C0 calculates link latency of received N ICMP packets

FIGURE 9. **LLM sequence.**

Algorithm 3 Detect Flooding Attacks Algorithm

```
 1: Input: M (Incoming LLDP packet)
 2: Procedure FloodingAttackDetect
 3:   Dstdp, Dst_port ⟵ EXTRACT(M)
 4:   PortID = ( Dst dp, Dst_port)
 5:   If PortID in PortIDList then
 6:       PortBlocker(PortlD)
 7:   else
 8:       PortIDList ⟵ ADD(PortlD)
 9:   On every n second do
10:       PortIDList = empty    # reseting the list
```

more investigation. Algorithm 3 is used to detect the LLDP flooding attacks.

On the other hand, the port blocker module is used to block detected fake links from the attack source. This module receives fake link information from the bidirectional link verifier, the outlier detector and the flood detector modules. In addition, this module constructs and sends flow limitation rules to designated switches to block fake links.

VII. EXPERIMENT SETUP

To evaluate the effectiveness and performance of LINK-GUARD against the LFAs discussed in Section III, we implement the experiment in a simulated OpenFlow network environment. The emulated testbed uses Mininet [41] as a network emulator and Open Vswitches [42] to simulate realistic OpenFlow switches. We use Wireshark [43] to capture network traffic. The RYU controller [44] is used to communicate with Open Vswitches. Unlike other SDN controllers such as OpenDaylight and Floodlight, RYU has been developed as an open-source and well-documented controller. The network topologies are built via Mininet, which runs on a virtual machine with an Ubuntu 18.04 system created by the VMware Workstation. Scapy [29], a Python-based interactive packet manipulation package,

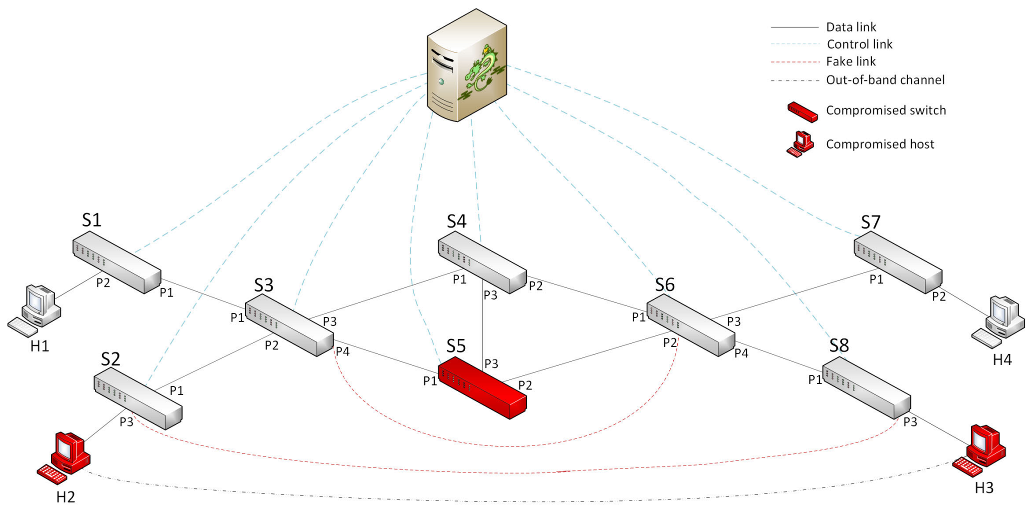


FIGURE 10. Experimental network topology.

is used to build our attack code and perform packet crafting because of its advanced and robust networking capabilities. Table 2 lists the details of the experimental environment for LINK-GUARD.

TABLE 2. Experimental environment for LINK-GUARD.

Resource	Configuration
Test-bed	Mininet (emulation) version 2.3.0
Victim/Attacker OS	Ubuntu 18.04.5 LTS
Controller	RYU version 4.34
OpenFlow switch	OpenVSwitch version 2.16.0
Network packet analyser	Wireshark version 3.6.1
Packet Manipulation	Scapy version 2.4.5

VIII. SECURITY ANALYSIS

LINK-GUARD revokes the detected fake link before adding it to the topology database of the controller, to prevent poisoning the overall view of the network topology. Furthermore, it prevents the attack source, compromised hosts, or switches from relaunching the attack. We evaluate the effectiveness of LINK-GUARD against four types of LFAs scenarios:

- LLDP injection attack scenario
- LLDP relay attack using compromised hosts scenario
- LLDP relay attack using compromised switch scenario
- LLDP flooding attack scenario

The experimental network topology used to evaluate the effectiveness of LINK-GUARD in resisting the four attack scenarios is illustrated in Figure 10. The data plane of the experimental network mainly consists of eight switches. All inter-switch links and the links that connect hosts to switches have a 1 Gbps link speed with a 1 millisecond (ms) link delay.

A. LLDP INJECTION ATTACK SCENARIO

In this scenario, an attacker host (i.e., H2) forges an LLDP packet and injects it into an attached OpenFlow switch S2. For instance, the chassis ID and Port ID of the forged LLDPDU packet are filled with switch S8 and port 3 details,

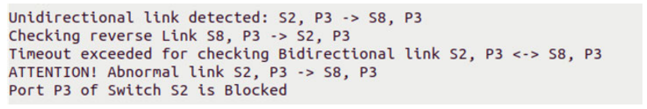

```
Unidirectional link detected: S2, P3 -> S8, P3
Checking reverse Link S8, P3 -> S2, P3
Timeout exceeded for checking Bidirectional link S2, P3 <-> S8, P3
ATTENTION! Abnormal link S2, P3 -> S8, P3
Port P3 of Switch S2 is Blocked
```

FIGURE 11. Detection of LLDP injection attacks.

respectively. Then, the fake packet is sent to port 3 of switch S2 via compromised host H2. In this case, the attacker can create a unidirectional link (DPID: S8, portID: 3 → DPID: S2, portID: 3), as shown in Figure 10. However, when the LINK-GUARD discovers a new unidirectional link by receiving a one-way LLDP packet, it will not proceed to the verification mechanism until it receives a two-way LLDP packet which represents the bidirectional link.

When using a single compromised host to inject forged LLDP packets, an attacker can only create a unidirectional link. Therefore, after a timeout (e.g., one second), the unidirectional link is considered fake. As a result, the controller blocks the port of the fake link, as shown by the RYU console output in Figure 11.

We consider the case where an attacker uses two compromised hosts (i.e., H2 and H3) to inject fake LLDP packets in order to create a bidirectional link. In this case, LINK-GUARD moves to a link verification mechanism by measuring the link latency before adding it to the topology database. This mechanism requires a real link connection between the two discovered ports. In this case, there is no real connection between H2 and H3 (i.e., an out-of-band channel). Therefore, the probe packets used to measure link latency will not be able to reach the controller. The newly discovered bidirectional link is considered fake if the controller does not receive the probe packets. The procedure after that is to block both ports of the fake link by sending a traffic limitation rule to both switches.

To reduce potential counterattacks by replicating LLMs using ICMP packets, we guide the controller to follow three steps for newly discovered bidirectional links. In the first step, the controller randomly selects the source and destination switches in order to reduce the chance of knowing which switch is responsible for sending ICMP packets to the controller. The second step is to use a random number for the total number of ICMP packets that the controller must collect to measure the link latency rather than a fixed number of packets. The final step is to check the FlowRemoved message received from the source and destination switches. The FlowRemoved message is sent to the controller via the switch when the flow entry is removed from a flow table. Both switches send a FlowRemoved message to the controller containing the packet count that utilises this rule. The controller checks the packet count and must match this with the number of packets that were selected before starting the LLM process. For instance, if the controller selects 10 probe packets that are used to measure link latency, then the FlowRemoved message sent from the destination switch to the controller must have a packet count equal to 10. The FlowRemoved message sent from the source switch must have a packet count equal to 9. If there is a mismatch in one of the packet counts in either message the controller raises an alarm for a potential counterattack.

B. LLDP RELAY ATTACK USING COMPROMISED HOSTS SCENARIO

In this scenario, an attacker aims to create a fake link by relaying the received LLDP packets between hosts H2 and H3, as shown in Figure 10. The two malicious hosts can use either an out-of-band or an in-band communication channel to forward LLDP packets.

We assume that the attacker is not only capable of relaying LLDP packets but also relaying the probe packets that are used to measure link latency for newly discovered links over an out-of-band wireless channel. As the first step to creating a bidirectional link, the attacker must relay LLDP packets in both directions. Otherwise, the link will be classified as fake. Furthermore, the controller must receive a selected number of samples to measure the link latency; this is the next step to classifying the link as benign or malicious. We assume that relaying probe packets (ICMP packets) over an out-of-band channel may create an abnormal increase in switch link latency if extra devices or channels are involved in relaying the probe packets. Therefore, after successfully calculating the link latency of a newly discovered link, we use a univariate outlier detection technique to detect outliers.

The univariate outlier detection technique is used to detect outliers from the distribution of values in a single feature space (i.e., link latency). The distribution of the one-way latency of the probe packets determines which univariate outlier detection method is most suitable for detecting outliers in the data distribution. The most common techniques are the Z-score [45], Tukey's boxplot [40], and adjusted boxplot [46].

In this experiment, we collect the link latency for four different network scales, which have 8, 20, 85, and 127 virtual switches, respectively. We use the Kernel Density Estimation (KDE) to visualise the link latency distribution, as shown in Figure 12. We observe that the distribution of the data

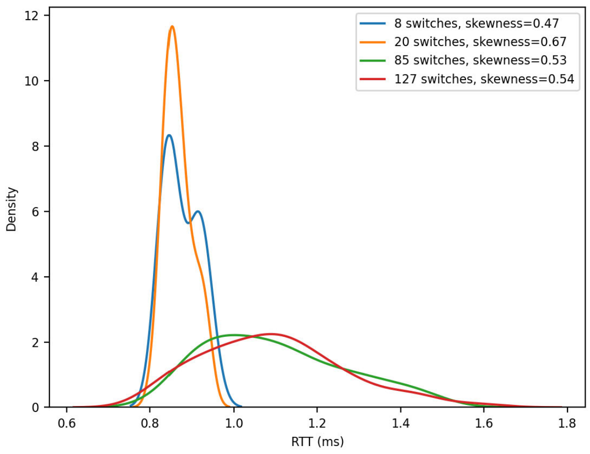


FIGURE 12. **Overall RTT distribution of 8, 20, 85, and 127 switches.**

is symmetric (normal distribution) with a light skew to the right-hand side of the distribution. Therefore, the Z-score is unsuitable for all of the network scales because it assumes that the data conform to a normal distribution [47]. The adjusted boxplot method, on the other hand, is used for moderately to extremely skewed data distribution. Thus, we use Tukey's boxplot to detect outliers because it does not assume that the data conform to a normal distribution and it is suited to moderately asymmetric distributions.

Tukey's boxplot detects outliers based on the interquartile range and it is used to obtain upper and lower bounds. If the data are outside the boundary, they are considered outliers. In other words, if the link latency lies above three times the interquartile range (IQR), the link is rejected (fake link). Otherwise, the link is normal and it is added to the topology database.

To evaluate the effectiveness of LINK-GUARD against out-of-band relay attacks, we configure an out-of-band link between two compromised hosts (H2 and H3) with five milliseconds latency. We measure the latency of all nine links with out-of-band fake links. We also measure the latency threshold for each discovered link in the topology, as shown in Figure 13. Overall, the latency of all the normal links is approximately one millisecond. By contrast, the fake link over the out-of-band channel is seven milliseconds, which is above the threshold of normal links. The result shows that LINK-GUARD successfully allocates the fake link (S8-P3<->S2-P3), as shown by the RYU console output in Figure 14.

Additionally, we measure the effectiveness of LINK-GUARD against in-band LLDP relay attacks. We conduct the attack by controlling the two compromised hosts (H2 and H3). We install a Scapy code that is executable in both compromised hosts. Each code is responsible for two main tasks. The first task involves listening to and relaying LLDP packets. For instance, the host H2 sniffs LLDP packets incoming from the connected interface to the OpenFlow switch. When receiving an LLDP packet, host H2 encapsulates the LLDP packet within the UDP packet as

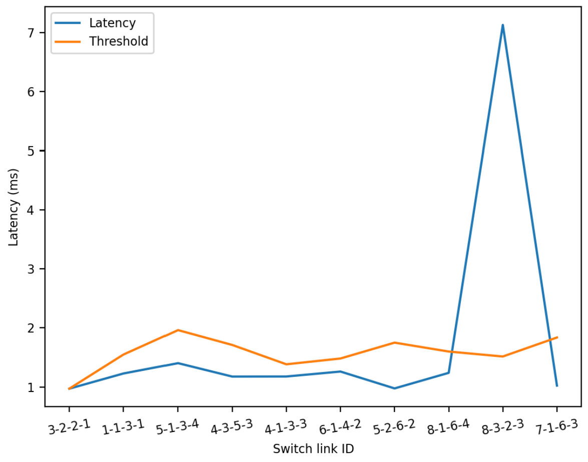
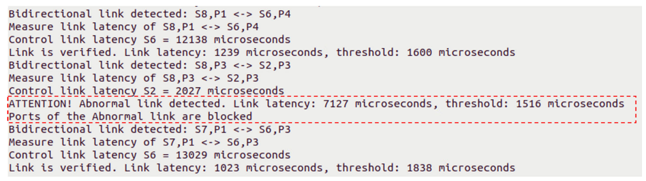


FIGURE 13. The threshold distribution with links latency.

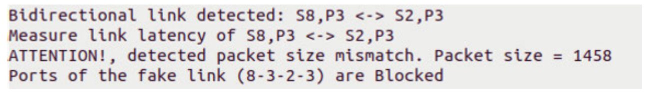


```
Bidirectional link detected: S8,P1 <-> S6,P4
Measure link latency of S8,P1 <-> S6,P4
Control link latency S6 = 12138 microseconds
Link is verified. Link latency: 1239 microseconds, threshold: 1600 microseconds
Bidirectional link detected: S8,P3 <-> S2,P3
Measure link latency of S8,P3 <-> S2,P3
Control link latency S2 = 2027 microseconds
ATTENTION! Abnormal link detected. Link latency: 7127 microseconds, threshold: 1516 microseconds
Ports of the Abnormal link are blocked
Bidirectional link detected: S7,P1 <-> S6,P3
Measure link latency of S7,P1 <-> S6,P3
Control link latency S6 = 13029 microseconds
Link is verified. Link latency: 1023 microseconds, threshold: 1838 microseconds
```

FIGURE 14. Alert from LINK-GUARD for detecting out-of-band fake links.

a payload. Then, it sends the UDP packet to host H3 over a UDP client/server connection. When host H3 receives a UDP packet, it extracts the LLDP packet and relays it to an incoming port. Similarly, host H3 sniffs and relays LLDP packets to host H2 to create a bidirectional link.

The second task involves sniffing and relaying the probe packets (ICMP packets). Both compromised hosts listen to probe packets because the source and destination switch selections are random. In our experiment, the controller selects switch S2 as the source switch and S8 as the destination switch. The controller sends an ICMP packet to port P3 of switch S2. After host H2 receives the ICMP packet, it encapsulates it in a UDP packet and sends it to host H3. Subsequently, H3 extracts the ICMP packet from the received UDP packet and sends it to port P3 of switch S8. However, an ICMP packet with 1500 bytes encapsulated in a UDP packet causes packet fragmentation. Packet fragmentation involves breaking the packet into smaller pieces when the packet is larger than the network MTU (usually approximately 1,500 bytes). Therefore, switch S8 sends an ICMP packet with a size smaller than 1500 bytes to the controller. As shown by the RYU console output in Figure 15, LINK-GUARD can successfully detect probe packets that do not match the original probe packet size. This packet size mismatch is considered to be an in-band relay attack.



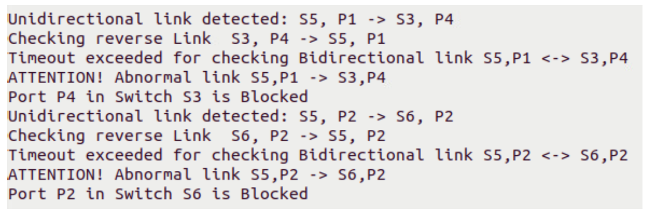
```
Bidirectional link detected: S8,P3 <-> S2,P3
Measure link latency of S8,P3 <-> S2,P3
ATTENTION!, detected packet size mismatch. Packet size = 1458
Ports of the fake link (8-3-2-3) are Blocked
```

FIGURE 15. Alert from LINK-GUARD for detecting in-band fake link.

C. LLDP RELAY ATTACK USING COMPROMISED SWITCH SCENARIO

In this scenario, the attacker aims to create a fake link by relaying LLDP packets over a compromised switch. The attacker hijacks switch S5 and installs the malicious flow table rules, as shown in Figure 10. An attacker forwards the LLDP packets from switch S3 to switch S6 without sending them to the controller, in order to create a fake link between switches S3 and S6. Likewise, the LLDP packets from switch S6 are sent directly to S3, resulting in a fake bidirectional link between switches S3 and S6.

However, in the attack scenario above, the attacker does not block the LLDP packets sent from the controller to switch S5 ports (i.e., P1 and P2). Therefore, LINK-GUARD can detect two unidirectional links: link (S5,P1-S3,P4) and (S5, P2-S6, P2). Due to the lack of reverse link information for both links, the unidirectional link ports are blocked, as shown by the RYU console output in Figure 16.



```
Unidirectional link detected: S5, P1 -> S3, P4
Checking reverse Link  S3, P4 -> S5, P1
Timeout exceeded for checking Bidirectional link S5,P1 <-> S3,P4
ATTENTION! Abnormal link S5,P1 -> S3,P4
Port P4 in Switch S3 is Blocked
Unidirectional link detected: S5, P2 -> S6, P2
Checking reverse Link  S6, P2 -> S5, P2
Timeout exceeded for checking Bidirectional link S5,P2 <-> S6,P2
ATTENTION! Abnormal link S5,P2 -> S6,P2
Port P2 in Switch S6 is Blocked
```

FIGURE 16. Alert from LINK-GUARD for detecting the unidirectional link with compromised switch.

We assume that an attacker is able to drop the controller LLDP packets sent to the compromised switch ports (i.e., P1 and P2 of switch S5). For example, an attacker can use a command-line firewall utility program to block egress traffic, such as an iptables command-line firewall [48]. Thus, the attacker hides the existing connected links from switches S5 to S3 and S6. At the same time, the controller forms a bidirectional link between switch S3 and S6, as shown by the flowmanager [48] tool in Figure 17.

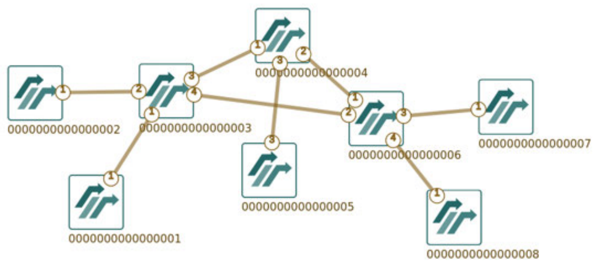


FIGURE 17. Topology view of RYU controller after the attack.

LINK-GUARD detects a bidirectional link between switches S3 and S6 via ports P4 and P2, respectively, based on the attack scenario above. Then, a probe packet (ICMP) is sent to the source switch's designated port (i.e., S3, P4). Next, the ICMP packet is forwarded through switch S5 to destination switch S6. The ICMP packet's TTL value is decreased because there are two different network subnets, which are considered one-hop counts. Consequently, the TTL value is decreased twice, once by switch S5 and once by destination switch S6. Overall, the number of ICMP packets sent from the destination switch to the controller is less than the number of packets required to measure link latency. As a result, the link is considered fake.

D. LLDP FLOODING ATTACK SCENARIO

In this attack scenario, the attacker intends to flood the controller with numerous LLDP packets to overload the controller bandwidth and consume the controller's CPU cycles. To simulate the attack, we use host H1 to flood the controller with a large number of LLDP packets (50,000 packets per second) in a 30-second attack using the TCPrelay tool [49]. If the controller receives more than one LLDP packet from a specific port in each discovery cycle, this is considered abnormal behaviour.

Compared with CTAD [32] in Figure 18, the result indicates that LINK-GUARD can detect an attack with fewer LLDP packets, which takes less time to block the port from which the attack originates. Furthermore, the controller may request the flow statistics of the blocked port to verify the packet count for further investigation.

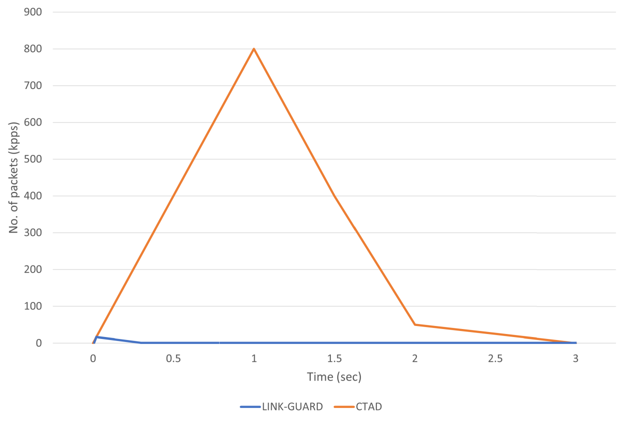


FIGURE 18. Evidence for LLDP flooding attack detection.

IX. PERFORMANCE EVALUATION

In this section, the performance of LINK-GUARD is evaluated in terms of the link validation delay, resource consumption (CPU processing and memory consumption), and detection rate. We consider four different network topologies to evaluate the performance of LINK-GUARD, which is sufficiently large to justify our validation. Table 3 lists the topologies with the number of switches, links, and ports.

TABLE 3. Topologies and key parameters.

Topology	Switches	Active Ports	Links Between Switches	Ports Between Switches
Linear	8	22	7	14
Linear	20	58	19	38
Tree,4,4	85	424	84	168
Tree,7,2	127	380	126	252

A. LINK VALIDATION DELAY

The link validation delay is the time required to verify the legitimacy of the new link. We measured the time from the unidirectional link discovery until the link classification decision. The link availability time depends mainly on the number of probe packets used to calculate link latency. We use 10 ICMP probe packets for this experiment to measure the link latency.

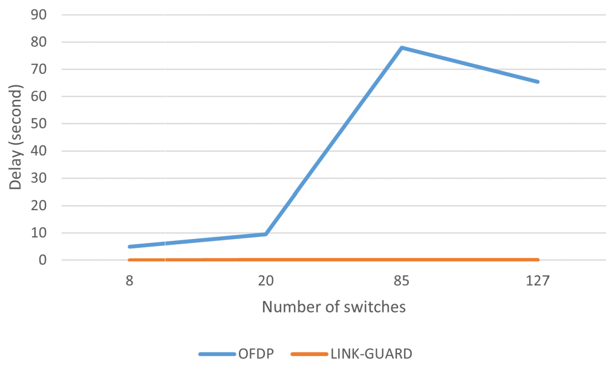


FIGURE 19. Link validation delay.

Figure 19 illustrates the average delay of link validation over four network scales: 8, 20, 85 and 127 switches. The result is compared with the same latency measurement method used in [32] and [34], denoted by the OFDP. It can be seen from the graph that the average delay of OFDP using 8 switches is around 5 seconds. The delay increases to 9.5 seconds for 20 switches, and it increases rapidly to 78 seconds for the 85 switches. However, the link delay for 127 switches decreases to 65.4 seconds because the number of active ports in this network is less than the 85 switch network, as shown in Table 2. On the other hand, the link validation delay created by LINK-GUARD is around 26 milliseconds for both the 8 and 20 switches network. The link delay increases slightly to 30 milliseconds for 85 switches and it remains constant over the 127 switch network.

Overall, LINK-GUARD creates a time overhead similar to real-time behaviour, making it a more scalable solution for dynamic and large SDN networks.

B. RESOURCE CONSUMPTION

We measure the controller resource consumption (CPU and memory usage) of LINK-GUARD, and the results are compared with those of RYU-OFDP (without LINK-GUARD). To measure the CPU and memory usage, we utilise the top command in the Linux shell. We evaluate the resource consumption over four network scales, with 8, 20, 85 and 127 switches, respectively.

Figure 20 depicts the average CPU consumption for detecting and validating all the links in the network topology. The average percentage consumption of RYU-OFDP is around 1.1%, 1.2%, 1.4%, and 1.5% for 8, 20, 85 and 127 switches, respectively. With the integration of LINK-GUARD, the average CPU usage overhead for 8 switches is 1.4%, reaching

2.1% with 127 switches, which is a 0.6% increase compared to RYU-OFDP with the same network scale. Thus, the CPU consumption of LINK-GUARD is considered reasonable.

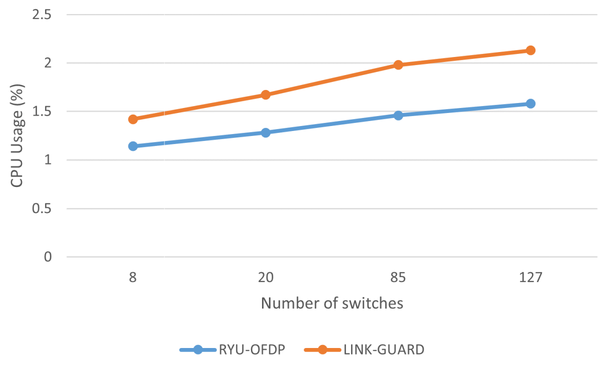


FIGURE 20. **CPU usage.**

Figure 21 is a line graph depicting the average amount of memory consumed by the controller (with and without the integration of LINK-GUARD), including link discovery and validation. LINK-GUARD's average memory consumption across all network scales is 129646.3 Kilobytes. With regard to the RYU-OFDP memory consumption, the average is 88809.4 Kilobytes. Overall, there is a nearly constant increase in LINK-GUARD's memory consumption for every network scale, compared with RYU-OFDP.

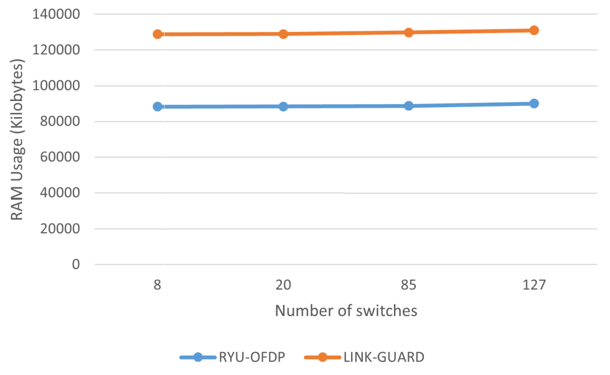


FIGURE 21. **Memory usage.**

C. DETECTION RATE

This section presents an experiment to examine the detection rate of LINK-GUARD for relay-type LFAs. We use a network topology with 127 switches, which is considered a large network size for this type of evaluation. The network topology contains 126 links, 20 of which are configured as fake links, distributed randomly over the network. We use the link Traffic Control (TC) parameter provided by Mininet to set the internal switch link latency to different delay times to emulate fake links.

To measure the switch-internal link latency using OFDP, the controller injects timestamps into every LLDP packet during the link discovery procedure. Once the controller has received the LLDP packets, it extracts the timestamps and computes the LLDP propagation delay. In addition, the controller sends echo messages to measure the control link latency (delay between an SDN controller and a switch) for both the source and destination switches. The same method was used in [32] and [34]. For LINK-GUARD and OFDP, we compute the median of ten probed latency.

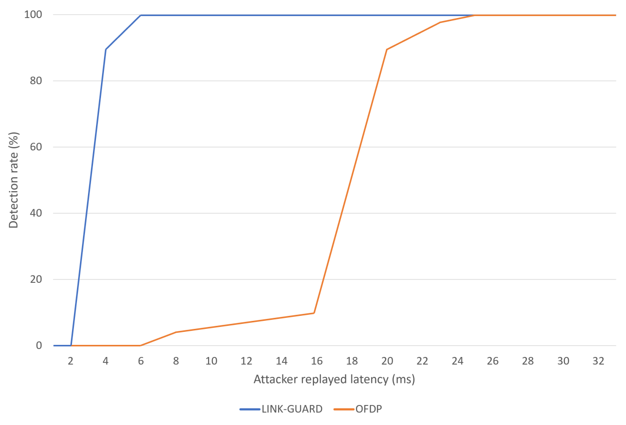


FIGURE 22. **Detection rate.**

The line chart in Figure 22 illustrates the detection rate results for LINK-GUARD and the OFDP. As we can see from the results, LINK-GUARD can detect fake links at high speeds as low as 6 ms. On the other hand, OFDP is able to detect fake links at speeds as low as 25 ms. Overall, the LLM method used by LINK-GUARD is considered a more scalable solution for large and low-rate SDN networks.

X. LIMITATIONS AND FUTURE WORK

This section discusses LINK-GUARD limitations and highlights possible future work to improve LINK-GUARD.

A number of approaches have been proposed for link discoveries in SDN networks. A recent study [50] has divided the proposed approaches into three categories: the LLDP-based Link State Discovery approach; the Tree Exploration Link State Discovery approach; and the Layer-2 based Link State Discovery approach.

The LLDP-based Link State Discovery approach is based on neighbour discoveries. Neighbouring switches discover their network links via the exchanges of point-to-point LLDP packets. With is approach, the controller sends a LLDP packet to each active port, or to each switch on the network. Each switch, upon the receipt of an LLDP packet which is typically relayed from its neighbouring switch, forwards the packet back to the SDN controller. In the Tree Exploration Link State Discovery approach, network links are discovered by using a tree exploration method. With this method, the controller only sends a single probe packet to a single switch which then floods the network, thus discovering the topology in the whole network. With the Layer-2 based Link State Discovery approach, network switches are structured into a hierarchical structure. Link discovery packets are transmitted by the controller to the switches based on this structure. The collected link related data are aggregated by designed switches at each hierarchical level before being forwarded to the controller.

LINK-GUARD requires link information on the two-way direction of the link in order to form a bidirectional link. Bidirectional link information is one of the requirements to verify the legitimacy of the new link. LINK-GUARD can be applied to the LLDP-based Link State Discovery approach because it provides bidirectional link information in every discovery cycle. However, the Tree Exploration Link State Discovery and Layer-2 based Link State Discovery approaches provide only one-way direction link information (unidirectional link) to the controller in every discovery cycle. Thus, LINK-GUARD can not be applied to these approaches, which we consider as future work.

LINK-GUARD uses the link latency of probe packets to detect LLDP relay attacks and to identify any fake link. If the latency is sufficiently low, LINK-GUARD may not be able to detect such attacks and any resulting fake link. For example, if the attacks is mounted on two compromised hosts, and if the probe packets used can be relayed at a high speed over an out-of-band channel or if the attacks are mounted on a compromised switch and if this switch is in the same broadcast domain as the target switches, then the resulting latency can be or less than 5 ms. In such cases, LINK-GUARD may not detect the fake links.

LINK-GUARD is designed under the assumption that all the network media and devices have similar transmission capabilities. In a heterogeneous SDN network, where network media and devices may have different transmission capabilities, LINK-GUARD may not be able to detect all the fake links that are created through relay-type link fabrication attacks. This is because if network media ad devices have different transmission capabilities, there will be multiple correct latency baselines. As part of our future work, we will extend the design to support the heterogeneous SDN networks.

Finally, the design of LINK-GUARD assumes that the network controller is trustworthy, and attacks are only mounted via host and switch compromises. It is obvious that the controller may also be compromised. Our future will also cover what if attacks are mounted via compromising network controllers. For example, an attacker may compromise a network controller and send false topology data to topology-dependents services run at the application layer via the controller. Detecting any trust breach between the control layer and the application layer is also an open issue.

XI. CONCLUSION

Topology discovery in SDN is an essential service for topology-aware applications. In this study, we implement and evaluate LFAs over different current mainstream controllers. As a result, we find that each controller is vulnerable to at least one type of LFA. We propose LINK-GUARD to improve the security of the topology discovery mechanism. LINK-GUARD presents a simple and effective detection method for LLDP injection attacks based on bidirectional link verification. In addition, a novel link latency measurement method with a statistical analysis technique is used to detect both host and switch-based LLDP relay attacks. LLDP flooding attack detection is based on counting the number of LLDP packets received from each port in each discovery round. We evaluate the effectiveness of LINK-GUARD under different attack scenarios. In addition, we evaluate the performance of LINK-GUARD through different network scales. The results show that LINK-GUARD effectively secures topology discovery against host-based and switch-based LFAs. In addition, the mechanism used to measure link latency is considered a scalable solution for dynamic, large and low-rate SDN networks.

REFERENCES

[1] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 3rd Quart., 2014.

[2] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Dec. 2014.

[3] P. Goransson, C. Black, and T. Culver, *Software Defined Networks: A Comprehensive Approach*. Burlington, MA, USA: Morgan Kaufmann, 2016.

[4] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, and L. Singh, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, 2013.

[5] S. Khan, A. Gani, A. A. Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 303–324, 1st Quart., 2017.

[6] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in OpenFlow-based software defined networks," *Comput. Commun.*, vol. 77, pp. 52–61, Mar. 2016.

[7] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2015, pp. 635–640.

[8] A. Nehra, M. Tripathi, M. S. Gaur, R. B. Battula, and C. Lal, "TILAK: A token-based prevention approach for topology discovery threats in SDN," *Int. J. Commun. Syst.*, vol. 32, no. 17, pp. 1–26, 2019.

[9] R. Wazirali, R. Ahmad, and S. Alhiyari, "SDN-OpenFlow topology discovery: An overview of performance issues," *Appl. Sci.*, vol. 11, no. 15, p. 6999, Jul. 2021.

[10] A. Nehra, M. Tripathi, M. S. Gaur, R. B. Battula, and C. Lal, "SLDP: A secure and lightweight link discovery protocol for software defined networking," *Comput. Netw.*, vol. 150, pp. 102–116, Feb. 2019.

[11] T. Alharbi, M. Portmann, and F. Pakzad, "The (in)security of topology discovery in software defined networks," in *Proc. IEEE 40th Conf. Local Comput. Netw. (LCN)*, Oct. 2015, pp. 502–505.

[12] X. Zhao, L. Yao, and G. Wu, "ESLD: An efficient and secure link discovery scheme for software-defined networking," *Int. J. Commun. Syst.*, vol. 31, no. 10, pp. 1–18, Jul. 2018.

[13] A. Azzouni, R. Boutaba, N. T. M. Trang, and G. Pujolle, "SOFTDP: Secure and efficient OpenFlow topology discovery protocol," in *Proc. IEEE/IFIP Netw. Operations Manage. Symp. (NOMS)*, Apr. 2018, pp. 1–7.

[14] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.

[15] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, 1st 2014.

[16] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 623–654, 1st Quart., 2016.

[17] O. Salman, I. H. Elhajj, A. Kayssi, and A. Chehab, "SDN controllers: A comparative study," in *Proc. 18th Medit. Electrotechnical Conf. (MELECON)*, Apr. 2016, pp. 1–6.

[18] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in software-defined networking (SDN)," *Comput. Netw.*, vol. 112, pp. 279–293, Jan. 2016.
[19] *Open Networking Foundation*. Accessed: Jun. 11, 2022. [Online]. Available: https://opennetworking.org/
[20] I. Alsmadi and D. Xu, "Security of software defined networks: A survey," *Comput. Secur.*, vol. 53, pp. 79–108, Sep. 2015, doi: 10.1016/j.cose.2015.05.006.
[21] J. Flathagen and O. I. Bentstuen, "Proxy-based optimization of topology discovery in software defined networks," in *Proc. Int. Conf. Military Commun. Inf. Syst. (ICMCIS)*, 2019, pp. 1–5.
[22] J. Wang, Y. Tan, and J. Liu, "Topology poisoning attacks and countermeasures in SDN-enabled vehicular networks," in *Proc. IEEE Global Commun. Conf.*, Jan. 2020, pp. 1–6.
[23] *Tcpdump*. Accessed: Jan. 18, 2022. [Online]. Available: https://www.tcpdump.org/manpages/tcpdump.1.html
[24] D. Smyth, S. McSweeney, D. O'Shea, and V. Cionca, "Detecting link fabrication attacks in software-defined networks," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2017, pp. 1–8.
[25] V. Shah, R. Patel, and R. Nayak, "Short range inter-satellite link for data transfer and ranging using IEEE802.11n," *Int. J. Comput. Appl.*, vol. 164, no. 1, pp. 23–25, Apr. 2017.
[26] S. S. Baidya and R. Hewett, "Link discovery attacks in software-defined networks: Topology poisoning and impact analysis," *J. Commun.*, vol. 15, no. 8, pp. 596–606, 2020.
[27] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: Attacking an SDN with a compromised openflow switch," in *Proc. 19th Nordic Conf. (NordSec)*, Tromsø, Norway. Cham, Switzerland: Springer, Oct. 2014, pp. 229–244. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-11599-3_14
[28] W. M. Syahidillah. *Multipath Routing SDN Controller*. Accessed: Jan. 18, 2022. [Online]. Available: https://github.com/wildan2711/multipath
[29] *Scapy*. Accessed: Jan. 18, 2022. [Online]. Available: https://scapy.net/
[30] E. Marin, N. Bucciol, and M. Conti, "An in-depth look into SDN topology discovery mechanisms: Novel attacks and practical countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1101–1114.
[31] T. Alharbi, M. Portmann, and F. Pakzad, "The (in)security of topology discovery in OpenFlow-based software defined network," *Int. J. Netw. Secur. Appl.*, vol. 10, no. 3, pp. 1–16, May 2018.
[32] L. D. Chou, C. C. Liu, M. S. Lai, K. C. Chiu, H. H. Tu, S. Su, C. L. Lai, C. K. Yen, and W. H. Tsai, "Behavior anomaly detection in SDN control plane: A case study of topology discovery attacks," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, 2020, pp. 357–362.
[33] A. Alimohammadifar, S. Majumdar, T. Madi, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi, "Stealthy probing-based verification (SPV): An active approach to defending software defined networks against topology poisoning attacks," in *Proc. Eur. Symp. Res. Comput. Secur.*, vol. 11099, 2018, pp. 463–484.
[34] R. Skowyra, L. Xu, G. Gu, V. Dedhia, T. Hobson, H. Okhravi, and J. Landry, "Effective topology tampering attacks and defenses in Software-Defined networks," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2018, pp. 374–385.
[35] X. Wang, N. Gao, L. Zhang, Z. Liu, and L. Wang, "Novel MITM attacks on security protocols in SDN: A feasibility study," in *Proc. 18th Int. Conf. Inf. Commun. Secur. (ICICS)*, Singapore, in Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 9977. Cham, Switzerland: Springer, Nov./Dec. 2016, pp. 455–465. [Online]. Available: https://www.semanticscholar.org/paper/Novel-MITM-Attacks-on-Security-Protocols-in-SDN%3A-A-Wang-Gao/890f1e8155f2e63506edde3542de129860256ebf
[36] X. Huang, P. Shi, Y. Liu, and F. Xu, "Towards trusted and efficient SDN topology discovery: A lightweight topology verification scheme," *Comput. Netw.*, vol. 170, Apr. 2020, Art. no. 107119, doi: 10.1016/j.comnet.2020.107119.
[37] Y. Jia, L. Xu, Y. Yang, and X. Zhang, "Lightweight automatic discovery protocol for OpenFlow-based software defined networking," *IEEE Commun. Lett.*, vol. 24, no. 2, pp. 312–315, Feb. 2020.
[38] F. Hauser, M. Schmidt, M. Haberle, and M. Menth, "P4-MACsec: Dynamic topology monitoring and data layer protection with MACsec in P4-based SDN," *IEEE Access*, vol. 8, pp. 58845–58858, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9044731/
[39] A. Kuamr and S. Shuka, "Topology validator—Defense against topology poisoning attack SDN," in *Quality, Reliability, Security and Robustness in Heterogeneous Systems* (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), vol. 402, X. Yuan, W. Bao, X. Yi, and N. H. Tran, Eds. Cham, Switzerland: Springer, 2021.
[40] J. W. Tukey, *Exploratory Data Analysis*, vol. 2. Reading, MA, USA: Addison-Wesley, 1977. [Online]. Available: https://www.google.co.uk/books/edition/Exploratory_Data_Analysis/UT9dAAAAIAAJ?hl=en&gbpv=0&bsq=J.W.%20Tukey,%20Exploratory%20Data%20Analysis,%20vol.%202.%20Reading
[41] *Mininet*. Accessed: Jan. 18, 2022. [Online]. Available: http://mininet.org/
[42] *Open Vswitch*. Accessed: Jan. 18, 2022. [Online]. Available: https://www.openvswitch.org/
[43] *Wireshark*. Accessed: Jan. 18, 2022. [Online]. Available: https://www.wireshark.org/
[44] *RYU SDN Framework*. Accessed: Jun. 11, 2022. [Online]. Available: https://ryu-sdn.org/
[45] R. Peck, *Introduction to Statistics and Data Analysis*. Pacific Grove, CA, USA: Duxbury, 2001.
[46] M. Hubert and E. Vandervieren, "An adjusted boxplot for skewed distributions," *Comput. Statist. Data Anal.*, vol. 52, no. 12, pp. 5186–5201, 2008.
[47] X. Yang, W. Zhou, N. Shu, and H. Zhang, "A fast and efficient local outlier detection in data streams," in *Proc. Int. Conf. Image, Video Signal Process. (IVSP)*, 2019, pp. 111–116.
[48] *Iptables*. Accessed: Jan. 18, 2022. [Online]. Available: https://linux.die.net/man/8/iptables
[49] *Tcpreplay*. Accessed: Jan. 18, 2022. [Online]. Available: https://tcpreplay.appneta.com/wiki/overview.html
[50] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey, "Toward adaptive and scalable OpenFlow-SDN flow control: A survey," *IEEE Access*, vol. 7, pp. 107346–107379, 2019.

ISMAIL AL SALTI received the B.Sc. degree (Hons.) in computer networking from the University of Technology and Applied Sciences (UTAS), Oman, and the M.Sc. degree (Hons.) in computer networking technology from Northumbria University, U.K. He is currently pursuing the Ph.D. degree in software-defined networking (SDN) security with The University of Manchester, U.K. His research interests include software-defined networks, information security, and network security.

NING ZHANG received the B.Sc. degree (Hons.) in electronic engineering from Dalian Maritime University, China, and the Ph.D. degree in electronic engineering from the University of Kent, U.K. Since 2000, she has been with the Department of Computer Science, The University of Manchester, U.K., where she is currently a Senior Lecturer. Her research interests include security and privacy in networked and distributed systems, such as ubiquitous computing, electronic commerce, wireless sensor networks, and cloud computing with a focus on security protocol designs, risk-based authentication and access control, and trust management.