Computer Networks 150 (2019) 102–116

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

SLDP: A secure and lightweight link discovery protocol for software defined networking

Ajay Nehra[a,*], Meenakshi Tripathi[a], Manoj Singh Gaur[b], Ramesh Babu Battula[a], Chhagan Lal[c]

[a] Malaviya National Institute of Technology, Jaipur, India
[b] Indian Institute of Technology, Jammu, India
[c] University of Padova, Italy

ARTICLE INFO

Article history:
Received 22 August 2018
Revised 29 November 2018
Accepted 21 December 2018
Available online 25 December 2018

Keywords:
Topology discovery
Link discovery protocols
Network poisoning
Flooding attacks
Replay attacks
Software defined networks
Openflow protocol

ABSTRACT

In Software Defined Networks (SDNs), the global view of the underlying network topology is created and maintained at the logically centralized controller. SDN achieves it by decoupling the data plane from the control plane. The up-to-date global view at SDN controller enables the applications (running on top of it) to innovate through dynamic network programmability. To establish a global view, a controller needs to discover a physical topology of the underlying SDN network infrastructure, which is challenging due to various reasons such as the lack of SDN protocols standardization and authentication mechanisms, use of sub-optimal link discovery protocols (e.g., OFDP and LLDP), dynamic topology due to movement of virtualized data centers, switches, and multi-tenant cloud networks, and lack of integration of security schemes for the topology discovery.

In this paper, we propose a SDN Link Discovery Protocol (SLDP) for efficient discovery and extraction of topology information in SDN networks. The design of SLDP is motivated from the need of a secure, lightweight, and efficient link discovery protocol in SDN. SLDP aims to prevent, detect, and mitigate various security threats such as poison, replay, and flooding attacks, which are due to lack of source authentication, lack of packet integrity checks, and reuse of static packets. SLDP creates and maintains the global network topology at SDN controller by using smaller size and lower number of SLDP packets during the topology discovery process. Thus, it significantly minimizes the topology discovery overhead in the network. We implemented SLDP on Mininet emulator, and the results show the effectiveness and correctness of SLDP concerning topology discovery time, CPU computational time, and bandwidth overheads, when compared with the traditional OpenFlow Link Discovery Protocol (OFDP). Additionally, SLDP successfully prevent, detect, and mitigate various attacks (e.g., poison, replay, and flooding) in different SDN scenarios.

1. Introduction

For any data center, the essential requirements are robustness and manageability. Software Defined Network (SDN) offers programmability, flexibility, and openness to ensure these requirements [1,2]. Due to the separation of data plane and control plane, the SDN controller exhibits a global view of the underlying network topology, which enables it to make the optimal decisions for various applications that runs on top of the controller. For instance, applications such as load-balancing and shortest pathfinder, uses the global view to function efficiently. The view construction and maintenance require the discovery of underlying network topology, which consists of switches and links. The existing SDN controllers use OpenFlow Discovery Protocol (OFDP) with Link Layer Discovery Protocol (LLDP) packets for link discovery, which is prone to various security threats.

In SDN, the global view is generated by performing the switch discovery, the link discovery, and sometimes the host discovery. Once an OpenFlow-enabled switch connects to the network, it performs a TCP three-way handshake with a pre-stored remote socket residing at the SDN controller. After successful handshaking, both negotiate on the OpenFlow version. Subsequently, the switch is asked for its capabilities and ports status. These steps help controller to discover the switch with available ports. To perform various topology-aware activity, link discovery is mandatory. Most of the SDN controllers use OpenFlow Link Discovery Protocol (OFDP) and Link Layer Discovery Protocol (LLDP) for the discovery process. An LLDP packet is generated at the controller, and sent to a

* Corresponding author.
E-mail addresses: 2014rcp9553@mnit.ac.in, ajay.nehra@hotmail.com (A. Nehra), mtripathi.cse@mnit.ac.in (M. Tripathi), rbbattula.cse@mnit.ac.in (R.B. Battula).

https://doi.org/10.1016/j.comnet.2018.12.014
1389-1286/

switch with the forwarding instruction [3,4]. When a switch receives such a packet, it consults with flow entry and forward the LLDP packet to the controller. The controller puts source information in the LLDP packet, and it also receives destination information on successful reception of the same packet. This information is used to create a unidirectional link. The same process is repeated in the discovery of each unidirectional link. To discover the hosts in topology, the controller uses traffic from hosts. Such host discovery is controller dependent.

1.1. Motivation and contributions

Each controller implements a variant of OFDP with variation in base LLDP packet. The main focus of the controller is link discovery between the Openflow switches (OF-switches). Few controllers also provide some security extensions and the latency information for each link. Major security threats are LLDP Replay, LLDP Poison, and LLDP flooding, which are caused due to the lack of source authentication, lack of integrity check, and reuse of static packets. Some SDN controllers suggest the inclusion of a hash field in LLDP packets to perform the integrity check. In particular, the POX [5], RYU [6], and ONOS [7] controllers does not provide any security, while the OpenDayLight [8], Floodlight [9], and HPEVAN [10] introduce hash in LLDP packet to prevent the attack. The state-of-the-art also proposes few security solutions such as authors in [11] suggest that instead of sending LLDP packet to every port on a switch, send a template to switch with an instruction to generate separate LLDP packet to each port because it reduces controller-to-switch traffic at the cost of slight switch overhead, SPHINX [12] proposes the use of static binding to prevent the attack, Topoguard [13] and [14] proposes to introduce key-based hash in LLDP packet for making it more secure. ESLD [15] proposes a secure, efficient topology discovery protocol, in which lower number of LLDP packets are generated with MD5 based authenticator. Few switch agent-based distributed discovery protocols are also proposed in [16–18]. However, these security solutions are not able to address all the LLDP based threats. More precisely, the detection of LLDP attacks is partially addressed and the prevention is entirely missing in the state-of-the-art. The LLDP packets are traditionally used with the objective to advertise identity, capability, and neighbors information to a neighbor, and the same is adopted in SDN. Thus, it forces to use the large size LLDP packet with a non-optimal use of packet space. SLDP shows that the packet size reduction can be done without affecting the desired functionality. In current practices, LLDP packet is generated and sent to each switch port, and some of these ports are attached to end hosts. Hence, the packets for these ports are of no use as it is shown with SLDP's link discovery mechanism.

In this paper, we propose SLDP, which is a novel link discovery protocol for SDN networks. SLDP contains three levels of security. Additionally, it is lightweight because of its new link discovery packet structure, and it is more efficient as less number of SLDP packets are generated and transmitted. We present the design of SLDP which includes an SLDP packet structure, system architecture, and event sequence. Traditional LLDP packet has some of type-length-values (TLVs) for no use in SDN like time to live (TTL) and EndTLV. Few TLVs which are useful takes some extra bits like field type, sub-type, and length. SLDP introduces fixed length positional packet structure. For security, SLDP uses token based prevention approach to prevent poison, replay, and flooding attacks. Even for low probable attacks, some more levels are introduced like poison detection with mitigation, and flood detection with mitigation. Initially the controller sends SLDP packet to each port of a switch, but in the later iterations some of the ports are declared as non-eligible. This helps the controller to generate and send less number of SLDP packets. SLDP is implemented on Mininet [19] environment with RYU controller. The time taken in link discovery packets creation and verification is far less then RYU's original OFDP implementation. The topology discovery performed is quick in SLDP as compare to OFDP. This is achieved because lightweight packets are sent only to the required ports. Lower number of and size of packets reflects in lower CPU and bandwidth resource utilization. In particular, this paper has the following major contributions:

- We propose a secure, lightweight, and efficient link discovery protocol (SLDP) for SDN networks. We present the design of the major components of SLDP, which includes a new packet format, system architecture, flow entry structure, and event sequence. To ensure the security against various attacks, SLDP uses a token-based technique that generates random source MAC addresses for SLDP packets, and it uses the randomness to create a flow entry for the SLDP packets.
- We fully implemented SLDP on Mininet emulator with RYU controller. The performance analysis done on different SDN scenarios shows the effectiveness of SLDP regarding prevention and detection of various attacks (e.g., replay, flooding, and poison attacks), computational overhead, topology discovery time, and bandwidth consumption. We also compare SLDP with OFDP protocol to show the effectiveness of SLDP over the state-of-the-art.

1.2. Organization

The rest of the paper is organized as follow. In Section II, we present overview of traditional link discovery protocol of SDN, and the related work which includes the state-of-the-art research efforts on secure and efficient link discovery protocols in SDN networks. Additionally, Section II also includes a discussion on the attack vector and manifestation of attacks related to link discovery in SDN. In Section III, we present details of our proposal (i.e., SLDP) which includes the motivation for the design of SLDP, its desired characteristics, system architecture, and security analysis for various LLDP-based attacks. The evaluation setup details and the performance evaluation of SDLP protocol on various SDN topologies is presented in Section IV. Finally, we conclude our paper with future research directions in Section V.

2. Background and related work

In a traditional network, each networking device has its local view to take various decisions such as related to packets forwarding, and to perform flow or access control. One of the key promises of SDN is to use a global view to take effective decisions. The global view is a view of the physical arrangement of network components which includes switches, hosts, and links that are administrated by controller(s). Global view helps the controller to make the suitable or optimal decision. Applications such as load balancing, shortest pathfinder, end to end delay guarantor, best path selector, to name a few, need the global view to make an optimized decisions [20]. If the exact topology is not known, it's hard for the controller to make effective decisions. A global view discovery consists of link discovery, which we address in this article. Other discoveries in SDN are switch discovery and host discovery, which are part of the topology as it is shown in Fig. 1. There are various links in topology, but link discovery only identify links between two OpenFlow switches (OF-Switches).

2.1. Existing implementations

link discovery is a process to identify links between OF-Switches. There are certain possible cases in which non-OF-Switches separates two OpenFlow enabled switches. The controller

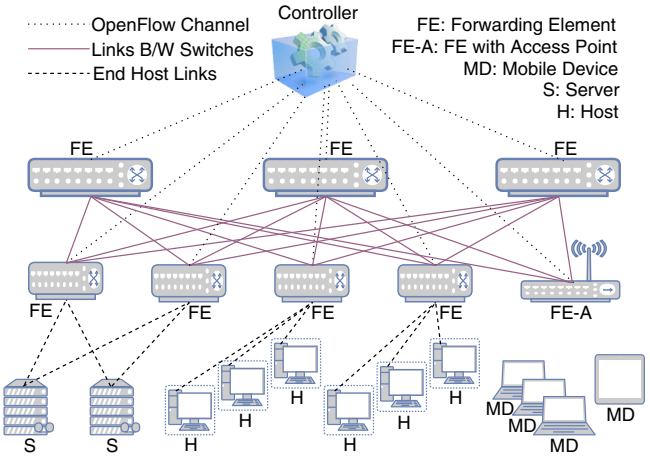

Fig. 1. Global view construction in SDN.

does not control a non-OpenFlow switch, hence, we assume that it works as per specifications. However, a link discovery process must identify the links. In SDN, so far the link discovery is performed with a non-standard OpenFlow Discovery Protocol (OFDP) with Link Layer Discovery Protocol (LLDP) packets. Each controller, e.g., POX [5], RYU [6], FloodLight [9], OpenDayLight [8], ONOS [7], and HPEVAN [10] implement their own OFDP variant. However, all of these are using either LLDP packets or alike Broadcast Domain Discovery Protocol (BDDP) packets. As shown in Fig. 3, LLDP/BDDP packets are initiated by the controller, pass through switches, and confirms a link between switches.

Before understanding link discovery in SDN environment, let's look at LLDP packet format which is depicted in Fig. 2. The LLDP packet is a collection of different set of Time-Length-Value (TLV)s. Different controllers maintain a different set of TLVs. The Chassis, Port, and TTL TLVs are generally used to store source data path id (*dpid*), source port number, and packet expiry time. It is Because LLDP [21] is originally used in traditional networks, and SDN is just using the similar packet format. Thus, each combination of TLVs serve different purposes. In next section, we will show that few of TLVs are not required for the topology discovery process.

As per Fig. 3, LLDP packets are generated for each port on each OF-switch (step 1). PACKET_OUT packet with LLDP packet as a payload is generated. After reaching at the switch, payload LLDP packet is transferred on a specified output port (step 3). If such a LLDP packet is received at the switch, its forwarding table is consulted for further action. After consultation, a PACKET_IN message is generated with received LLDP as payload (step 4). In step 5, the controller receives PACKET_IN message with event information or destination dpid and destination port. Because the LLDP packet holding source dpid and source port, the controller has complete source to destination information on which the LLDP packet is traveled. This information helps to create a unidirectional link between source and destination. The process will be completed for each unidirectional link to provide a complete set of unidirectional links.

Each controller uses its own variant of OFDP and LLDP packets. In each implementation of OFDP, a received LLDP packet is validated against some parameters such as destination MAC, contents of Chassis Id, and setting of certain fields. Here the validation means whether the received LLDP packet is genuine or not. Different LLDP packets have a different set of TLVs, and each TLV serves a specific purpose. In POX controller, a System Description TLV is available along with mandatory TLVs holding 'dpid:3' like string. Upon receiving LLDP packet, the parsing algorithm uses destination MAC address, number of TLVs, and TLV sequence for validation. After successful validation, each OFDP implementation extract source information. The source information is extracted from Port and System Description TLV. The RYU controller uses only mandatory fields in LLDP packet. The Chassis Id and Port TLVs are used as validation and source information extraction. The OpenDayLight controller also uses few TLVs to ensure security along with mandatory TLVs.

In OpenDayLight LLDP parsing algorithm, System Name TLV is checked against not nullness to ensure packets validity. After validation, a hash is extracted and matched with stored hash to ensure packet's integrity. FloodLight controller also uses security TLV with a time-stamp TLV to calculate link latency. Port TLV length is used to ensure validness of LLDP packet. The stored hash is matched against a received hash to ensure integrity. A received time-stamp is subtracted from current time-stamp to calculate latency. ONOS controller using two separate TLV along with mandatory TLVs. In the first and second TLV 'ONOS Discovery' and 'of:0000000000000003' is stored respectively. The later one is the dpid of an OF-switch. Source information is extracted from Chassis and Port TLVs in parsing algorithm. The HPEVAN controller uses a TLV to store hash value in it. The destination MAC address is used to verify LLDP packet. Once parsing algorithm finds that received hash is same as a stored hash, the source information is extracted from System Description and Port TLVs to create a unidirectional link. For a detailed analysis of each implementation of OFDP, the structure of each LLDP packet, and hash calculation please refer to our recent work [22]. The summary of conditional TLV and source of information extractor is represented in Table 1.

2.2. Related work

Apart from present deployment of OFDP and LLDP packet in the various controllers, the research community is also putting hard efforts to make efficient and secure link discovery. Authors in [20] proposes ForCES based link discovery which provides additional computational capabilities to run LLDP at the switch level. The controller queries periodically to gather the topology information. Authors in [16] propose a switch agent based topology discovery mechanism. Initially, the controller generates and send a multicast message, called TDP-Request. Upon receiving, the switches change from Standby to either Father nodes or Active node. Each node collect neighbor information but only Father nodes send the information asynchronously to the controller. SPHINX [12] utilizes an abstraction of flow graphs, which is produced by PACKET_IN and FEATURES_REPLY messages. Same flow graphs are employed to validate all the network updates. Authors in [17] proposes SDN-RDP, a distributed resource discovery protocol. More than one controllers manages various switches, hence the proposed protocol works in two phases, one for the controllers announce and another for joining the switches. For each event, packets are moved in various network entities to form the topology. SHTD [18] is a layer two topology discovery with autonomic fault recovery protocol. Topology discovery is performed with controller sending a topoRequest message. The propagation of this multicast message with nodes in four roles, i.e., non-discovered, leaf, v-leaf or core and each port in four states, i.e., standby, parent, child or pruned discover the topology. Autonomic fault recovery is performed with the help of managed components and autonomic manager. The autonomic manager detects the port status to make the update for managed components.

Authors in [11] suggest a unique variant of topology discovery in which LLDP packets are only generated as one for each switch instead of each port on each switch. It is quite simple but reduces the number of LLDP packet from the controller to switch drastically. TopoGuard [13] classifies packet integrity check failure and source authentication failure as possible reasons for LLDP packet

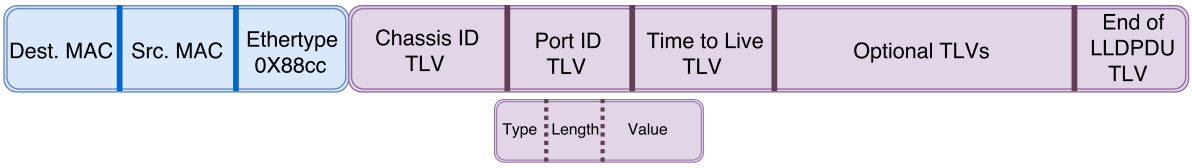

Fig. 2. LLDP/BDDP packet format.

Table 1
Summary table for different controllers.

Controller(Ver.)	Conditional Tlvs					Information Source	
	TlvCnt	CTlv	PTlv	TTlv	SDTlv	Dpid	Port
POX(0.2.0)	√	√	√	√		*CTlv*	*PTlv*
RYU(4.12)		√	√			*CTlv*	*PTlv*
OpenDayLight(3.0.7)					√	*CTlv*	*PTlv*
FloodLight(1.2)			√			*Uk1Tlv*	*PTlv*
ONOS(1.9.0)		√				*CTlv*	*PTlv*
HPEVAN(2.7.18)			√			*Uk1Tlv*	*PTlv*

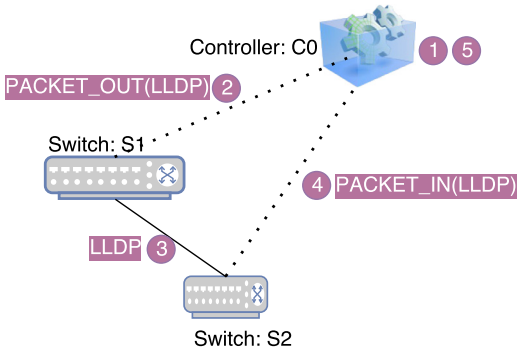

Fig. 3. LLDP/BDDP movement for link discovery.

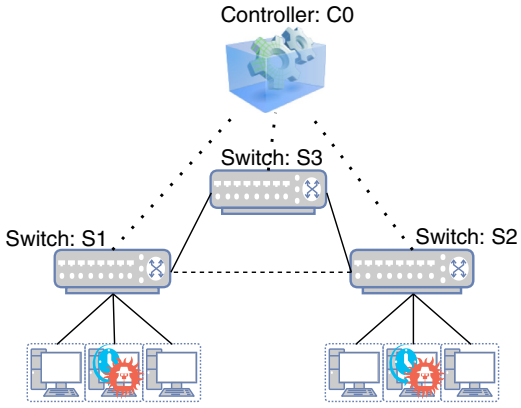

Fig. 4. Topology for attack vector.

based threats. The author also suggests that the controller have to stop sending LLDP packets to any host. To prove LLDP packet integrity and authentication of origin, an HMAC TLV is appended to the LLDP packet. OFDP_HMAC [14] introduces dynamic HMAC authentication to ensure both integrity and authentication. The calculated HMAC is attached to each LLDP packet. The dynamic key for HMAC is used to prevent LLDP Poison/Replay attacks. ESLD [15] is a secure, efficient topology discovery, which works in stages that includes port classification, LLDP packets sending, and link detection. Our proposed approach uses host information to identify desired ports, hence lower number of packets are generated and sent. LLDP packets are generated for identified port with key based MD5 as the authenticator to each packet to prevent replay and poison attacks. All links are discovered with LLDP packet propagation.

2.3. Attack vector

The LLDP packets are generated for each port on each switch. These packets are reached to target switch, the switch forwards them to the controller with the help of a dedicated flow entry or default miss entry. The major threats for the link discovery are Replay, Poison, and Flooding [23] attacks. Let's investigate these attacks with the help of Fig. 4, which consists of three OpenFlow switches, switches *S*1 and *S*2 have three hosts attached to each of them. The two hosts are malicious, e.g., either a person with malicious intentions is operating them or malicious application are installed on them. The switches *S*1 and *S*2 are connected with *S*3, and the dash line between *S*1 and *S*2 represents a fake link.

Replay Attack (RA): Due to LLDP packet propagation to each port on a switch, the attached hosts also receive LLDP packets. However, if one of the attached host send a received LLDP packet from some other host that is attached to another switch, than the receiving switch and the controller has no way to identify the source of that LLDP packet. For instance, if a malicious host at switch *S*1 receives LLDP and share it with malicious host attached to *S*2, then the host at *S*2 send the LLDP packet on local port to *S*2, , and again the *S*2 sends the received LLDP packet to *S*1 with its malicious host. Finally, *S*1 and *S*2 will send these packets to the controller, and the controller confirms that *S*1 and *S*2 have a direct link.

Poison Attack (PA): The attacker creates fake LLDP packets and sent it to the attached switch. The switch is unable to differentiate genuine and fake packets, thus the packets are sent to the controller. The controller also has no way to find the source of the packet and it cannot evaluate the integrity of the packet. In this way, a generated false link creates topology poison. For instance in Fig. 4, if an attacker at *S*1 creates an LLDP packet containing information like Chassis id is 2 and port is 3, then the switch would forward it to the controller. The controller makes a unidirectional link between *S*2 to *S*1.

Flooding Attack (FA): When a controller receives a fake crafted LLDP packet, it computes logic. Hence when an attacker sends a flood of LLDP packets, e.g., 50,000 packets per second, the resource consumption at controller increases rapidly, and it negatively effects the benign packets service rate. Additionally, these large number of fake LLDP packets waste switch to controller bandwidth and controller CPU cycles.

The possible cause of these attacks are failing to check source authentication of the received packet, and failing to verify the integrity of packet, static content and constant flow entry. For instance, if a is controller unable to identify the source of LLDP packets, and the LLDP packet holds static content then the controller is prone to replay attack. If the controller is unable to identify a source of LLDP packets, fail to check the integrity of LLDP packet, and the LLDP packet have static content, then poison attack may happen. If the switch has a constant flow entry to pass LLDP packets to the controller then flooding attack may happen [22].

2.4. Manifestation of attacks

Effects of described attacks are ranging from a fabricated link to controller fingerprinting, which leads to wastage of network resources.

Fabricated Link: Replay and poison attacks create fabricated links. Any such fabricated link poison the topology, which effects topology aware applications, e.g., load balancer.

Controller Fingerprinting: If a controller generates and sent LLDP packets to each port, the host also receives these LLDP packet. Each controller have different LLDP packets [22] which make the easy guess for controller identification.

Resource Wastage: In flooding attack, the controller receives large number of LLDP packets, and the processing of these packets waste controller CPU cycles. These packets traverse from switch to the controller over TCP/TSL layer, hence it causes bandwidth wastage. When OF-switch is configured with TSL to ensure security, a large amount of computation has to be done to encrypt the fake LLDP packets, this further increases the resource consumption at switches and controller due to encryption/decryption process.

3. The SLDP protocol

In the previous sections, we have discussed various security vulnerabilities in existing controllers. In this section, first we provide motivations for a new design of a link discovery protocol. In particular, we propose a SDN Link Discovery Protocol (SLDP). The design of SLDP consists of SLDP packet format, system architecture, and event sequence. We also defines SLDP characteristics along with SLDP packet structure in this section.

SLDP discovers a unidirectional link between two OpenFlow enabled forwarding elements, which may also be separated by a non-OpenFlow switch. The protocol is a lightweight, efficient, and secure solution for discovering links in SDN physical topology. The discovered links along with the switch information is used to create a global view at the controller.

3.1. SLDP Design motivations

A link discovery protocol can be considered good if it discovers link as quickly as possible, and it is secure against known threats while consuming less bandwidth and CPU cycles. Below, we provide adequate evidence that motivates us to design a new link discovery protocol in SDN.

Security: A secure link discovery in SDN ensures accurate topology discovery, minimum bandwidth and minimum CPU resource wastage. With the current security extensions, no optimal uses of bandwidth and CPU is acheived. In the previous section, the article defines security threats and their effects. The security threats are possible due to the controller's inability to perform source authentication, packet integrity check, and static packet creation.

Table 2 illustrates security strength of controllers against LLDP Poison (LP), LLDP Replay (LR), LLDP flooding (LF) attacks. RC (Row Craft) and LC (Library Craft) are two classes of attacks which are used to perform the security attacks on topology in SDN. In RC, the attacker uses LLDP packet information and parsing algorithm to perform the attack, while in LC the attacker uses a library of the controller to perform the attack. Few controllers (e.g., FloodLight, OpenDayLight, and HPEVAN) attach a hash to the LLDP packets. To perform the attack on these controllers, the library information is needed, which may have the information about the hash generation algorithm (or a secure static key) that is required for the successful attack. To understand more about how attacks can happen on different controllers, please refer to [22].

Table 2
Different controllers with attack vector.

Controller(Ver.)	Vulnerability	LLDP Attack			Attack
		LP	LF	LR	Type
POX(0.2.0)[5]	No hash	√	√	√	*RC*
RYU(4.12)[6]	No hash	√	√	√	*RC*
OpenDayLight(3.0.7)[8]	Static hash	√	√		*LC*
FloodLight(1.2)[9]	Static hash	√	√		*LC*
ONOS(1.9.0)[7]	No hash	√	√	√	*RC*
HPEVAN(2.7.18)[10]	Static hash	√	√	√	*RC*

POX, RYU, and ONOS has no security content in their generated LLDP packets, hence these controllers are prone to attacks. The OpenDayLight controller generates an MD5 hash of a string, i.e., 'openflow:1:2', which an attacker can also create if she knows the MD5 library and a secret key. Both the information can be achieved, if an attacker performs reverses engineering to the controller code. Hence, having static hash key won't help to protect from such attacks. In Floodlight controller, local machine interfaces are required to calculate the hash, hence the attacker is not able to compute the same hash on the local machine. But once the hash is calculated, it is kept same for further LLDP packets. The attacker has an opportunity to put the same hash in fake crafted LLDP packets. In case of HPEVAN, same mistake is repeated for all the packets, and same hash is used. Hence, an attacker can copy and use the same fake LLDP packets. A point to note in OpenDayLight is that hashes for each switch port are separate, but in case of FloodLight and HPEVAN hash for each LLDP packet is same. In a nutshell, most of the industry and academic grade controller are insecure, specifically against the LLDP-based threats.

The research community is also trying to secure the link discovery process. Table 3 provides the summary of such efforts. TopoGuard [13] identifies source authentication and integrity check failures as possible reasons for the described threats. Their proposed approach only considers LLDP based poison attack but not the remaining attacks. SPHINX [12] detects the poison attack with static mapping of ports with hosts. OFDP_HMAC [14] identifies correct reasons, but still, packets are broadcast to each port. LLDP flooding is not detected or prevented. All approaches are detection and mitigation based, hence leaves scope to look for some preventive measures which can also reduces resource consumption during the attacks. In conclusion, some security improvements are possible. For instance, ESLD [15] only generates LLDP packet for non-host ports on each switch. But the approach is based on host traffic, and an attacker can forge its behavior. Also key-based hash is sent to each packet, which is time-consuming.

Lightweight: Currently, link discovery in SDN is performed with controller specific OpenFlow Discovery Protocol (OFDP) implementation and LLDP packets. In traditional networks, LLDP packet is designed to be a vendor-neutral link layer protocol for LANs. LLDP is used to advertise network elements' identity, capabilities, and neighbors. Different fields in LLDP packet are Chassis ID, Port ID, Time To Live, Port description, System name, System description, System capabilities, Custom TLV, and End of LLDPDU. First three and last one are compulsory, and the rest are optional. Each networking device is supposed to run an LLDP agent. The LLDP agent

Table 3
Comparison of different research proposals for security.

Approach	Authentication	Integrity	LLDP broadcast	Poison	Flood	Replay
TopoGuard[13]		y	y	y		
SPHINX[12]			y	y		
OFDP_HMAC[14]	y	y	y	y		y
ESLD[15]		y		y		y

Table 4
Length in Bytes for different LLDP packets.

Deployments	Size of link discovery frames(bytes)
POX(0.2.0)	41
RYU(4.12)	40
OpenDayLight(3.0.7)	85
FloodLight(1.2)	75
ONOS(1.9.0)	66
HPEVAN(2.7.18)	67
Target	$14 + 8 + 4 = 26$

Table 5
Number of Switches,Links,Ports and Hosts.

Topology	Switch	Link	Port	Host	eligible ports
tree,4,4	85	340	424	256	168
tree,7,2	127	254	380	128	252
fat tree	80	384	705	64	641

gathers remote device information and advertise local information to remote devices. The collected data is stored in management information database (MIB) and it can be queried with the Simple Network Management Protocol (SNMP).

Currently, SDN community is using a borrowed packet format to perform link discovery. LLDP packet structure has some features which can not fit anywhere in the picture with SDN's topology discovery phase. Firstly, we need chassis id and port id for source information. TTL field in LLDP protocol is used to instruct remote LLDP agent for validness of the information. But in SDN, no LLDP agent is installed in any of the forwarding elements. Hence, the TTL field is not required. For validness of link in SDN, the controller send periodic packets. In SDN, no forwarding element is interested in name or capabilities of neighbour elements. It is because the controller takes care of this information and receives it in first few packets of communication with forwarding elements. Also, the end of LLDP packet is not needed if we can provide a fix length packet for discovery.

In TLV structure, the LLDP have to insert various types and subtypes along with the length. If we need fixed length chassis id (dpid) and port id, then no need to put all type and subtype values. The OpenFlow specification specifies the length of dpid and port id as eight and four bytes respectively. Because type, subtype, and length field also consuming length, it's good to remove this structure if it is not required for the topology discovery process.

Table 4 gives length taken by each LLDP frame generated by different SDN controllers. We can see that if we only need 26 bytes then why to go for 40 bytes in RYU and 85 bytes in the OpenDayLight controller. A reader may argue that controller might be storing some hash value in it. Yes, we agree but not ready to recommend it because we can see in Table 2 that all controllers are vulnerable then what is the use of storing the hash. Secondly, computation of hash is costly process. It will be more worse, if the controller has to calculate a different hash for each LLDP packet.

Efficient: The link discovery process works as follow, the controller generates LLDP packets and send them to all switches. The receiving switch forwards these packets to the instructed ports. If any switch receives a LLDP packet from its neighbor switch, it forwards the packet to the controller. The controller parses the received LLDP packet and creates a link. In present deployments, LLDP packets are generated for each port on each switch. If some of the ports are attached to hosts, LLDP packets for those ports are of no use. If the controller has to generate less number of LLDP packet, fewer resources will be consumed at the controller. Same arrangement will also prevent a malicious host to perform controller fingerprinting.

Table 5 shows statistics about the number of switches, number of hosts, number of ports, and number of links in few SDN topologies. For the structure of topologies, please consider Section IV. In tree,4,4 topology, the controller has to generate 424 LLDP packets to discover 340 links. In the same topology 256 host are there, which means 256 generated LLDP packets are of no use. The last column in the table specifies non-eligible ports for LLDP packet. In particular, a controller can reduce resource consumption by reducing the number of LLDP packets.

In this section, we demonstrated with various examples that a lightweight, efficient, and secure link discovery is still needed to achieve optimal results with lesser resources in SDN.

3.2. Desired characteristics

SLDP aims to discover links in more secure, efficient, and lightweight way. More precisely, SLDP will work correctly if holding following characteristics:

Definition 1 (Correctness):

- SLDP must discover the link between two OpenFlow enabled switches.
- SLDP must discover the link between two OpenFlow enabled switches separated with a non-OpenFlow switch.
- SLDP must provide latency for each discovered link.
- SLDP should secure against replay, poison, and flooding attacks. **(Secure)**
- SLDP packet size must be kept to minimum. **(Lightweight)**
- SLDP must perform link discovery with less number of packets. **(Efficient)**

If the discovery process is secure, the controller will work as intended, hence, better controller resource utilization. If the link discovery is done with lightweight packets (i.e., lower packet size and less number of packets to discover the topology), then the discovery process uses less bandwidth and light traffic on network interfaces. If discovery process is efficient, system requires less bandwidth and CPU resources to generate the discovery packets. In particular, the SDLP's theoretical and practical analysis gives confidence towards its correctness. We theoretically examine some of the stated correctness one by one. Security-related correctness is analyzed with test cases that we will discuss in the later section. Furthermore, the experimental setup and evidence are demonstrated in Section V.

SLDP must discover the link between two OF-switches. For instance, all unidirectional links in a topology belongs to a set $L = \{l_{12}, l_{21}, l_{23}, l_{32} \ldots . l_{mn}, l_{nm}\}$, and SLDP's discovered links belong to set $D = \{d_{12}, d_{21}, d_{23}, d_{32} \ldots . d_{mn}, d_{nm}\}$. The following condition

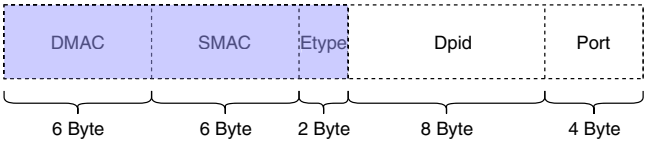

Fig. 5. SLDP packet format.

must be followed $\forall x[x \in L \Leftrightarrow x \in D]$, i.e., all elements which belong to L must also belong to D. SLDP packets generated at the controller are sent to a switch with OpenFlow TCP/TLS channel with forwarding instructions. The switch obeys instruction and a linked switch receives SLDP packet. The receiving switch consults flow table to forward it to the controller. SLDP use the randomized information to create both flow entry and SLDP packet. Here one possible problem is to be considered which is called `race condition`. If an SLDP packet reaches to the second switch before flow entry packet, it must be dropped. To prevent the race condition, SLDP uses Barrier message [24]. The barrier message ensures the order of flow entry installation and SLDP packet sent instructions. It confirms clarification of a single doubt while SLDP discovers the link.

SLDP must discover the link between two OF-switches that are separated with a non-OpenFlow switch. Layer two or three switch forwards layer two broadcasts. SLDP Ethernet frame uses broadcast destination address. Hence, once a broadcast packet is received by layer two or three device, it must be forwarded to all of its ports. An OF-switch is connected to one of its port so it will receive the packet. Packets are received over non-OpenFlow switches also, however the remaining process for link discovery stays the same.

SLDP must provide latency for each discovered link. In SLDP, whenever a packet is sent for discovery, the time stamp is noted. After traversing two switches, the same packet comes to the controller. Upon receiving, SLDP records the second time stamp. The latency is calculated with both these time stamps. SLDP ensures that the discovery packet size stays to minimum. As Table 4 and Fig. 5 suggests, SLDP takes the minimum bits for a link discovery packet. SLDP uses 26 bytes, while others discovery protocols are taking in a range from 40 bytes to 85 bytes.

SLDP must perform link discovery with less number of packets. As Table 5 suggests, eligible ports are less than the number of total ports irrespective of the topologies. SLDP find out non-eligible ports after few iterations and remove it form eligible port list. For instance, a controller's OFDP implementation is generating 'o' bytes link discovery packet. SLDP implementation for the same controller is taking 's' bytes. If $p_1, p_2, p_3 \ldots p_n$ are ports of switches in a topology, and $h_1, h_2, h_3 . h_m$ are the hosts attached to switches on some of the ports. Here 'n' and 'm' are the numbers of ports and hosts attached to switches respectively. Hence, the total profit in terms of bytes can be represented as Eq. 1 as per the iteration, i.e., every five seconds.

$$\left\{ \sum_{i=1}^{n} p_i - \sum_{i=1}^{m} h_i \right\} \{o - s\} \tag{1}$$

3.3. SLDP Packet format

SLDP is designed to identify links between two OF-switches, which may be separated with a non-OpenFlow switch. For each discovered link, the latency is also provided. SLDP is secure against replay, poison, and flooding attacks. Lightweight and efficient link discovery is desirable. For all these features, SLDP uses a simple yet effective frame format, which can be seen in Fig. 5.

To understand SLDP working methodology, understanding of the SLDP packet structure is mandatory. Two partitions are shown in Fig. 5, first is Ethernet header and second is SLDP payload. SLDP

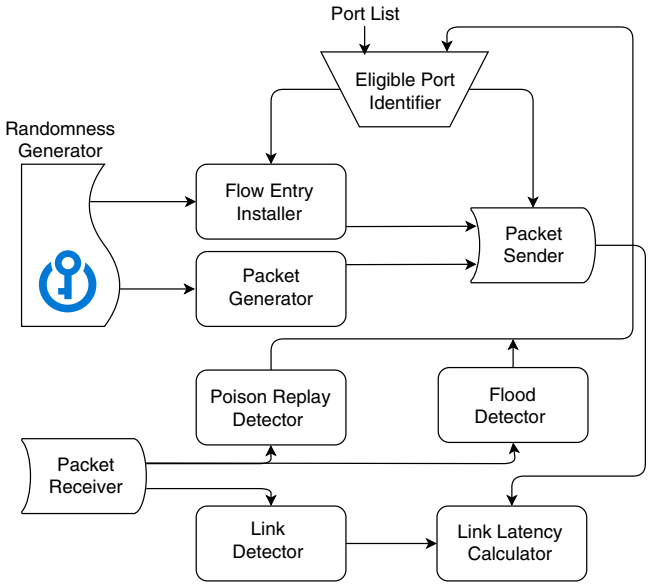

Fig. 6. SLDP System Architecture.

utilizes both partition to work correctly. In Ethernet header destination, a broadcast address is used. In source address field, a random MAC address is used. In EType a custom type is introduced. The dpid and port are the source information of the link. In a nutshell following information is used in SLDP.

DMAC → ff:ff:ff:ff:ff:ff
SMAC → Random MAC address
EType → 0xabcd
dpid → Source dpid
port → Source port

A reader may argue that how security and latency will be provided in SLDP with such a basic packet structure. For detail of security and latency, we will discuss these in the next section.

3.4. SLDP System architecture

SLDP system architecture explains about basic function blocks, which are designated for dedicated work. In Fig. 6, system architecture for SLDP is illustrated. Link discovery is a periodic activity, in each cycle, few operations needs to be performed, i.e., random MAC address generation, flow entry installation, SLDP packet generation and transmission. Randomness generator block will provide random MAC address, which is used in SLDP packets and flow entries. Randomness ensures all restrictions, which are applicable to a MAC address. A flow entry installer module installs flow entries in each OF-switches. Packet generator takes random source MAC and creates SLDP packet. Once flow entry is installed, the SLDP packet with the same randomness is allowed to pass to the controller. The packet sender node sends an SLDP packet to source switch. The Unique selling proposition (USP) of this approach is an eligible port identifier, which identifies eligibility for each iteration. Initially, all ports are considered eligible, but after each iteration, the list is updated. In SLDP whenever a switch awakes, its every port is added in the *ePorts* or eligible port list. Therefore, in the next cycle the SLDP packets are generated for each eligible port including the latecomers of the previous cycle, and the rest of the process for these latecomer ports remains the same.

An OF-switch receives an SLDP packet with instructions to forward it on a particular port. On the other end of the tunnel, when a OF-switch receives the SLDP packet and it consults the flow entry table. Due to flow entry installer, there will a flow entry, which forwards the packet to the controller. The packet receiver receives

the packet and validates it for SLDP packet. SLDP packet comes in wrapped in PACKET_IN packet. The header contains destination information for the link. Both source and destination information are used to form the link. Each packet is sent on a time stamp and receives on another time stamp such information is used to calculate latency in link latency calculator. If receiving packet alerts Poison, Replay, and Flood detector, the eligible port list will be updated.

Flow Entry Structure: In each SLDP packet cycle, a new flow entry with provided randomness is installed in each participating switch. Here an example of that flow entry is shown.

```
*** s1 ----------------------------------
NXST_FLOW reply (xid=0x4):
cookie=0x0, duration=1.687s, table=0, n_pa
ckets=2, n_bytes=52, hard_timeout=5, idle_
age=0, priority=65535, dl_src=96:66:e8:27:
9f:94, dl_dst=ff:ff:ff:ff:ff:ff, dl_type=0
xabcd actions=CONTROLLER:65535
```

In the above example, source MAC address is randomly generated, while destination MAC address is set as a layer two broadcast address. Ethernet type is fixed to 0xabcd and action is set to the controller, i.e., matching packet will move to the controller. Each time a port is declared as non-eligible, SLDP removes it from eligible port list. Algorithm 1 gives the abstract idea of maintaining the eligible port list. The updateEports procedure requires three arguments. One for eligible port list, i.e., *ePorts*, and other two for the operation selected, i.e., *opCode* and *portId* for a targeted port to addition or removal. Possible operations are addition (ADD) and removal (REMOVE).

Algorithm 1 Update eligible port list.

Require: ePorts, opCode, portId

```
1: procedure UPDATEEPORTS
2:     if opCode = ADD then
3:         ePorts = ePorts + portId
4:     else if opCode = REMOVE then
5:         ePorts = ePorts - portId
6:     else
7:         return
8:     end if
9: end procedure
```

If a port is not receiving an SLDP packet for a long time or a packet is not reached back to the controller which is designated for a port, then the port is removed from the eligible port list. Algorithm 2 illustrates that such ports are removed after *tflow* time period.

Algorithm 2 Calculate port eligibility.

Require: ePorts, port

```
1: procedure PORTELIGIBLE
2:     On each tflow seconds
3:     portId, portTstamp ← EXTRACT(port)
4:     if portTstamp + teport ≤ now then
5:         UPDATEEPORTS(ePorts, REMOVE, portId)
6:     else
7:         return
8:     end if
9: end procedure
```

To detect Poison and Replay attack, Algorithm 3 is used. First part of the algorithm suggests the case where an attacker is sitting on a non-OpenFlow switch and sends fake SLDP packet to switch. Because of packet broadcast destination address, this packet travels for two directions to the controller. At the controller, if a packet (i.e., *sldpPkt*) is already available to SLDP packet set (i.e. *sldpPktSet*), then an alarm is generated. In later part of the algorithm if a unidirectional link (i.e., *ulink*) is not accompanying with the reverse direction within the report time, then the destination for that link is suspected, hence it will be removed from the eligible list (i.e., *ePorts*).

Algorithm 3 Detect Poison and Replay attacks.

Require: ePorts, ulink, linkSet, sldpPkt, sldpPktSet

```
1: procedure PRDETECT
2:     On each received sldpPkt
3:     if sldpPkt ∈ sldpPktSet then
4:         dpId, portId ← EXTRACT(sldpPkt)
5:         GENERATEALERT(dpId, portId)
6:     end if
7:
8:     On each teport seconds
9:     if REV(ulink) ∉ linkSet then
10:        srcPort, dstPort ← EXTRACT(ulink)
11:        portId ← EXTRACT(dstPort)
12:        UPDATEEPORTS(ePorts, REMOVE, portId)
13:    end if
14: end procedure
```

Algorithm 4 Detect Flooding attacks.

Require: ePorts, sldpPkt, maxPorts

```
1: procedure FLOODDETECT
2:     On each received sldpPkt
3:     dpId, portId ← EXTRACT(sldpPkt)
4:     if COUNTFOR(dpId, portId) > maxPorts then
5:         UPDATEEPORTS(ePorts, REMOVE, portId)
6:     end if
7: end procedure
```

Algorithm 4 is used to detect the flooding attack. If at any switch port (i.e., portId), the number of SLDP packets are received more than the maximum number of ports on available switch (i.e., maxPort), it generates suspicion. SLDP generates packet periodically, one for each port on each switch (not always), hence if any port receiving more than *maxPort* is eligible for removal.

In SLDP, the following are the periodic tasks: flow entry installation, packet generation, sent, reception, and link discovery. The packet generation is restricted with flow entry installation. Lets suppose T_f is a time interval for flow entry installation. After T_f, a new set of flow entries are installed, thus new randomness is inserted in flow entries. Let $l_1, l_2, l_3.l_n$ are the latencies or round-trip-time in delivering the SLDP packets back to the controller, i.e., the time between the SLDP packet generation by the controller and the same SLDP packet received by the controller. Here, l_i is the latency for i^{th} link discovery in topology, therefore, $T_f > Max\{l_1, l_2, l_3.l_n\}$ will ensure that the SLDP packets are delivered back to controller on time. If not, i.e., few SLDP packets not route back to controller due to latency in the network, then the controller falsely calculates that few links does not exist in the constructed global topology. If T_f is kept sufficiently large, then the SLDP will become more vulnerable to described attacks. It is because the attacker gets more time to craft a packet with the desired randomness. Test case #1 in Section III-E gives a probability analysis of being attacked while T_f is five seconds. Advancing T_f will increase the probability of being

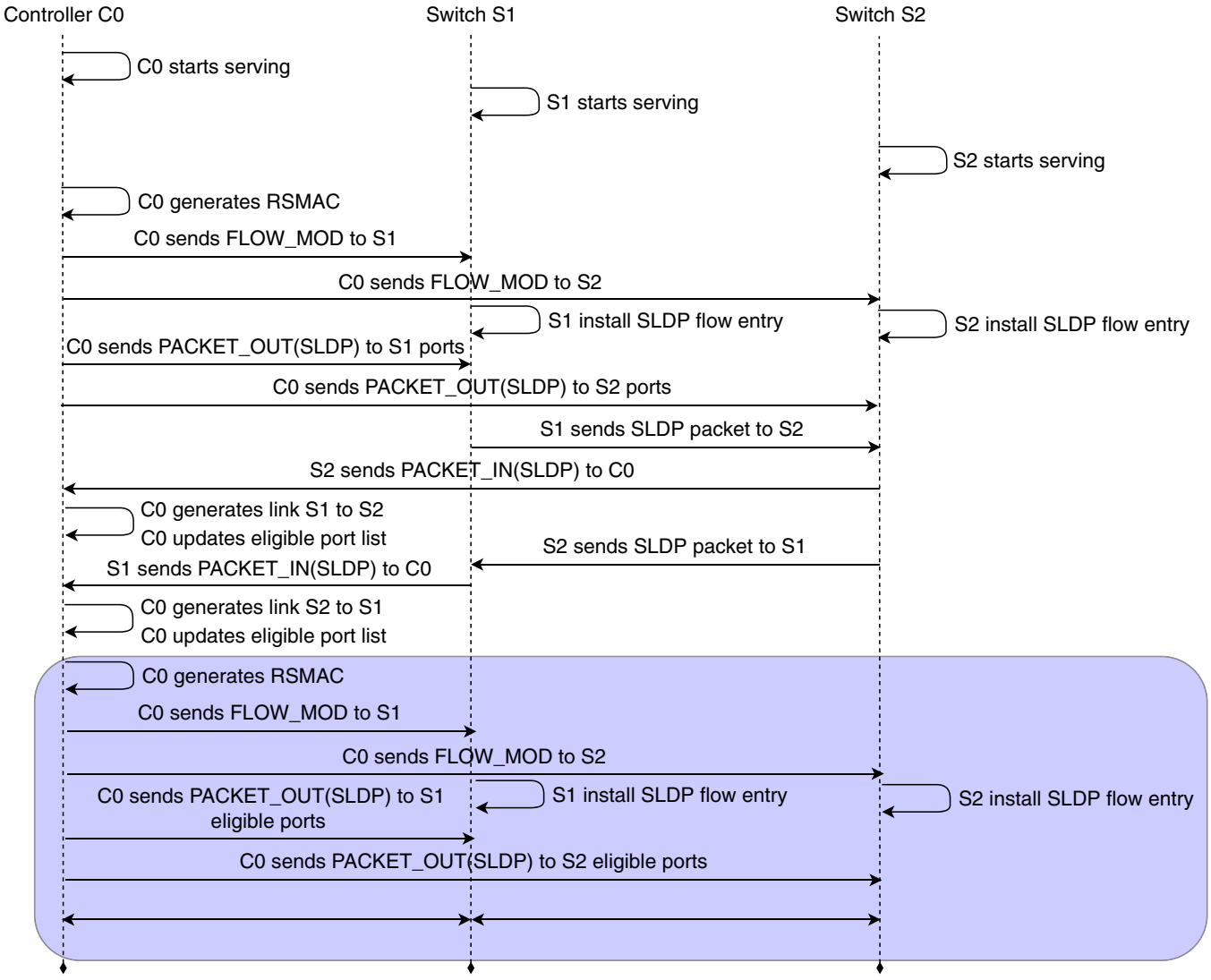

Fig. 7. Event sequence in SLDP.

attacked. Also, Fig. 18 suggests that the computational overhead for link discovery in SLDP is almost constant, i.e., $\approx$ 1 second, irrespective of the examined topologies. We chooses the time interval (T_f) to 5 seconds, which is based on the available literature [4,15], different implementations, and observations. Our observations indicate that for any communication, the controller processes symmetric and asymmetric massages, which are very low as compared to the entire packets. Therefore, there was not identified any significant difference in the controller overhead with the varying time interval.

Event sequence in SLDP: Fig. 7 illustrates events sequence to understand SLDP in more detail. Here, three vertical time lines are shown for each participating entity, i.e., the controller *C*0 and two switches *S*1 and *S*2. Initially, the controller *C*0 starts serving after the switch *S*1 starts serving. *C*0 updates the eligible port list with information it received from *S*1 i.e., ports. Later switch *S*2 starts and *C*0 updates the eligible port list again. Hence, each participating entity is ready to participate. *C*0 generates random source MAC address, which is used in flow entries and SLDP packets. *C*0 sends flow entry in OpenFlow message FLOW_MOD to *S*1 and *S*2 with generated randomness. After both *S*1 and *S*2 receives FLOW_MOD from *C*0, both installs SLDP flow entry. Only after installation of flow entries, *C*0 generates and sends SLDP packet in PACKET_OUT for the eligible port list for *S*1 and *S*2. *S*1 receives SLDP frame in PACKET_OUT from *C*0 and unwraps SLDP frame from the received packet. *S*1 sends SLDP frame to the designated port. *S*2 receives SLDP frame and use installed SLDP flow entry to generate PACKET_IN. The controller *C*0 receives PACKET_IN from *S*2 and extracts eventDpId and eventPort information from PACKET_IN header. The dpId and portId information is stored in SLDP packet. *C*0 creates a link with source and destination information. Here source information is content of SLDP packet, i.e., dpId and portId. Destination information is eventDpId and eventPort information. Later *C*0 updates the eligible port list and makes it ready for next iterations. The same process is repeated when *S*2 sends SLDP frame to *S*1. Link discovery is a periodic process, i.e., entire process is repeated after a fixed time interval. The highlighted area in the event sequence diagram shows the repetitively executed instructions.

3.5. Test case analysis

In this section, we show the assurance of the correctness of SLDP with few test cases. SLDP promises to provide lightweight, efficient, and secure link discovery protocol. SLDP works with less number of bits in a packet and less number of packets for dis-

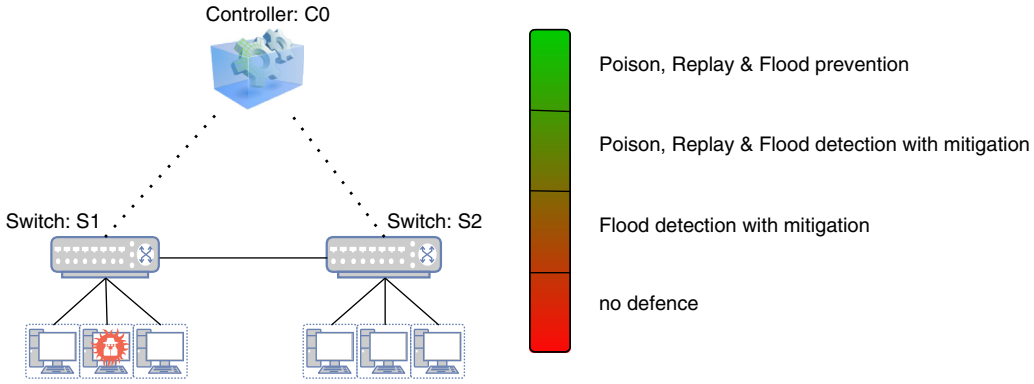

Fig. 8. Attacker host starts work after switch

covery process. The protocol provides a probable security at three different levels, which are as follows.

```
A: Poison, Replay, and Flooding prevention
B: Poison, Replay, and Flooding detection and mitigation
C: Flooding attacks detection and mitigation
```

As the name suggests, best defense strategy is poison, replay, and flood prevention. A controller creates an environment in which no such attack will happen. Second best line of defense is poison, replay, flood detection and mitigation. While detection is performed, still the system resources, i.e., CPU and bandwidth are wasted. To fix further, attack mitigation helps. SLDP removes few ports from the eligible port list to prevent further attacks. The last line of defense is flooding detection and mitigation. In some cases, SLDP can prevent all, while in others, SLDP detect and mitigate the attacks. We ensure and prove that at least SLDP detects flooding and mitigates for future incidents.

Now we explain the SLDP working by using few test case scenarios. In each test case, a controller is attached with two or three switches. Switch to host are attached as shown in Figs. 8 to 12. Some of the host are infected with the malicious application or controlled with a person with malicious intention. A sidebar in each test case figures is showing the strength of the security defense in that test case.

Test case # 1 In this test case, an attacker host exists which is attached to an OF-switch as shown it is shown in Fig. 8. If the malicious host starts serving after the switches, and try to perform replay, poison, and flooding attack. The SLDP prevents the network from all the stated attacks. In case of SLDP, no SLDP packet comes to any of attached hosts, hence protects against the reply attack. Poison on another hand uses crafted packet, but SLDP working suggests a packet can only moves to the controller if it has exact randomness as flow entry does. If the host is unable to pass one packet to the controller then performing the flooding is too hard.

Attacker generated packets for replay or poisoning will only reach to the controller if the packets have the same randomness as the flow entries. In each cycle, SLDP generates a new random number, which is used in both flow entry and link discovery packets. For instance, random MAC address is denoted as a set $M = \{m_1, m_2, m_3 \ldots m_n\}$, and the packet generated by an attacker is denoted by a set $A = \{a_1, a_2, a_3 \ldots a_x\}$. Then the desired condition for successful prevention is $\frac{|A|}{|M|} = 0$. Let's examine the chances that a fake packet have the same randomness.

MAC address is 48 bits long, hence $|M| = 2^{48} \approx 2.81^{*}10^{14}$. Size of SLDP packet is 26 bytes. Total number of such packets on 10 GBps in one second is equal to $(10^{*}10^{9})/26 \approx 3.85^{*}10^{8}$. If the flow entry remains constant for five seconds, then in the same time period the number of total packets are $|A| \approx 1.92^{*}10^{9}$. Now $\frac{|A|}{|M|} = 0.00000683214 \approx 0$ confirms that even in theory the full speed packets are generated, which makes the probability of a successful attack nearly to zero. If one fake packet is hard to reach the controller, then the flooding attack is hard to believe.

Test case # 2 As shown in Fig. 9, an attacker attached to an OF-switch starts working before the switch starts. In this case, because the SLDP chose the entire port list as an eligible port list in the beginning, the malicious host will also receive the SLDP packet. As the host has the packet or the randomness stored in a packet, it can perform attacks. Even though attacker successfully makes a unidirectional link, it will still be unable to make a reverse link. The attacker only sits on one end of the fake link; to make it in reverse direction, attacker's control is required on other side of the fake link. The above information helps the SLDP to detect such attacks. To mitigate attacks, the SLDP removes packet receiving switch port from the eligible port list.

Test case # 3

The Fig. 10 shows two attacker hosts attached to different OF-switches. Now both the switch receives SLDP packets, extracts the randomness and craft the fake SLDP packets with the same randomness. Both hosts can create the fake bidirectional link at the controller (shown with dash lines). In this case, the detection of replay and poison is not possible, however the SLDP will be able to detect and mitigate the flood attacks.

Test case # 4 As shown in Fig. 11, a non-OpenFlow switch separates a link between two OF-switches. An attacker or host can be attached to an OF-switch, but it will not receive any randomness information to perform an attack. Hence, SLDP prevents poison, replay, and flood attacks.

Test case # 5 Fig. 12 represents an attacker host attached to a non-OpenFlow enabled switch. The attacker always gets randomness information due to the broadcast destination MAC address. If the attacker craft an SLDP with the spoofed randomness, the SLDP packet reaches via both $S1$ and $S2$ switches. Hence, SLDP detects poison, replay and flood attacks. However, because a non-OpenFlow enabled switch can not be controlled by the controller, mitigation is not possible.

4. Simulation results and discussions

To prove any given proposal's correctness, theoretical and practical analysis is necessary. In this section, we focus on the experimental evidence for correctness of SLDP. Typologies and experimental environment are discussed to justify the experiments. All the experiments are performed on Mininet [19] network emulator. To perform experiments three different topologies are used as shown in Figs. 13, 14 and 15. Table 4 is shows the relative statis-

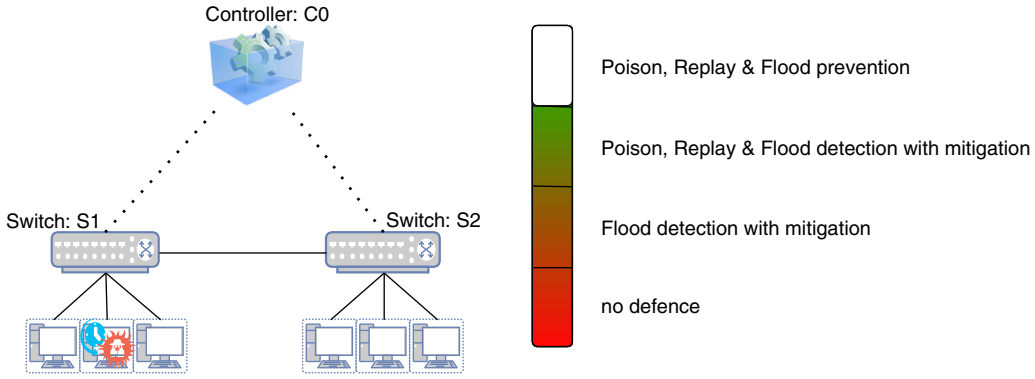

Fig. 9. Attacker host starts work before switch

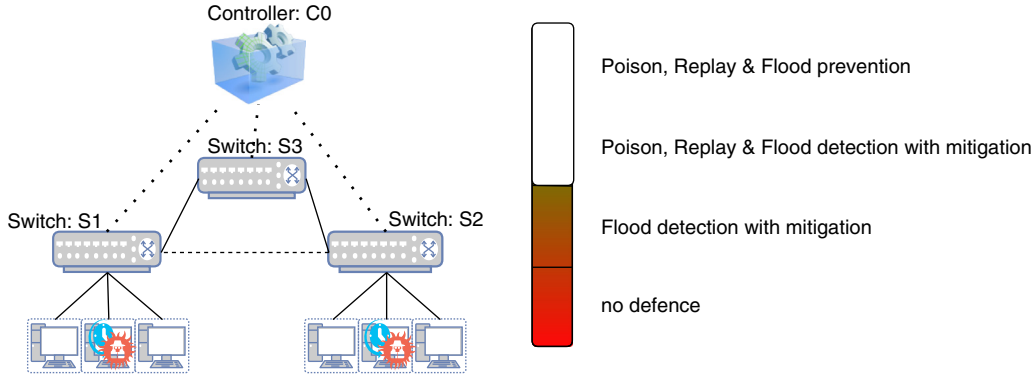

Fig. 10. Two attacker hosts starts work before switch

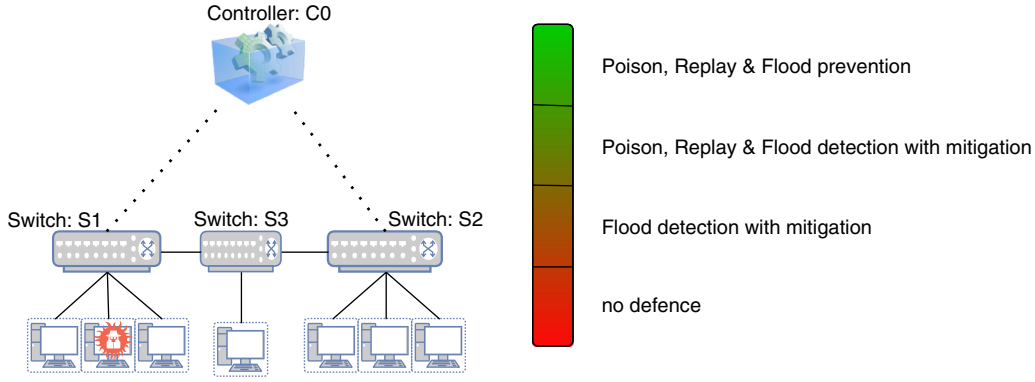

Fig. 11. A non-OpenFlow switch separates link of two OpenFlow switches.

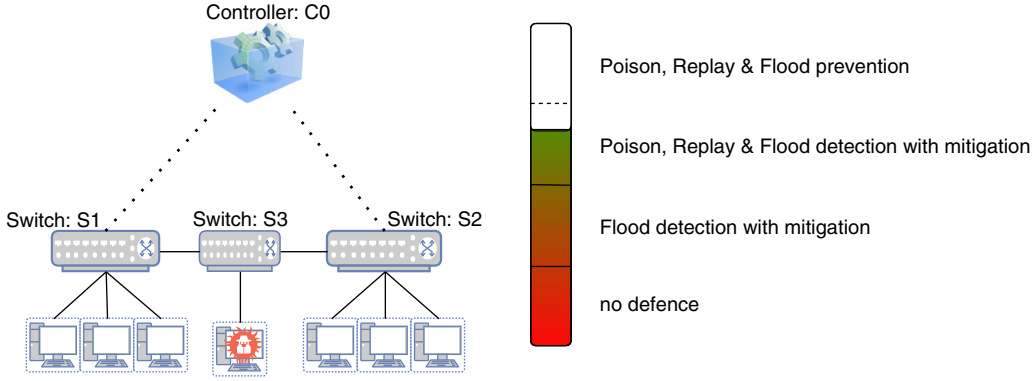

Fig. 12. Attacker host starts work before switch.

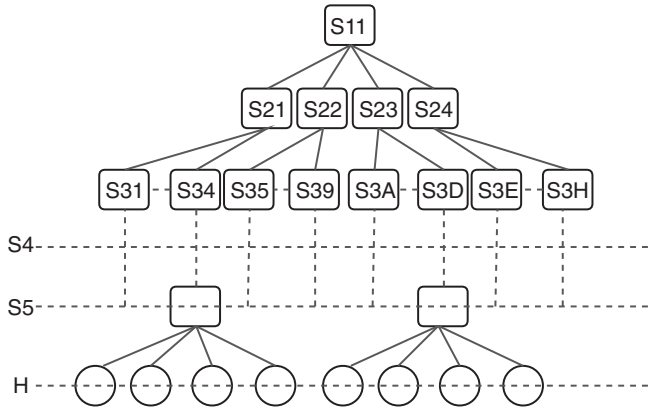

Fig. 13. Topology 1: 'tree,4,4'.

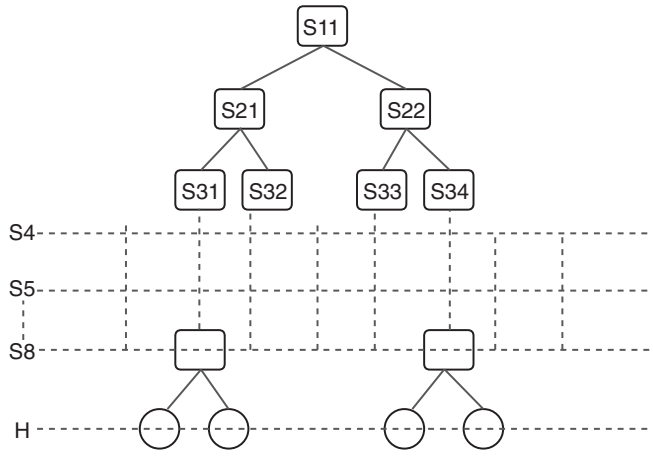

Fig. 14. Topology 2: 'tree,7,2'.

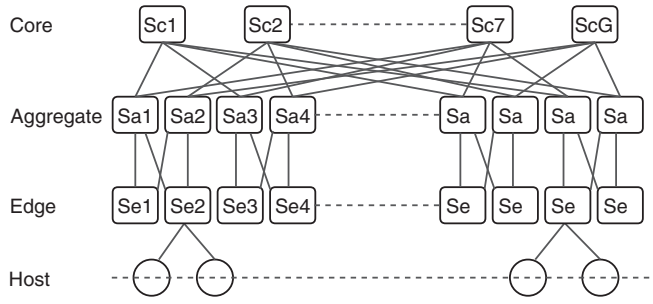

Fig. 15. Topology 3: 'fat tree' .

Table 6
Number of Switches,Links,Ports and Hosts.

Topology	Switch	Link	Port	Host
tree,4,4	85	340	424	256
tree,7,2	127	254	380	128
fat tree	80	384	705	64

tics, i.e. number of switches, links, hosts and ports. We consider the topologies that has 80, 85 and 127 switches, which is fairly large to justify our experiments. In the Table 6, the links in section 4 are between switch to switch and switch to host. In link discovery process only switch to switch links are considered, which can be obtained from the number of hosts subtracted from the number of links in the table. 'tree,4,4' and 'tree,7,2' are topologies from Mininet environment. 'fat tree' is tree topologies to simulate a data center arrangement of switches.

Table 7
Experimental environment for SLDP.

Resource	Configuration
Test-bed	Mininet(emulation)
Victim/Attacker OS	UBUNTU 16.04*LTS*(64*bit*)*
Victim configuration	4CPUs and 4 GB
Attacker configuration	4CPUs and 4 GB
Controller	RYU
Attack traffic	150,000 Packet/Sec
Software switch	OpenVSwitch(2.5.0)
Network	1 Gbps

In Fig. 13, five layers of switches are used. Except for root and leaf switches, each switch is connected to five switches. For example, $S35 - S39$ represents four connected switches in layer three. $S4$ and $S5$ are switches in layer four and five respectively. Hosts connected to layer five switches are represented with symbol H.

In Fig. 14, switches are connected with three switches except for root and leaf switches. The eight layers of switches(total 127 switches and 254 links) forms a tree topology.

As in Fig. 15, the fat tree topology is shown and alike topologies are used typically in a data center. Here the switches are arranged in a three-layer arrangement, namely core, aggregate, and edge. Core switches are connected to alternate aggregate switches for redundancy. The aggregate switches are also connected to more than one edge switches. In our experiments, we choose two hosts per edge switch.

Table 7 gives the details of an execution environment for validation of SLDP, which includes operating system and test-bed details. The SLDP implementation is tested in the Mininet [19].

Generating results in own established environment is always questionable. In this article, author hooks 'print' statement over several desired code locations to generate the results. Even this 'print' statement is performing I/O causes CPU resources. The results can be viewed in relative term rather than absolute. For comparison, other implementations are considered which varies from two implementations of the same controller, i.e., RYU-OFDP, RYU-SLDP or various controllers, e.g., POX, RYU, and ONOS. RYU-OFDP is OFDP implementation in RYU, while RYU-SLDP is SLDP implementation in RYU controller.

The controller selection in any experimental setup is dependent on aspects such as programming language, documentation, and controller updations. Most controllers are written in either JAVA or python. For instance, the POX and RYU controllers are written in python while OpenDayLight, Floodlight, ONOS, and HPE-VAN are written in JAVA. Few controllers are developed for academic purpose, i.e., POX, Floodlight, and RYU, while others are industry grade controllers, i.e., OpenDayLight, ONOS, HPEVAN. For our experimental setup we use RYU because it is being developed as open source using python and it is a well-documented controller.

SLDP is a lightweight link discovery protocol. The number of bits required in SLDP is least among other implementation. As Fig. 16 shows, the SLDP requires 26 byte long Ethernet frame to accomplish link discovery along with efficiency and security. SLDP remove some unnecessary fields and restructure the packet to reduce its size. SLDP uses position based data separation to save some additional space. In each link discovery, few discovery packets are generated and sent over OpenFlow channel to switch. The traditional implementation of link discovery generates LLDP/BDDP packets for each port on each switch. Edge switches are connected to host, which infers no need to generate packet reaching to end host. The SLDP approach also helps to prevent the controller fingerprinting. In SLDP, few non-eligible ports are identified and SLDP packet is sent to each port except ineligibles. Fewer generation of SLDP packet also consumes lesser CPU and bandwidth resources.

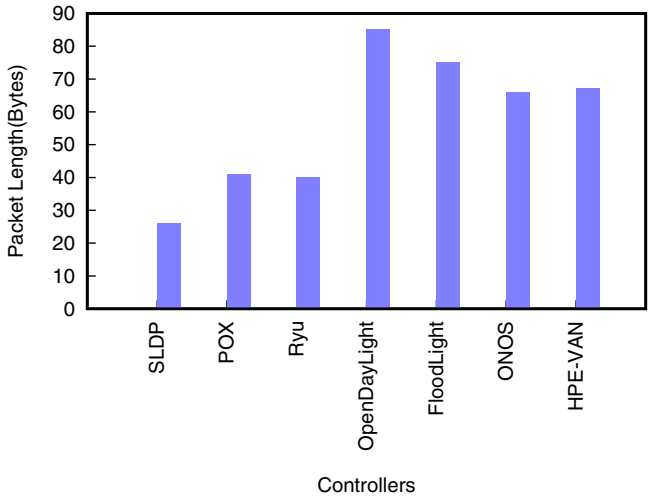

Fig. 16. Link discovery packet length among all controllers.

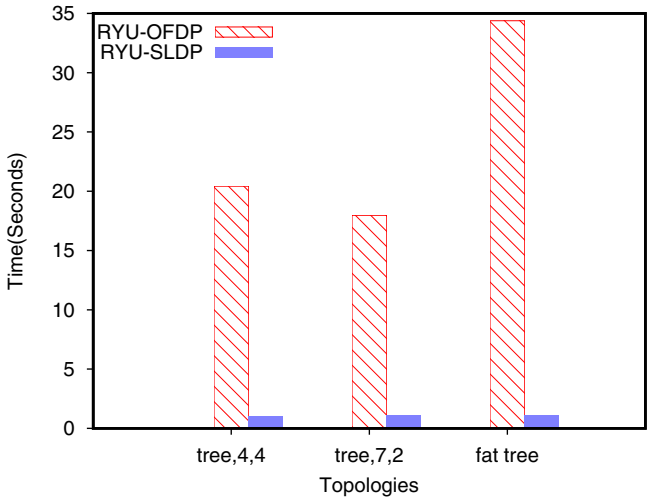

Fig. 18. Computation overhead for link discovery.

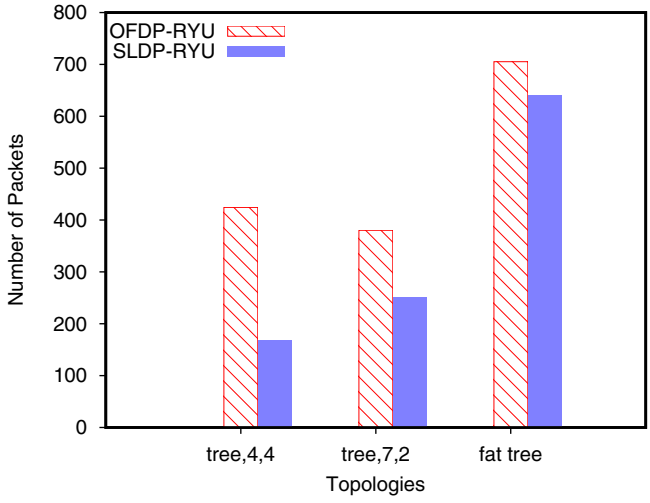

Fig. 17. Number of packets in link discovery with three topologies.

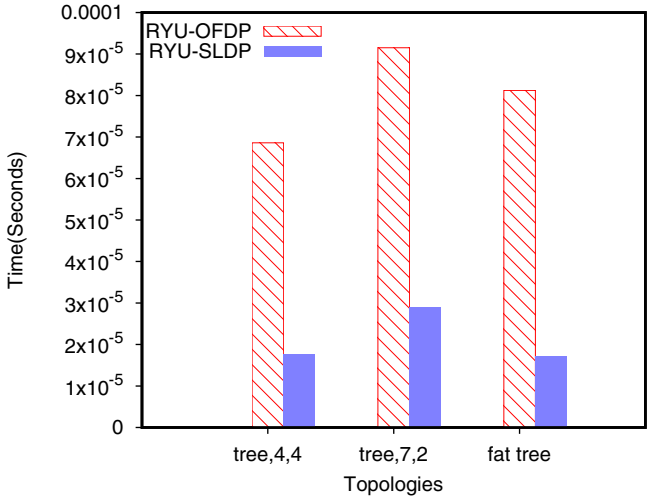

Fig. 19. Link discovery packet construction overhead.

Fig. 17 gives the idea of number of discovery packet required with or without the solution in different topologies. RYU-SLDP always require lesser packet irrespective of topology.

CPU resources are required for sending and parsing link discovery packets. Fig. 18 shows resource consumption in RYU-OFDP and RYU-SLDP with three topologies. It is clearly identified that SLDP is doing so well because of its lightweight and an efficient number of the packet. As shown in Table 4, the number of ports in topology tree,7,2 is lower than other topologies (i.e., tree,4,4 and fat tree). In OFDP number of ports is equal to the number of LLDP packets for any cycle. Hence, in the tree,7,2 topology, less number of packets are being sent and parsed during the link discovery. The same is reflected in Fig. 18, which shows the lowest overhead for OFDP.

Fig. 19 demonstrates, the time taken in link discovery packet construction. As lighter packet takes lesser time to be constructed. In this figure, there is a clear difference between RYU-SLDP and RYU-OFDP. RYU-SLDP is taking lesser time to construct.

Upon receiving any link discovery packet, it is verified and parsed. Fig. 20 shows that RYU-SLDP is parsed in smaller time than RYU-OFDP. SLDP use lighter and simple packet which is parsed on a less complex algorithm.

Most of the link discovery implementations in SDN are creating static LLDP or BDDP packets. Later these packets are sent periodically. This kind of propagation has a vulnerability that can be exploited by adversaries [22]. SLDP uses a new packet for each iteration for a fixed port. Fig. 21 shows event sequence for discovery packet construction and verification. RYU-OFDP create packet once and later use them, while RYU-SLDP generates and verifies SLDP packet alternatively.

Topology discovery time is a time taken by the controller to build entire topology. It is measured as the time taken from first LLDP or SLDP packet generation to complete topology discovery. Fig. 22 illustrates RYU-SLDP is performing much better then RYU-OFDP irrespective of the topology. The reason behind it is small packet size and the efficient way of processing them.

One of the key to success for SLDP is the calculation of eligible ports. SLDP only sends SLDP packet to eligible ports. Initially, all ports are eligible ports but later some ports are declared as non-eligible. Fig. 23 demonstrates eligible ports over time for SLDP with three topologies.

SLDP packets are generated and sent periodically, it also include installation of flow entry, which allows SLDP packet to reach back to the controller. In traditional implementation of link discovery, a unique flow entry is installed once and it remains forever. It is because link discovery packets are created once in OFDP, an thus the strategy works. With the SLDP, it is not the case, and it is considered as initial overhead for RYU-OFDP only. Fig. 24 shows the initial overhead, which is the sum of the initially generated LLDP packets and flow entry installation which is done once for RYU-OFDP implementations. However, in RYU-SLDP, the same is a pe-

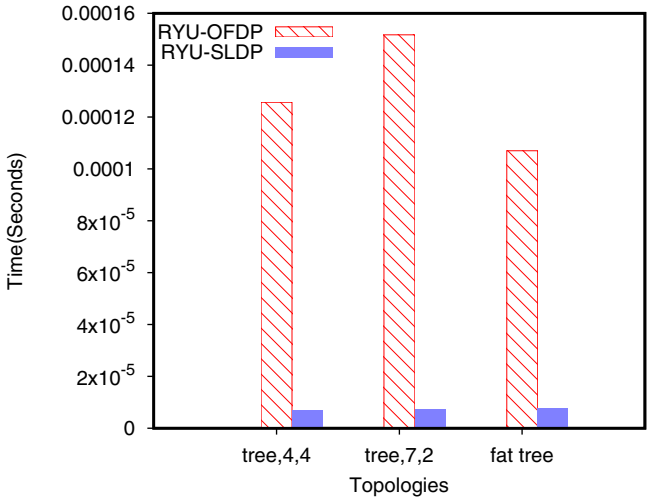

Fig. 20. Link discovery packet verification overhead.

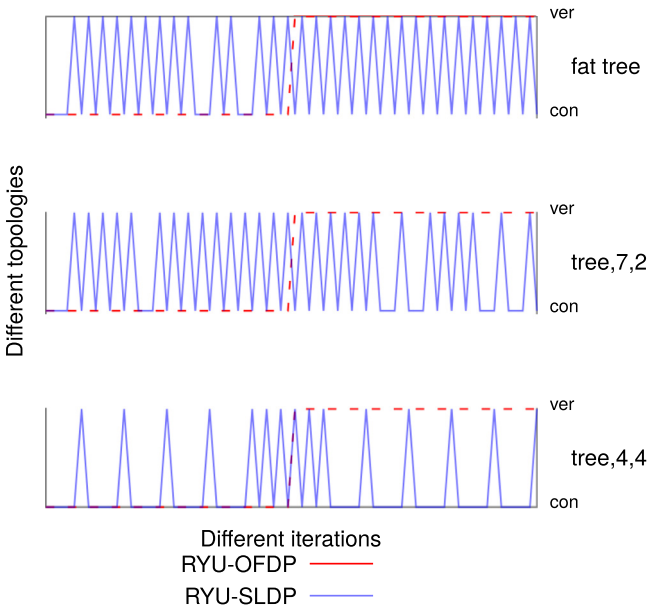

Fig. 21. Link discovery packet construction and verification sequence.

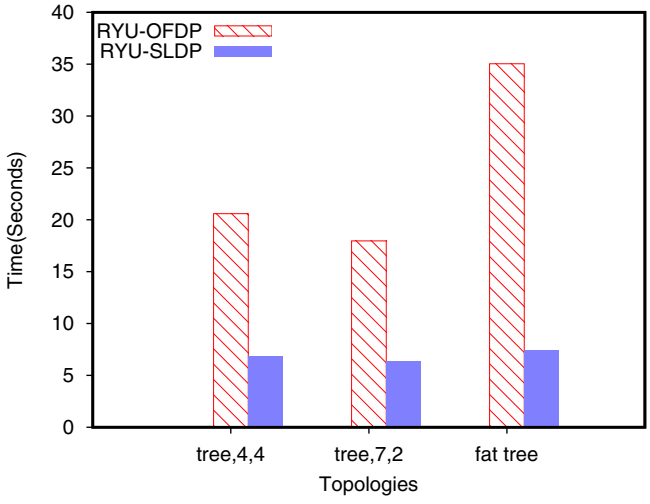

Fig. 22. Topology discovery time.

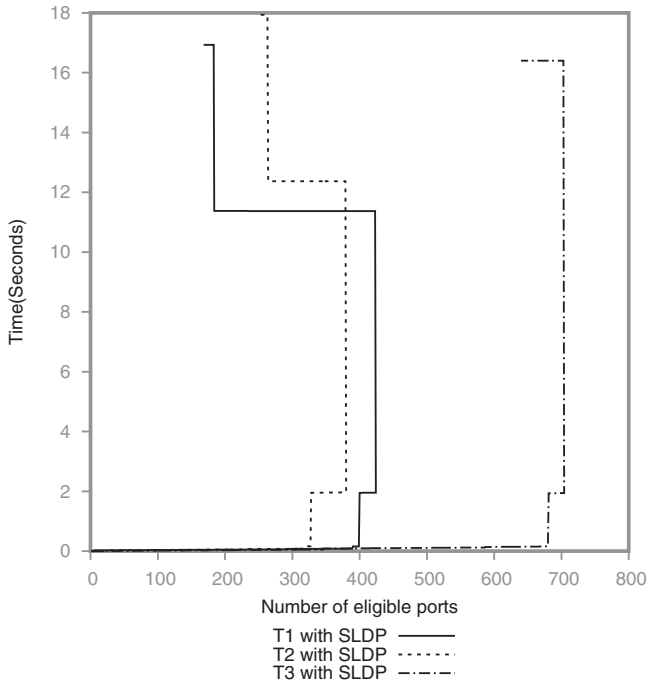

Fig. 23. Eligible ports over the time.

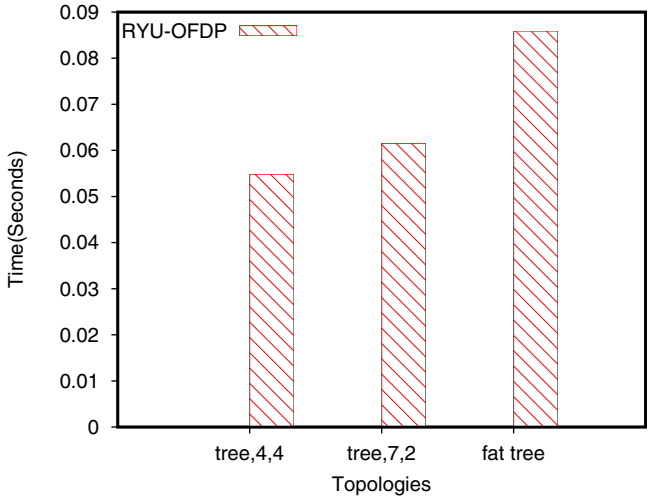

Fig. 24. Initial overhead in RYU-OFDP.

riodic task. Hence, the initial overburden is limited to RYU-OFDP implementations only. Additionally, Fig. 18 shows the overhead in link discovery for both RYU-OFDP and RYU-SLDP. RYU-OFDP have initial overhead apart from overhead to link discovery.

SLDP demonstrate link detection ability with lighter weight packet. Result demonstrates that the low number of discovery packets are generated, which results a lower time for SLDP packet construction and verification.

5. Conclusion and future work

Link discovery in SDN is crucial for topology-aware applications. In this paper, first we present the motivations i.e. security, lightweight, and efficient, for the design of a new link discovery protocol in SDN. Poison, replay, and flooding attacks are the major targeted threats. Our proposal i.e., SLDP, which provide a design of discovery packets which are lightweight. Additionally, SLDP architecture is discussed and event sequence is also described, SLDP

uses token based prevention with the lightweight and lower number of packet to provide security during link discovery. Latency for each link is also calculated using different time stamps. In Mininet environment, proposed protocol is designed and analyzed for correctness and effectiveness. Resource penalty for overall computation is far less than the original implementation. Link discovery packet construction and verification time are also low. We are also planning to release the assumption of a bug-free switch and controller. Distributed link discovery is challenging, and it can also be investigated in future as well.

Acknowledgement

The work is partially supported by DEiTY Government of India project, ISEA-II in the Department of Computer Science and Engineering at Malaviya National Institute of Technology, Jaipur.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.comnet.2018.12.014.

References

[1] D. Kreutz, et al., Software-defined networking: a comprehensive survey, Proc. IEEE 103 (1) (2015) 14–76.
[2] B.A.A. Nunes, et al., A survey of software-defined networking: past, present, and future of programmable networks, IEEE Commun. Surveys Tutorials 16 (3) (2014) 1617–1634.
[3] L. Ochoa Aday, et al., Current trends of topology discovery in openflow-based software defined networks, Int. J. Distrib. Sens. Netw. 5 (2) (2015) 1–6.
[4] S. Khan, et al., Topology discovery in software defined networks: threats, taxonomy, and state-of-the-art, IEEE Commun. Surv. Tutorials 19 (1) (2017) 303–324.
[5] PoxURL: https://github.com/noxrepo/pox.
[6] Ryu, URL: https://osrg.github.io/ryu/.
[7] Onos, URL: http://onosproject.org/.
[8] Opendaylight, URL: https://www.opendaylight.org/.
[9] Floodlight, URL: http://www.projectfloodlight.org
[10] Hpevan, URL: https://marketplace.saas.hpe.com/sdn/content/sdn-controller-free-trial.
[11] F. Pakzad, et al., Efficient topology discovery in openflow-based software defined networks, Comput. Commun. 77 (2016) 52–61.
[12] M. Dhawan, et al., Sphinx: Detecting security attacks in software-defined networks., NDSS, The Internet Society, 2015.
[13] S. Hong, et al., Poisoning network visibility in software-Defined networks: new attacks and countermeasures, Proceedings 2015 Network and Distributed System Security Symposium (February) (2015) 8–11.
[14] T. Alharbi, M. Portmann, F. Pakzad, The (in)security of topology discovery in software defined networks, in: 2015 IEEE 40th Conference on Local Computer Networks (LCN), 2015, pp. 502–505.
[15] Z. Xin, Y. Lin, W. Guowei, Esld: an efficient and secure link discovery scheme for software defined networking, Int. J. Commun. Syst. 31(10) e3552.
[16] L. Ochoa-Aday, C. Cervello-Pastor, A. Fernandez-Fernandez, Discovering the network topology: an efficient approach for SDN, ADCAIJ: Adv. Distrib. Comput. Artif. Intell. J. 5 (2) (2016).
[17] Y. Jimenez, C. Cervello-Pastor, A. Garcia, Dynamic resource discovery protocol for software defined networks, IEEE Commun. Lett. 19 (5) (2015) 743–746.
[18] L. Ochoa-Aday, C. Cervello-Pastor, A. Fernandez-Fernandez, Self-healing topology discovery protocol for software-defined networks, IEEE Commun. Lett. 22 (5) (2018) 1070–1073.
[19] B. Lantz, B. Heller, N. McKeown, A network in a laptop: Rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, in: Hotnets-IX, ACM, New York, NY, USA, 2010, pp. 19:1–19:6.
[20] G. Tarnaras, E. Haleplidis, S. Denazis, SDN And forces based optimal network topology discovery, Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft) (2015) 1–6.
[21] Link layer discovery protocol and mib, URL: http://www.ieee802.org/1/files/public/docs2002/lldp-protocol-00.pdf.
[22] A. Nehra, M. Tripathi, M.S. Gaur, 'global view' in sdn: Existing implementation, vulnerabilities & threats, in: Proceedings of the 10th International Conference on Security of Information and Networks, in: SIN '17, ACM, New York, NY, USA, 2017, pp. 303–306.
[23] T.H. Nguyen, M. Yoo, Analysis of link discovery service attacks in SDN controller, International Conference on Information Networking (2017) 259–261.
[24] Openflow switch specification, URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-specv1.3.0.pdf.

Ajay Nehra is a Doctoral research scholar since January 2015 in the Department of Computer Science & Engineering at Malaviya National Institute of Technology Jaipur, India. He obtained his M. Tech. Degree in Computer Science & Engineering from the Central University of Rajasthan, India in 2012. He earned Bachelor in Engineering in Computer Engineering from the University of Rajasthan in 2008. He has been awarded scholarships for both masters and the doctoral degree from Ministry of Human Resource Development, Government of India. His current research area includes Software defined networks, Information security and Network security.

Meenakshi Tripathi is an Associate Professor at Malaviya National Institute of Technology (MNIT), Jaipur (India). She obtained her Ph.D. from MNIT in 2015. Her research areas are Wireless Sensor Networks (WSN), Software Defined Networks (SDN) & Internet of Things (IoT). She has published more than 30 research papers in refereed International & National Journals & proceedings of National & International conferences. She has presented her research ideas by participating in various conferences within India & outside India like China, Canada, USA etc. She is working on various research projects funded by the Department of Science and Technology, Ministry of Electronics and Information Technology, India of worth around Rs. 50 Lakhs. She is the reviewer for several renowned journals such as International Journal of Communication System (Wiley), Security and Communication Networks (Hindawi), Wireless Communications, etc. She has also been part of several International conferences such as SPACE 2015, ICISS 2016, ICSP 2019 etc. She is an upright member of institute of Electrical & Electronics Engineers (IEEE) & Association of Computing Machinery (ACM). She is also the Chairman of Computer Society of India (Jaipur Chapter).

Prof. Manoj Singh Gaur completed his Master's degree in ComputerScience and Engineering from IndianInstitute of Science Bangalore, Indiaand Ph.D. from University of South-ampton, UK. Prof. Gaur has been afaculty in Department of Computer Science and Engineering, Malaviya National Institute of Technology Jaipur, India and currently Director, IIT Jammu. His research areas include Networks-on-Chip, Computer and network security, Multimedia streaming in wireless networks.

Ramesh Babu Battula received the B.Tech. degree in information technology from Acharya Nagarjuna University, Guntur, India, the M.Tech. degree in computer science and engineering from the Indian Institute of Technology Guwahati, Guwahati, India, and the Ph.D. degree from the Malaviya National Institute of Technology (MNIT), Jaipur, India, with a focus on routing in next-generation networks, in 2016. He is currently an Assistant Professor with the Department of Computer Science and Engineering, MNIT. His current research interests include secure communication, computer security, information security, performance modeling, and next generation (xG) networks. He is a member of ACM and has authored many scientific papers in networks and security.

Chhagan Lal is Postdoc fellow in Department of Mathematics, University of Padua, Italy. He obtained his Bachelors in Computer Science and Engineering from MBM Engineering College, Jodhpur, India in 2006. He obtained his Master's degree in Information Technology with specialization in Wireless communication from Indian Institute of Information Technology, Allahabad in 2009, and PhD in Computer Science and Engineering from Malaviya National Institute of Technology, Jaipur, India in 2014. He has been awarded Canadian Commonwealth scholarship in 2012 under Canadian Commonwealth Scholarship Program to work in University of Saskatchewan in Saskatoon, Saskatchewan, Canada. His current research areas include Blockchain Analysis, Security in Wireless networks, Software-defined networking, Underwater acoustic networks, and context-based security solutions for Internet of Things (IoT) networks.