Securing Topology Discovery in Software Defined Networks: Trends, Gaps, And Future Directions

AHMAD ENESI SIYAKA[1], SALISU ALIYU[2], SAHABI YUSUF ALI[3]
[1, 2, 3] Computer Science Department, Ahmadu Bello University, Zaria-Nigeria

Abstract- Software Defined Networking (SDN) has revolutionized network architecture by separating the control plane from the data plane, offering enhanced flexibility, programmability, and centralized control. However, this paradigm shift introduces significant security concerns, particularly in the area of topology discovery, where threats such as topology poisoning, link fabrication, and host hijacking are prevalent due to the lack of standardization in SDN protocols and the dynamic nature network environments like virtual data centers and cloud infrastructures. This survey explores various security mechanisms proposed for topology discovery in SDN, with a focus on the OpenFlow protocol. It reviews key approaches designed to mitigate common vulnerabilities, including the use of authentication, encryption, and anomaly detection techniques. The survey highlights the trade-offs between security measures and network performance, analyzing their effectiveness in addressing topology-related threats while minimizing overhead. The findings suggest that while many solutions enhance SDN security, challenges such as resource consumption, latency, and packet processing overhead persist. Future research should aim to develop lightweight, scalable mechanisms that balance robust security with operational efficiency, ensuring optimal performance of SDN in dynamic, large-scale networks.

Indexed Terms- Software Defined Network, Topology Discovery, Link Layer Discovery Protocol (LLDP), OpenFlow

I. INTRODUCTION

Software-Defined Networking (SDN) is a new area in networking that is focused on making networks programable by decoupling the control plane from the data plane of the network [14]. There is the presence of a controller on top of which high-level Northbound Application Programming Interfaces (APIs) run to achieve very advanced functionality such as load balancing, traffic shaping, rule-based access control, or general traffic monitoring which were in the past, the function of middleboxes in the network. The controller typically has a northbound and a southbound interface through which it communicates with the APIs and the network elements at the data planes, respectively.

The OpenFlow communication protocol, standardized by the Open Networking Foundation (ONF) in 2011, is the de-facto open-source standard for the southbound interface [10]. The OpenFlow protocol has received significant attention from the research community [9] and is currently being used in real-world SDN networks, such as Google's B4 network [6]. The centralized controller is responsible for installing and configuring the forwarding table in the programable switches. This makes it possible to control the network behavior from a centralized location. This was very easy to accomplish as switches already implemented flow tables in their design [6], so, vendors were only required to open up interfaces so that separate software could populate these flow tables.

Switches contain several flow tables, each with its own set of flow rules. A flow rule consists of three fields: matching criteria, action (e.g., drop the packet), and priority. At the initial reception of a packet by the switch which it has not learned from the controller how to handle, it sends such packet in an encapsulated OpenFlow packet-in message to the controller which receives it and in turn installs the flow rule for handling such packet in the controller through a packet-out message. So, in the next occurrence of such packet in the switch, it acts on it based on the installed rule matching the packet header as long as the rule remains valid. The SDN controller makes use of the Link Layer Discovery Protocol (LLDP) for establishing the existence of links amongst network components. In the traditional topology discovery architecture, the LLDP packet is sent as a broadcast message at a fixed periodic interval to connected OpenFlow Enabled-Switches [11]. Because the switch broadcasts the packet to all its ports, a malicious host connected to one of the ports can get the LLDP packet as an attack vector to poison the network topology.

In the case of a relay attack, the adversary passes on this LLDP packet directly to another switch that does not have direct link to the originating switch and in turn revert the same packet to the controller which learns falsely that there is a direct link between the switches as there is no way of verifying the authenticity of the packet source [20]. In the case of a host hijacking attack, the malicious host modifies the packet to reconstruct the packet header as may be desired to trick the controller that a known host moved to another location in the network. Therefore, if there is no way of verifying that the host has actually moved, the controller updates the flow table to reflect this new host location information.

Several researchers have shown that it is possible to launch security attacks at the application, control, and data planes ([17] and [19]), while other works proposed counter measures for improving the security of SDN ([16] and [18]). Among the proposed attacks, topology attacks that aim at poisoning the network topology are one of the most dangerous types. Unlike in traditional networks where adversaries can only tamper with the topology of a small fraction of the network by convincing a set of switches/routers of a specific (fake) topology event, the consequences of such attacks in SDN networks can be more severe due to the controller's centralization and the global view it has of the network. As the controller contains all the network topology information in a centralized location, adversaries can influence any part of the network regardless of their location. Furthermore, the complication of SDN attack cases also comes from the fact the OpenFlow enabled-switches lack sufficient logic and capabilities to implement traditional countermeasures such as dynamic Address Resolution Protocol (ARP) inspection.

From what precedes, it becomes very clear that maintaining the correct and genuine network topology view at the controller is of paramount importance. The core services and applications in SDN require real-time and accurate topology information to perform their tasks correctly. The possibility of attack and compromise to the SDN topology discovery service can open the way for adversaries to gain access to the network traffic. This can mean the ability to bypass security policies in the network, carry out Man-in-The-Middle (MiTM) or Denial-of-Service (DoS) attacks. Or even hijack the identity of hosts in the network to divert their traffic for malicious operations. The host could be a whole server that handles a very large amount of traffic in the network

In TCP/IP architecture, the layering abstraction is in the vertical dimension while the plane abstraction is in the horizontal dimension with the first path being the physical infrastructure and some sequence of layers on top of that as shown in Figure 1. Each layer in the layer abstraction directly depends on the services of other layers, the plane abstraction has no such dependency. Instead, every layer has a specific function and responsibility in the network.

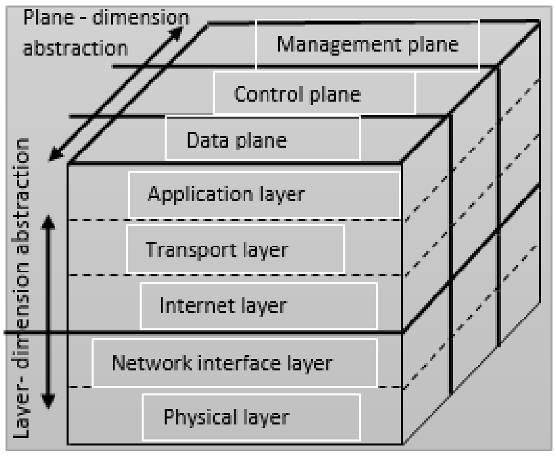

Fig. 1. Illustration of TCP/IP Layering and Plane

In the current Internet Protocol (IP), functions such as packet forwarding, flow control, access control, routing, and network management are a collection of functions performed by the same set of IP protocols. The control/management and forwarding are tightly coupled together whereas this constitutes an obstacle to the ever-demanding flexibility required by the growing Internet and Internet services. The introduction of SDN opens a new chapter in the decoupling needs of the network and allows each plane to scale vertically without being restricted by the initial tight coupling in the layered approach.

This paper contributes to the field by:

a. Providing a comprehensive analysis of recent advancements in security mechanisms for topology discovery in Software Defined Networking (SDN), with a focus on OpenFlow protocol.
b. Highlighting the benefits and limitations of existing solutions, particularly regarding their trade-offs between enhanced security and network performance.
c. Offering insights for researchers and practitioners to understand the challenges of balancing robust security with operational efficiency in dynamic, large-scale SDN environments.

The paper is structured into six sections: Section I introduces the study, Section II discusses the Layering Architecture of SDN, and Section III focuses on the SDN elements and the communication protocol between them. Section IV discusses LLDP structure and its security challenges. Section V reviews existing security mechanisms for addressing topology discovery related threats, and identifies areas for further research. Sections VI conclude the study.

II. LAYERING ARCHITECTURE OF SDN

SDN maintains a global view of the network as an abstraction to the application layer thereby hiding most of the complex maintenance and configuration functions carried out by individual network and network devices in a traditional environment. This task is now taken up by the controller. In this paper, our focus is on topology discovery services of the SDN controller and how to optimize the security in the process. Every switch on the network is managed by the controller through the process of installing appropriate forwarding rules [14]. The southbound interface is the interface between the controller and the infrastructure. To achieve control and management of the underlying infrastructure through the installation of the forwarding rules, the switches need to allow dynamic configuration of their flow table. So, switch manufacturers can easily open interfaces in switches to allow this configurability. One of such southbound interface standards that are widely adopted is OpenFlow [10]. At the top of the SDN controller is the application layer which allows high-level programs to interact with the controller for network management and configuration functions. SDN architecture is presented in Figure 2.

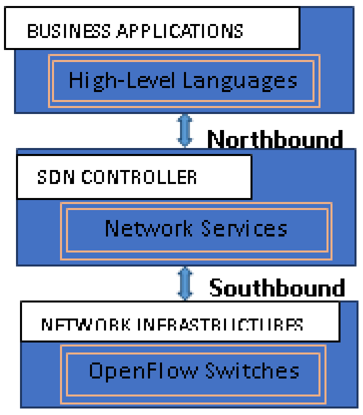

Fig. 2. SDN Architecture [1]

Through the application layer, the network programmer can define high-level network policies, high-end services, and network functions such as routing, and traffic engineering. Between the application layer and the controller is the northbound interface. Currently, the northbound specific standard is still under development [7] unlike the southbound which has already witnessed some level of standardization and adoption.

III. THE SDN ELEMENTS

The SDN architecture consists of three elements: The controller, Forwarding Elements (Programmable Switches), and the communication protocol between them [10]. The controller which is the central control unit or network operating system is responsible for gathering and communicating the network information for the programming needs of the network administrator or developer through some standard interfaces. It is also saddled with the responsibility of dynamic configuration of the forwarding elements in the network due to the global view it has gained as a result of topology learning and discovery. Because of its centralization and control ability, it eases the task of network administration which can now be achieved from a single point and programmatically. The controller can provide more advanced functionality for the network such as traffic engineering and network user abstraction. The switches and routers now act as mere forwarding elements without the complication of policy and switching rules in them. Network policies and security definitions come from the centralized controller more efficiently [18].

OpenFlow is the most widely accepted standard protocol for defining the southbound interface of SDN [10]. This interface lies between the controller and the network infrastructure and through it, the controller is able to communicate with the OpenFlow enabled switches at the infrastructure layer of the network. It allows the controller to control and manage the switches through the installation of forwarding rules in their flow tables [14]. Other existing protocols for the southbound interface such as Simple Network Management Protocol (SNMP), Border Gateway Protocol (BGP) and Path Computation Element Protocol (PCEP) [21]. The reason for OpenFlow wide acceptance is the fact that it is being promoted by the Open Network Foundation (ONF) (Open Networking Foundation, 2021) which makes it the current dominant standard. The current version of OpenFlow as at the time of this research is version 1.5 (Open Flow Standard version, 2021). Since the first released version (1.0) of the OpenFlow protocol, it has undergone some changes. However, the differences in the various version do not affect the implementation of our proposed discovery scheme as it works for all versions in the same way. An OpenFlow enabled switch supports basic match-action, this allows every incoming packet to be compared against the rules in the switch's flow table and the associated match-action is executed against it. Supported OpenFlow match fields includes the switch ingress port, various packet header fields such as source and destination MAC addresses, source and destination IP addresses, UPD/TCP port numbers.

The SDN need a way of knowing how the network infrastructure is laid out in the network to provide basic network functions such as routing, flow control, basic Match/Action operations on received packets [11], and even the current functions of some middleboxes like network load balancing, firewalling and quality of service. Therefore, there is the need for up-to-date information of Topology at the SDN Controller. Only a de facto protocol – OpenFlow Discovery Protocol (OFDP) which adopts the layer 2 Link Layer Discovery Protocol (LLDP) packet format is used for this service in most of the available controllers and most of them follow the same structure for topology discovery as derived from the original SDN controller-Networking Operating System (NOX). In the OFDP discovery approach, The SDN controller is required to generate an LLDP packet for the individual port on each switch carrying the specific Data Path Identity (DPID) and Port Identity (PortId) as applicable [11]. These generated LLDP packets are sent in a packet-out message to the designated switches with individual instruction to forward the packet to the mentioned port. In the OpenFlow Enabled-Switches, there exist a preconfigured flow rule that allows for three possible actions on the received LLDP packet - Broadcasting to all switch ports, drop the packet, or send to the controller via a packet-in message which also contains metadata with necessary parameters for link discovery like the chases number of the switch, the ingress port where the LLDP packet was received from and other details in the Type-Length-Value (TLVs) of the packet which makes the controller guess the existence of a link between the LLDP packet source and destination [11]. Figure 3 illustrates the flow of this discovery packet in a network comprising of three switches. While Figure 4 shows how an adversary can succeed in poisoning the SDN topology view making use of the static LLDP packet. The reverse link is learned in another phase of LLDP propagation and a bidirectional link is established. This discovery process is performed at fixed periodic interval typically around 5 seconds to update the network topology view at the controller [11] and should incase a link is nonexistent any more or a new link is added resulting in topology change, the controller learns this through repetition of the process.

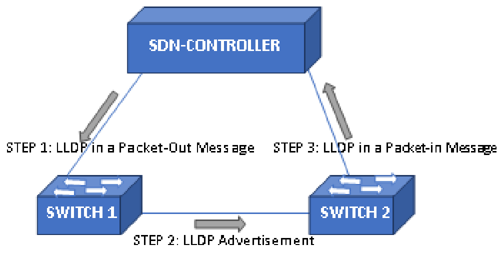
SDN-CONTROLLER
STEP 1: LLDP in a Packet-Out Message
STEP 3: LLDP in a Packet-in Message
SWITCH 1
SWITCH 2
STEP 2: LLDP Advertisement

Fig. 3. SDN Link Discovery Procedure [11]

IV. STRUCTURE OF LLDP PACKET

LLDP exchanges information through specific units of data called Link Layer Discovery Protocol Data Unit (LLDPDU). These data unit consists of TLVs and each TLV field corresponds to a certain type and length. LLDP standard IEEE 802.1AB has three TLVs that are mandatory at the beginning of an LLDPDU in the following order:

- Type 1 = Chassis ID (Identifies the device)

- Type 2 = Port ID (Identifies the port)
- Type 3 = Time to live (Tells the receiving device how long the received information should remain valid)
- Following these mandatory TLVs, an LLDPDU can include additional, optional TLVs:
- Type 4 = Port description (displays details about the port)
- Type 5 = System name (displays given name for the device)
- Type 6 = System description (displays version of the software)
- Type 7 = System capabilities (tells the primary function and capabilities of the device)
- Type 8 = Management address (shows the IP or MAC address of the device)
- At the end of an LLDPDU the following TLV is mandatory:
- Type 0 = End of LLDPDU (Signals the end of the data unit)

LLDP Replay Attach: In this type of attack, the adversary is a host sitting on one of the ports of the OpenFlow switch connected to the controller. Since the LLDP packet gets to each port on the switches [11], the attacker grabs the packet and through a compromised host on another switch either a machine or malicious software, inserts the LLDP packer directly to the second switch without modification. By default, the switch has been configured to forward such received LLDP packets directly to the controller. Therefore, the attacker can deceive the controller and create a fake link between switch 1 and switch 3 which was previously non-existent as shown in Figure 4. The attacker can then utilize this fake link for malicious activities on the network like packet sniffing and DDOS attack. Topology poisoning attacks can be in the form of link fabrication or host hijacking. Link fabrication is achieved as explained above by replay or forgery of the LLDP packet. In the host hijacking type of topology poisoning, the attacker can create a fake packet with a source address the same as that of a real host in the network. This packet is injected into the network and when the controller received the LLDP packet, identifies that the host has migrated and goes ahead to update the flow table with this new location information. All packets meant for such host are now diverted to the newly installed host location on the network.

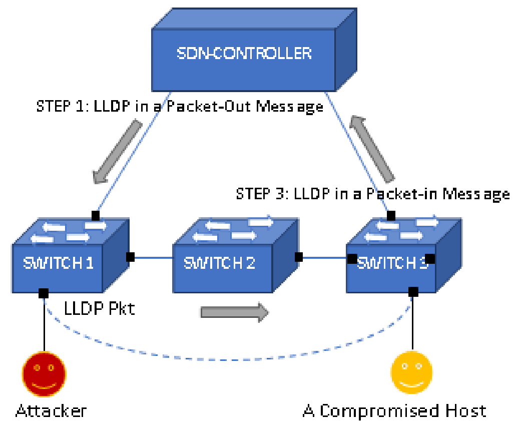

Fig. 4: LLDP Relay Attack for Topology Poisoning [11]

V. LITERATURE REVIEW

This study is based on an in-depth review of the literature, focusing on emerging trends in securing topology discovery in Software Defined Networking (SDN) environments. Articles published from 1990 to 2022 were examined, emphasizing OpenFlow-based security mechanisms and mitigation strategies against topology-related threats such as topology poisoning and link fabrication. The primary keywords used in the search included "SDN Security," "Topology Discovery," "OpenFlow Protocol," and "Network Threat Mitigation."

A total of 150 publications were identified through the literature search, of which 21 were deemed most relevant to this study. Ensuring secure topology discovery is critical to enhancing SDN's reliability in dynamic and large-scale network environments. Consequently, this study provides valuable insights into effective security strategies and highlights areas for further research and development.

Since the introduction of Software Defined Network (SDN), The research community has constantly put effort to improve its performance and adoption in the general networking world. The main protocol supported by the Open Network (ONF) is the OpenFlow protocol which adapts the topology discovery scheme from the legacy layer 2 LLDP. Concentration is therefore on how to ensure effective and efficient topology discovery in SDN. Some of such works included that of [4] where they attempt to extend the security architecture of the SDN controller by adding additional attributes (like the device type and list of hosts) that can guarantee the legitimacy of topology update requests. With such new attributes, it can show if the traffic is received from a connected host or the switch itself. It uses encryption authentication to tackle link fabrication attacks however, their work called TOPOGUARD is not effective against replay-type link fabrication attacks because they make use of a static packet that can be reused by an attacker to create a fake link in the network. And their concentration was mainly on SDN poisoning attacks without much attention to other types of attacks in the SDN topology discovery protocol. Another approach called SPHINX [2] uses static mapping of ports with a host on the network to detect poisoning attacks. It uses a flow graph to detect anomalous behavior in the data plane and based on the state of the traffic, it compares with some set of predefined or learned 'normal' behavior to alert the system of any divergence. It is more of an alert system without much concentration on the technical details of the attack. Since SPHINX is more of a policy-based system, it brings about an additional burden on the network administrator who will have to be writing and reviewing network security strategies to keep the system relevant as attacks become more sophisticated. Alharbi *et al.* (2015) proposed OFDP-HMAC which introduces the addition of extra TLVs in the LLDP packet to ensure its authentication and packet integrity with Cryptographic Message Authentication (MAC) that has a random key chosen for every round of LLDP generation. But the problem is that they still broadcast to all the switch ports. Therefore, it is also very easy for adversaries to have access to their discovery packet for packet reengineering or direct injection in the network to create fake links thereby poisoning the network topology view of the controller. Other models such as that of [15] proposed to transfer some of the topology verification tasks to the switch with additional computational power. Their model is based on Internet Engineering Task Force's (IETF) Forwarding and Control Element Separation (ForCES) framework. The main focus of their work is to gather LLDP data directly from the network devices while the controller queries for the topology information periodically.

SDN Resource Discovery Protocol (SDN-RDP) which is a distributed resource discovery protocol was proposed by [8]. In this approach, more than one controller is responsible for managing all the switches in the network, the controller needs to announce its presence in the network and then join the switches. It takes a two-phase approach, First the *forwarding phase* where the controllers announce their presence thereby learning the leaf nodes that are involved in the creation of the control channel. While the second phase is the *backward phase* that allows the individual nodes to select the preferred link direction to the controller.

[12] proposed a verification strategy that is based on switch agent. The mode of operation requires that the controller generates and sends out a multicast message to the switches in the network. When the switches receive this message, they are changed from standby node to either Father node or Active node. Each of the nodes on the network is tasked with the collection of neighbor information which is collated and asynchronously sent to the controller by a single designated Father node.

[13] suggests a layer two topology discovery that makes use of an autonomic fault recovery protocol. A topoRequest message is sent out by the controller as a multicast. While the nodes operate in four roles, (non-discovered nodes, leaf nodes, v-leaf nodes, and core nodes). Each port on the switches too can be in either of four states (standby state, parent state, child state, and pruned state). It is able to achieve automatic fault recovery with the help of some managed components and an autonomic manager. It is the task of the autonomic manager to detects the status of each port and send updated information to the managed components. SDN Link Discovery Protocol (SLDP) [11] is another very important work that was successful in reducing the number of LLDP packets generated by the controller through the introduction of what they called an 'eligible list' of ports due for LLDP reception. With this concept, the overhead of LLDP packet generation at a fixed periodic interval to all the switch ports was reduced including minimized wasteful LLDP packets as several criterial like port inactivity after some specific period can lead to the delisting of such port. They also introduced a fixed-length positional packet structure for LLDP. In addition, they used a token-based approach that generates random source MAC address for each SLDP packet. Though this work achieved a reduced number of SLDP packet generation by gradually learning and maintaining an eligible list in the controller with a well-written algorithm for port exclusion due to several factors such as when a port is not receiving SLDP packet over some defined period or there is no reversal of

the generated SLDP to the controller from a particular port, or when a port is flagged for malicious operation, the controller also checks if a successful unidirectional link is not completed as a bidirectional link within a certain period and flags such source port as malicious and delist. SLDP is proposed to be lightweight for topology discovery due to its small size and few numbers of packets required as a result of eligible list introduction in their approach. It significantly reduced the packet processing overhead at the side of the controller making the network faster in terms of topology discovery time.

SLDP packet size is limited to the minimum number of bytes necessary for SDN link discovery. In contrast to other approaches which use 40bytes to 85 bytes, SLDP uses only 26bytes doing away with those bytes that are not required for topology discovery.

It was implemented in the Mininet emulator in the architecture as shown in Figure 5. Though the number of SLDP is reduced even while there is no topology change occurring at the eligible port, they still receive the SLDP packet and this is still CPU and memory wastage coupled with the security demand and MAC creation overhead associated with this 'wasteful' SLDP packets.

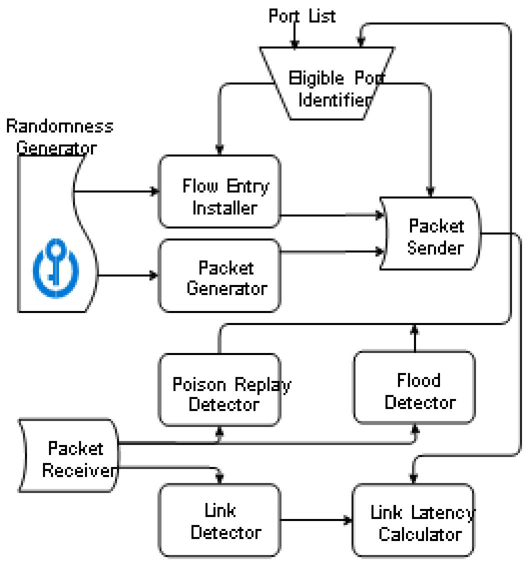

Fig. 5. SLDP System Architecture [11]

[5] proposed a topology verification scheme called TrustTopo that addresses (1) Host location verification strategy using path tracking and asynchronous rollback technique and (2) For Link verification, uses chaotic model and dynamic password generation strategy. TrustTopo leverages on the chaotic model strategy which dynamically generates the passwords. The nature of the model is such that an attacker cannot calculate the password generation space in advance, this approach helps to maintain the integrity of the information. They also make use of Fingerprint codes to ensure the unforgeability of LLDP packets, this gives two levels of trust in the LLPD generation. First, the attacker cannot predict the password in advance, and secondly, statistical learning will also not help in the forgeability of the packet security components. Unlike the simple authentication approach (i.e., the random number scheme), the TrustTopo strategy can effectively resist brute-force attacks. TrustTopo combines the chaotic model based on the random number and the random transformation rules to obtain the "double random" feature. As a result, the TrustTopo strategy achieves the dynamic and unpredictability of the passwords by taking advantage of the large differences between different conversion rules. To tackle the problem of host hijacking, TrustTopo relied on the fact that a host should not be reachable at its previous location which forms a corresponding feature event for verification. Such event is used to judge the authenticity of the host migration request. They achieve this with the path tracking and asynchronous rollback technique. TrustTopo architecture is given in Figure 6.

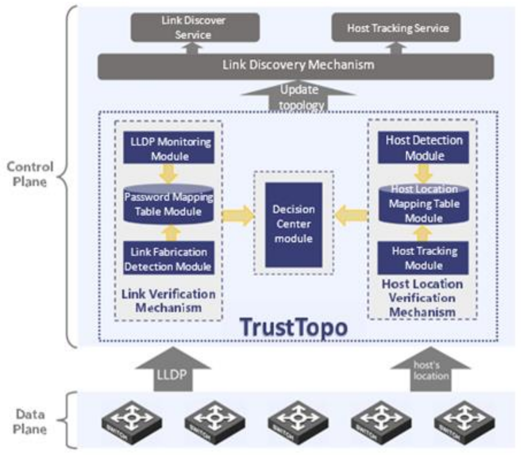

Fig. 6. TrustTopo Architecture in the SDN controller (Huang *et. al.*, 2020).

TrustTopo Link Verification Strategy focused more on two protective majors, 1) how to achieve unforgeability of the LLDP packet and 2) Effectively detect a relayed LLDP packet. The authors used chaotic model based on chaotic motion which is both common and complex in the linear dynamic systems. In specific terms, the chaotic motion is complex and non-repetitive movement which is always confined to a limited area. It has the characteristics of avalanche effect as any slight change to the initial condition greatly affects the system state. Packet hop-count is also used in the network to verify relay-type link fabrication attack since the construction and processing of the LLDP packet will result in packet transmission delay. The authors also add timeline to the generated passwords to prevent relayed packet reaching the controller for topology updates.

The approach of [5] -TrustTopo is able to place integrity on the generated LLDP packet for topology change request at the controller, makes use of dynamic packet at every round of LLDP generation, and takes care to protect the controller from the three types of attacks namely: Poisoning, Flooding and Replay. They have however not solved the problems with the packet generation overhead that occurs at the controller. These problems seem to be general with almost all the research works reviewed.

The primary gap identified across the literature is the challenge of balancing security with operational efficiency. While advanced security measures like encryption and dynamic packet generation help protect the network, they often come at the cost of increased resource utilization and network latency. Future research should focus on developing lightweight, scalable solutions that minimize packet overhead while maintaining robust security. Mechanisms such as on-demand topology verification, improved bidirectional checks, and better utilization of existing secure channels (e.g., OpenFlow) offer promising directions for addressing these challenges.

CONCLUSION

This survey has reviewed various approaches aimed at enhancing topology discovery and security in Software Defined Networking (SDN) with a focus on mitigating common vulnerabilities such as topology poisoning, link fabrication, and replay attacks. The reviewed literature highlights several innovative techniques, including TOPOGUARD's device-type verification, SPHINX's anomaly detection, OFDP-HMAC's cryptographic integrity checks, and TrustTopo's dynamic password generation and path tracking mechanisms. Each approach contributes, particularly in securing the topology discovery process in SDN, but of the approaches still suffer from inefficiencies, such as excessive resource use due to fixed periodic LLDP broadcasts or the need for complex key and password management. Additionally, several of the models introduce added administrative burden through policy-based systems or encryption overhead, which may limit scalability and performance in larger or more dynamic networks.

REFERENCES

[1] Alharbi T., Portmann M., and Pakzad F. (2015, October). The (in)security of topology discovery in software defined networks. In 40th IEEE Conference on Local Computer Networks, LCN 2015, Clearwater Beach, FL, USA, pages 502–505.

[2] Dhawan M., Poddar R., Mahajan K., and Mann V. (2015, January). SPHINX: Detecting Security Attacks in Software-Defined Networks. DOI: 10.14722/ndss.2015.23064

[3] Duan, Q., N. Ansari, and M. Toy (2016, October). Software-Defined Network Virtualization: An architectural framework for integrating SDN and NFV for service provisioning in future networks. IEEE Network.

[4] Hong S., Xu L., Wang H. and Gu G. (2015, February). Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures, in: Proc. of Annual Network and Distributed System Security Symposium (NDSS'15).

[5] Huang, X., P. Shi, Y. Liu, and F. Xu (2020, April). TrustTopo, a lightweight and efficient SDN topology verification scheme. East China Normal University, Shanghai, China. Computer Networks 170 (2020), 107119.

[6] Jain, S., A. Kumar, S. Mandal, and J. Ong et al. (2013, August). B4: Experience with a globally-deployed Software Defined WAN. In ACM SIGCOMM Conference. Pages 3–14.

[7] Jammal, M., T. Singh, A. Shami, and R. Asal et al. (2014, July). Software defined networking: State of the art and research challenges. Computer Networks 72 (0), 74–98. DOI: 10.1016/j.comnet.2014.07.004.

[8] Jimenez, Y., C. Cervello-Pastor, and A. Garcia (May, 2015). Dynamic resource discovery protocol for software defined networks. IEEE Commun. Lett. 19(5), 743–746.

[9] Kloti, R., V. Kotronis, and P. Smith (2013, October). OpenFlow: A security analysis. In IEEE International Conference on Network Protocols (ICNP). Pages 1–6.

[10] McKeown, N., T. Anderson, H. Balakrishnan, and G. Parulkar et al. (2008, March). OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, pages 69–74.

[11] Nehra, M., M. Tripathi, S. Gaur, and R. B. Battula et al. (2018, December). SLDP: A secure and lightweight link discovery protocol for software-defined networking.

[12] Ochoa-Aday, L., C. Cervelló-Pastor, and A. Fernández-Fernándeza (2016, November). Discovering the Network Topology: An efficient approach for software-defined networks. Advances in Distributed Computing and Artificial Intelligence Journal. ADCAIJ, Regular Issue Vol. 5 N. 2. http://adcaij.usal.es.

[13] Ochoa-Aday, L., C. Cervello-Pastor, and A. Fernandez-Fernandez (March, 2018). Self-healing topology discovery protocol for software-defined networks. IEEE Commun. Lett. 22(5), 1070–1073.

[14] Pakzad, F., M. Portmann, W. Tan, and J. Indulska (2015, September). Efficient topology discovery in OpenFlow-based software defined networks. Computer Communications, DOI: 10.1016/j.comcom.2015.09.013.

[15] Tarnaras, G., E. Haleplidis, and S. Denazis (April, 2015). SDN and FORCES-based optimal network topology discovery. Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), 1–6.

[16] Wang, H., G. Yang, P. Chinprutthiwong, and L. Xu et al. (2018, October). Towards fine-grained network security forensics and diagnosis in the SDN era. In ACM SIGSAC Conference on Computer and Communications Security (CCS), pages 3–16.

[17] Xu, L., J. Huang, S. Hong, and J. Zhang et al. (2018, August). Attacking the brain: Races in the SDN control plane. In USENIX Security Symposium, pages 451–468.

[18] Xue, L., X. Ma, X. Luo, and E. W.W. Chan et al. (2018, October). LinkScope: Towards detecting target link flooding attacks. IEEE Transactions on Information Forensics and Security (TIFS).

[19] Zhang, M., G. Li, L. Xu, and J. Bi et al. (2018, September). Control plane reflection attacks in SDNs: New attacks and countermeasures. In Symposium on Research in Attacks, Intrusions, and Defenses (RAID).

[20] Zuo, Z., R. He, X. Zhu, and C. Chang (2019, May). A novel software-defined network packet security tunnel forwarding mechanism. Mathematical Biosciences and Engineering (MBE), 16(5), 4359–4381. DOI: 10.3934/mbe.2019217.

[21] (2021, March 30). Retrieved from OpenDaylight: URL http://www.opendaylight.org/project/technical-overview.