electronics

MDPI

Article

A Link Fabrication Attack Mitigation Approach (LiFAMA) for Software Defined Networks

Katongole Joseph [1], Odongo Steven Eyobu [1,2,*], Philemon Kasyoka [3] and Tonny J. Oyana [2]

[1] Department of Networks, School of Computing and Informatics Technology, Makerere University, Plot 56, Pool Road, Kampala P.O. Box 7062, Uganda; kajoseph.katongole@mak.ac.ug
[2] Geospatial Data and Computational Intelligence Lab, Makerere University, Plot 56, Pool Road, Kampala P.O. Box 7062, Uganda; toyana@cis.mak.ac.ug
[3] Department of Computing and Information Technology, School of Pure and Applied Sciences, University of Embu, Embu P.O. Box 6-60100, Kenya; kasyoka.philemon@embuni.ac.ke
* Correspondence: odongo.eyobu@mak.ac.ug

Abstract: In software defined networks (SDNs), the controller is a critical resource, yet it is a potential target for attacks as well. The conventional OpenFlow Discovery Protocol (OFPD) used in building the topological view for the controller has vulnerabilities that easily allow attackers to poison the network topology by creating fabricated links with malicious effects. OFDP makes use of the link layer discovery protocol (LLDP) to discover existing links. However, the LLDP is not efficient at fabricated link detection. Existing approaches to mitigating this problem have mostly been passive approaches that depend on observing unexpected behaviour. Examples of such behaviour include link latency and packet patterns to trigger attack alerts. The problem with the existing solutions is that their implementations cause longer link discovery time. This implies that a dense SDN would suffer from huge delays in the link discovery process. In this study, we propose a link fabrication attack (LFA) mitigation approach (LiFAMA), which is an active mitigation approach and one that minimises the link discovery time. The approach uses LLDP packet authentication together with keyed-hash-based message authentication code (HMAC) and a link verification database (PostgreSQL) that stores records of all known and verified links in the network. This approach was implemented in an emulated SDN environment using Mininet and a Python-based open-source OpenFlow (POX) controller. The results show that the approach detects fabricated links in an SDN in real time and helps mitigate them. Additionally, the link discovery time of LiFAMA out-competes that of an existing LFA mitigation approach.

Keywords: SDN security; link verification; topology discovery

Citation: Joseph, K.; Eyobu, O.S.; Kasyoka, P.; Oyana, T.J. A Link Fabrication Attack Mitigation Approach (LiFAMA) for Software Defined Networks. *Electronics* **2022**, *11*, 1581. https://doi.org/10.3390/electronics11101581

Academic Editors: Panagiotis Sarigiannidis, Thomas Lagkas, Vasileios Argyriou and Panagiotis Radoglou-Grammatikis

Received: 23 February 2022
Accepted: 12 April 2022
Published: 16 May 2022

1. Introduction

Each software defined network's (SDN) architecture is composed of two separate planes. These are the control and data planes. This design offers flexibility, as it improves network control and provides network engineers an easy way to respond to network requirements. SDN is therefore a good approach for building networks and transforming dynamic networks into manageable, simplified, centralised ones [1]. Recent research has shown that traditional networks cannot easily match the growth in demand, as each device is independent and carries its own brain.

The SDN generally represents the concept of programmable networks. This concept allows network administrators to run inexpensive hardware which is not vendor-specific/proprietary because of the programmability that allows system configuration at the controller. Additionally, the end devices do not require any intelligence except flows which are installed on them by the controller [2]. From the controller, the global network picture can be seen for all connected devices [3]. This is done by abstraction of the network logic from hardware implementation into software and promotes network innovation [4].

The OpenFlow protocol is the most common southbound application programming interface (API) for SDN. This protocol has a number of vulnerabilities that can be exploited by adversaries in order to poison the controller's view of the network. This is mainly done by crafting LLDP packets and injecting them into the network through a compromised host. LLDP packets are sent out by the controller during topology discovery as packet-out messages go to the switches and switches send packet-in messages back to the controller. Based on the path taken by each LLDP packet, the controller builds the topology links between different ports of the switches. Unfortunately, adversaries may craft these packets with specific parameters statically inserted, and the controller may not have a way of distinguishing between genuine and false packets. This creates forged or fabricated links, hence the term link fabrication attacks (LFAs).

SDNs provide a number of benefits, including solving and simplifying critical management tasks that traditional network management may not be able to. For example, SDNs solve the device discovery problem in traditional network management by default through a controller [4].

The advancements in networking introduced by SDNs present security risks and vulnerabilities, just like any other system. OpenFlow controllers suffer from two kinds of topology poisoning attacks, link fabrication and host hijacking attacks, as a result of the fact that they use OFDP for topology discovery [3].

The motivation behind this research was that maintaining security in SDNs should not have a high cost in terms of delays in the link discovery process, which is exhibited by the existing approaches [5–7] that mitigate link fabrication attacks. This research mainly focused of developing a link fabrication attack mitigation approach (LiFAMA) which makes use of the LLDP and a database. The key contributions of this research are therefore summarised as follows:

- A link fabrication mitigation approach that minimises the link discovery time compared to conventional approaches while maintaining security.
- A performance analysis of the traditional LLDP, other existing LFA mitigation approaches and LiFAMA using link discovery and verification time as standard metrics.
- An LLDP-based packet authentication approach with dynamic keys using hash-based message authentication code (HMAC).
- A database approach for the link discovery process that stores verified links in a Structured Query Language (SQL) database.

The rest of the paper is organised as follows: Section 2 discusses related works, Section 3 presents the relevant technical preliminary information concerning SDN, topology discovery and HMAC; Section 4 presents the link fabrication attack model; Section 5 presents our proposed LiFAMA; Section 6 describes the security analysis of LiFAMA, Section 7 describes the experiments carried out and tools used in the study; Section 8 discusses results and performance metrics used; and Section 9 concludes the study.

2. Related Work

A vast number of studies [5–9] on link fabrication attacks in SDNs have proposed solutions which are heavily predictive and behavioural. However, such approaches are not reliable, as many false positives are reported. The next paragraphs provide a review of existing works that report link fabrication and attack control approaches in SDNs.

Dylan et al. [5] proposed a fabricated link detection mechanism based on latency measurements of the LLDP packet from the time a packet-out is sent to the time a packet-in is received at the controller. The detection method is based on a statistical analysis of the various data link latency measurements realised. The major challenge with this type of mechanism is the number of false positives, and yet there many methods that do not have a finite expected output. A vetting process was added, but it also creates the possibility of having false positives hence, making it an unideal discovery method. The latency differences are not sufficient to distinguish genuine links from fabricated ones.

Dhawan et al. [6] discussed a number of security attacks in SDN and proposed SPHINX as a solution to these attacks. SPHINX intercepts all network communications between switches and the controller. Messages exchanged between the controller and switches include packet-in and feature-replies. It compares detected network behaviour with expected or predefined behaviour or policies defined within the controller. It is designed to detect both known and unknown attacks on SDN topology through learning network behaviour. This poses a challenge, as the system might encounter attacks that it might not have been trained to stop or even block legitimate traffic. SPHINX is generally a data or knowledge-driven solution whose detection accuracy will heavily depend on how much information is in the knowledge-base. Systems such as SPHNIX can be improved by introducing the use of semi-supervised learning methods to encounter unknown data. Currently, machine learning methods are prominent in the detection of attacks in networks where big-data can be extracted.

Sanaz et al. [9] introduced the link latency attack (LLA) that they claim could not be detected by TopoGuard and TopoGuard+. LLA is an attack where an adversary adds a fake or fabricated link through compromised hosts within the network. This is done by injecting packets into the network, thereby increasing the processing time for the packets. TopoGuard depends on latency, and with this attack, it could be evaded. They highlighted that the conventional discovery process lacks authenticity and is vulnerable to topology poisoning attacks such as the LLA. They proposed a machine learning-based link guard (MLLG) system to prevent this attack. MLLG verifies the LLDP packet against a dataset containing various LFAs to validate whether or not the link is valid. The action taken is to either drop the packet or update the topology database. MLLG was validated against traditional machine learning methods, including k-nearest neighbour (kNN), multi-layer perceptron (MLP), logistic regression (LR), support vector machines (SVM) and naive Bayes (NB). Sanaz et al. [9] did not report on the time MLLG takes to detect the LLA, yet this is important for mission-critical SDNs where any data loss is a major effect. Validation using other deep learning approaches, such as convolutional neural networks and long short term memory networks, would help with gaining further knowledge on which intelligent model will offer the best detection time.

Sungmin et al. [7] presented a number of vulnerabilities that exist in OpenFlow that need to be addressed in order to have a secure SDN environment. These include network topology poisoning and exploitation of the host tracking service. The processes of link fabrication and replay attacks are explained, and they clearly show the challenges faced by these kind of attacks. Then, they propose TopoGuard, which was implemented in a floodlight controller. TopoGuard is a real-time network topology poisoning detection system that implements packet authentication in order to verify packet authenticity. However, the TopoGuard mechanism is still susceptible to replay attacks and LLAs, as in Sanaz et al. [9]. Prasad et al. [10] further notes that Topoguard only considers network topology poisoning. This is why the LLA in [9] could not be detected by TopoGuard. Skowyra et al. [11] defines port amnesia and port probing as topology attacks against TopoGuard and hence proposed TopoGuard+. SDN attack detection tools or systems must therefore be designed to cover diverse possible attack windows. The key issue is to gain knowledge on all possible attack windows, yet this would require a comprehensive penetration test to be done.

Alimohammadifar et al. [8] proposed the stealthy probing-based verification (SPV) approach for detecting fabricated links in SDN. The SPV system works by sending out probing packets at certain intervals and marking links as legitimate or not based on the responses received from the probing process. Only probing packets are authenticated in order to prevent forging and replaying of probing packets. SPV allows creation of the link, and in the event that the probing process is not successfully sent back to the controller, the link will be taken as a fake link, and hence traffic may not be sent over the fabricated link. SPV cannot defend against attackers communicating entirely over an out-of-band channel and creating links that look like genuine links; this could partly be because LLDP packets are not authenticated. SPV performance can also be poor when probing packets are lost

during the channel communication, because SPV will assume that such packets were not returned to the controller due to being fake links. In order to mitigate the weakness in SPV, the LineSweep algorithm proposed in Kim [12] and Lin [13] can be used.

Sonali et al. [14] discussed the link discovery process and vulnerabilities as a result of compromised switches. They proposed a mitigation mechanism that uses active-ports for detection of both host-based and switch-based link discovery attacks in SDN. They provided an analysis of the impact of topology attacks on routing. They introduced a switch-based link discovery attack where switches are compromised by changing the switch MAC –table with forged MAC addresses. The mechanism proposed can detect both host-based and switch-based attacks. The proposed mechanism monitors active ports on every switch and active links in the network. An active port cannot connect multiple links at the same time. The process is monitored through link validation in the switch table.

Noemi et al. [15] sought to simplify network applications by introducing abstraction through the use of databases. They proposed the use of views to abstract the actual table structures where the data are stored, to add an extra layer of security. Their main aim was to manage access control with the SDN network through reflective specification of access control rules.

The proposed LiFAMA, in contrast to [5–7,9,11], is an active LFA mitigation mechanism that is not based on expected behaviour of packets but rather predefined settings within the controller using a link database and packet authentication. This makes management simple, as the network administrator only focuses on having the correct network parameters configured. It also makes implementation less resource-intensive and more efficient.

Table 1 shows a summary of existing LFA mitigation approaches for SDNs.

Table 1. Some existing LFA detection and mitigation approaches.

Algorithm	Type of Detection	Packet Authentication	Database
Smyth et al. [5]	Passive	No	No
Dhawan et al. [6]	Passive	No	No
Sungmin et al. [7]	Passive	No	No
Skowyra et al. [11]	Passive	No	No
Alimohammadifar et al. [8]	Active	Only probing packet	No
Sanaz et al. [9]	Passive	No	Dataset of LFAs
Our Proposed LiFAMA	Active	Yes	Yes

3. Technical Preliminaries

In this section, the SDN architectural composition, SDN protocols and cryptographic-hash-function HMAC are described.

3.1. SDN Architecture

A typical SDN implementation architecture comprises three layers. These include the application layer, the control layer and the infrastructure layer.

Application Layer: It contains network applications used by organisations, such as intrusion detection systems, firewalls, traffic policies and load balancers. In traditional networks, all the above applications would run on specialised hardware, but SDN replaces that specialized hardware with an application that uses the controller to manage behaviour of the hardware devices [16]. It is responsible for handling security and business applications, in summary. This layer communicates with the control layer over the north-bound interface, as shown in Figure 1, using the application control plane interface (A-CPI) [1].

Control Layer: This layer comprises centralised SDN controller software where all the intelligence of the SDN resides. The controller manages traffic flow and policies throughout the network [16] and packet dropping through programming [1]. It controls the overall SDN functions. In a distributed environment, controllers communicate with each other through the east-bound and west-bound interfaces. The controller communicates with the infrastructure layer on the south-bound API, normally using a protocol such as OpenFlow or netconf [1].

Infrastructure Layer: Commonly known as the data plane, the infrastructure layer is made up of both virtual and physical devices, such as switches, routers and access points to which other devices in the network connect [1]. The main function of the data plane is forwarding packets according to assigned rules and policies.

An SDN encompasses several types of technologies, including functional separation, network virtualisation and automation through programmability. It presents an opportunity for programmers to innovate due to its open nature.

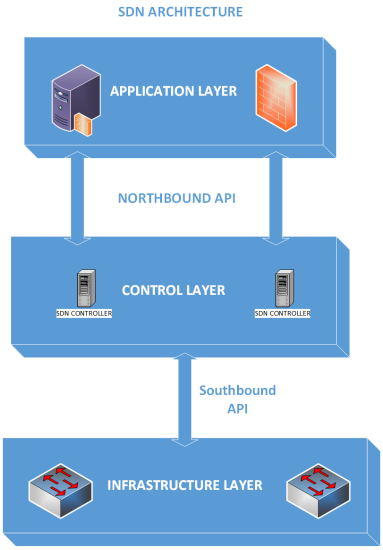

Figure 1. SDN implementation architecture.

3.2. Topologies

Mininent provides various default network topologies and provides the ability to create custom topologies for real life network situations. However, in this work, two topologies inbuilt in Mininent were used, i.e., the tree and linear topologies in Figures 2 and 3, respectively. The reason for choosing these topologies is that the number of links and varies for the same number of switches in the network. Usually, tree topology has more links than the linear topology, creating a good environment to test the link discovery process with our proposed method.

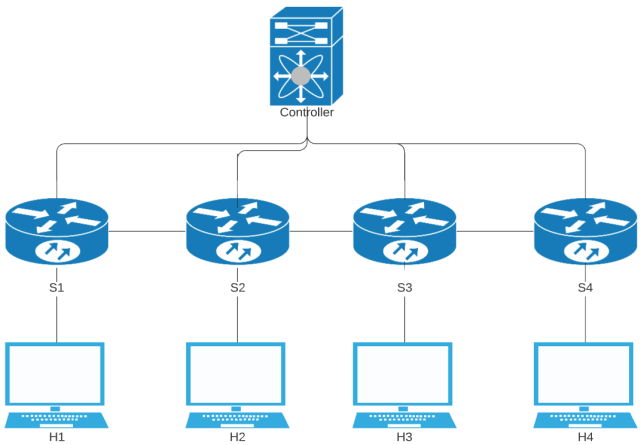

Figure 2. Linear topology.

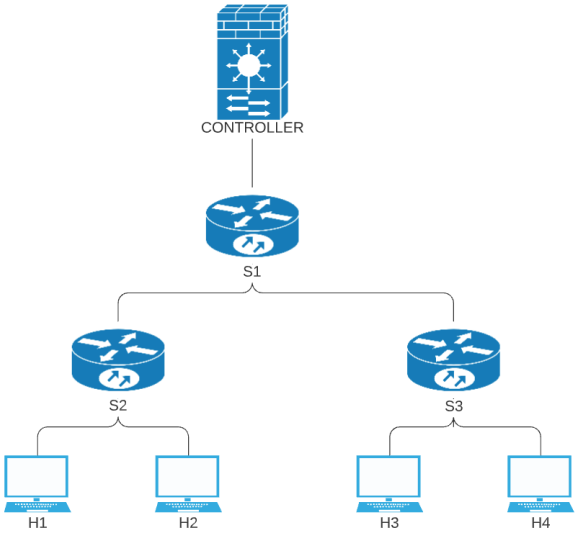

Figure 3. Tree topology.

3.3. SDN Protocols

3.3.1. OpenFlow Discovery Protocol (OFDP)

OFDP is the protocol used by OpenFlow controllers to discover the underlying topology [17]. The protocol allows the controller to communicate with the less intelligent forwarding elements such as switches to install flows and forwarding tables. OpenFlow leverages the OFDP protocol in order to achieve two-way communication between switches and controllers [17]. In an SDN, OFDP floods the LLDP with minor changes in order to carry out topology discovery [18]. OFDP works with advertisements only.

3.3.2. Link Layer Discover Protocol (LLDP)

LLDP is a layer-2 protocol that permits directly connected devices to advertise the existence of physical connections between them. The protocol is based on IEEE 802, which is mainly wired [18]. It is a vendor neutral protocol, unlike proprietary protocols such as Cisco Discovery Protocol and Foundry Discovery Protocol, which perform similar functions to LLDP. LLDP broadcasts the capabilities of the switches.

LLDP packets are sent from each connected interface at a fixed interval as an Ethernet frame. The basic LLDP PDU comprises a header and type-length-value (TLV) attributes [19]. Each LLDP packet has both mandatory and optional TLVs. The mandatory TLVs are Chassis ID, which is unique for every switch, Port ID and Time to Live. For POX controller, two more TLVs are added to the packet-out message, i.e., System Description and End TLV. Other optional/custom TLVs can be added to carry information relevant for any use the programmer may deem fit for his or her requirements [20]. Figure 4 shows the LLDP packet format. Table 2 shows all the TLVs for the LLDP packet.

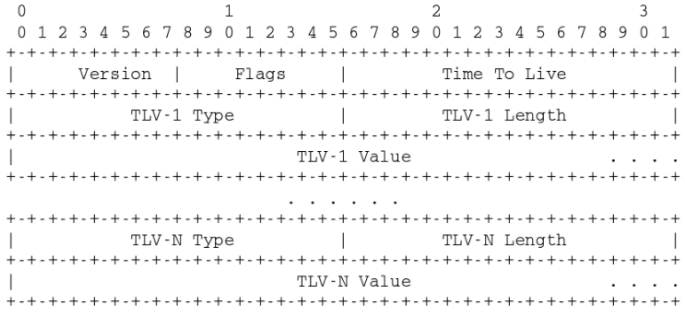

Figure 4. LLDP packet format. Adapted from [19].

Table 2. TLV type values. Adapted from [18].

TLV Type	TLV Name	Usage in LLDPDU
0	End of LLDPDU	Mandatory
1	Chassis ID	Mandatory
2	Port ID	Mandatory
3	Time to Live	Mandatory
4	Port Description	Optional
5	System Name ID	Optional
6	System Description	Optional
7	System Capabilities	Optional
8	Management Address	Optional
9–125	Reserved	Optional
127	Custom TLVs	Optional

An LLDP packet is represented by a dashed line in Figure 5. At first, the controller creates a LLDP packet and sends it to switch S1 as a packet-out message. On receiving the packet-out message, the switch will forward the LLDP packet-out port P1 because the instruction set in the packet-out message contains an instruction: outport = P2. After receiving the LLDP packet from switch S1, switch S2 will send a packet-in message that carries the LLDP packet to the controller according to a pre-installed rule in its flow table. The packet-in message contains some fields to denote the ingress switch S2 and the ingress port P3. Besides, the LLDP packet has the information about the source switch ID and source port ID (i.e., Chassis ID and Port ID). Therefore, the controller will deduce that there exists an internal link from (S1, P2) to (S2, P3).

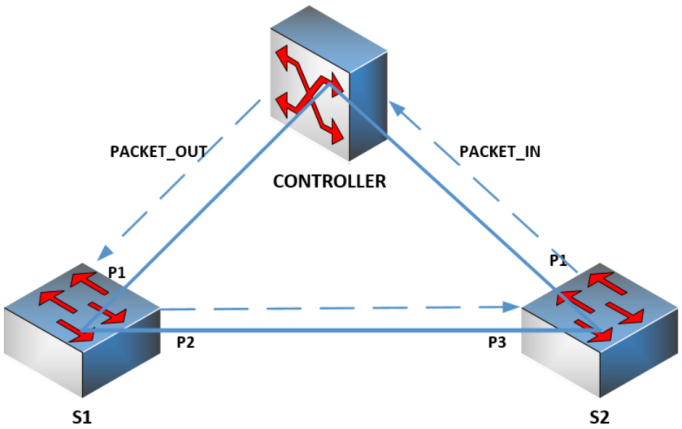

Figure 5. Link discovery process.

3.4. Keyed-Hash Based Message Authentication Code (HMAC)

The hash-based message authentication code (HMAC), also known as keyed hashed MAC, is a cryptographic algorithm that uses both a key and a cyptographic hash function. Hash functions that may be used are SHA-1, SHA-256 and MD5. It provides great resistance to crypto-analysis because of the fact that messages are hashed twice through the IPAD and OPAD. HMACs use cryptographic hash functions and keys as a method of authentication and verification of information [21].

HMACs provide a way of checking the integrity and authenticity of information transmitted over or stored over an insecure channel. In a typical scenario, HMACs may be used between two or more parties. However, when multiple parties are involved, it is advised that separate keys are used between each pair of communicating parties, since the key is used in calculation of authentication values.

3.4.1. Why HMAC

- HMACs use hash functions without making changes to the code.
- HMACs handle keys a simple way
- HMACs make it easy to change the underlying hash function in the event that the underlying hash function used is found to be weak.

The major advantage of using hash functions and HMAC in general is that they are fast compared to asymmetric key cryptographic algorithms.

The HMAC function comprises a secret key K and a message M over which the $HMAC(K, m)$ value is computed using Equations (1) and (2).

$$HMAC(K, M) = H(K \oplus OPAD \parallel H(K \oplus IPAD \parallel M)) \quad (1)$$

$$HMAC(K, M) = H(K^{+} \parallel H(K^{*} \parallel M)) \quad (2)$$

where $H()$ is the one-way cryptographic hash function, $\parallel$ stands for concatenation and $\oplus$ denotes the XOR or bitwise exclusive or operation. Outerpad (OPAD) and Innerpad (IPAD) are blocks that consist of repeated bytes of 0x5c and 0x36, respectively. K^{+} and K^{*} are keys derived by XORing key K with OPAD and IPAD, respectively.

3.4.2. HMAC Structure

- H: Hash function (e.g., MD5, SHA-1)
- IV: Initialisation vector as an input to the hash function. It is an arbitrary fixed-length number used once during an iteration. It is also called a nounce, meaning number used once.
- M: Message input to HMAC broken down into equal blocks from $m_1, m_2, \ldots, m_n$, including the padding block if required, as in Figure 6.
- K^{+}, K^{*}: Keys derived by XORing key k with IPAD and OPAD, respectively.

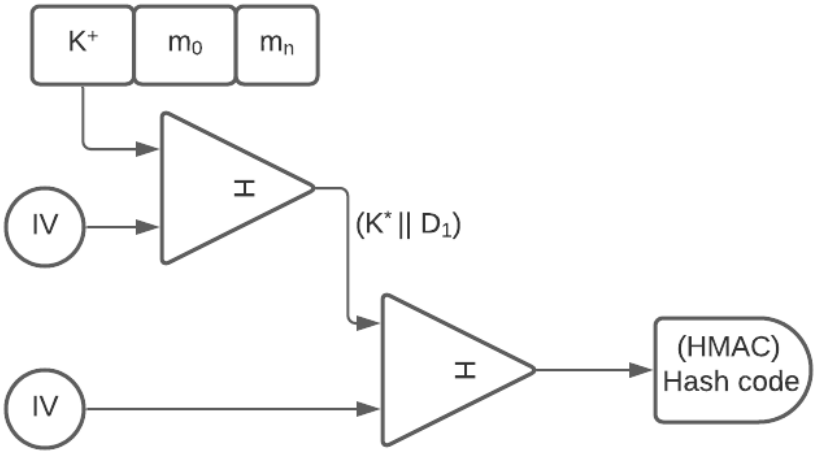

Figure 6. HMAC generation. Adapted from [22].

3.4.3. How HMAC Works

Key K is shared between the sender and the receiver. From the secret key K, two keys are derived by padding K with two constants, i.e., the inner-pad (IPAD) and the outer-pad (OPAD), to achieve a fixed-length key K^{+}. Equation (1) is equal to Equation (2).

The hashing algorithm is applied to $(K^* \parallel D_1)$ to generate an n-bit output called the hash code. The first algorithm produces an internal hash of the first derived key using the IPAD concatenated with the message. The final HMAC code is derived from the result of the first hash with the second key derived from the OPAD. Figure 6 shows the generation of the hash code from keys K^+ and K^* [22].

1. Key K is selected such that $0 < K < b$. If $K < b$, pad 0s to on the left until $K = b$, where b is the block size chosen based on the hashing algorithm to be used. For example, for sha256 the value of **b** would be 256 bits.
2. XOR k with IPAD to generate the first derived key K^+
3. Append K^+ with the message M that is broken down into blocks m_0, m_n.
4. Apply the hashing Algorithm, H, to $(K^+ \parallel M)$ to generate a fixed-length, n-bit output digest, D_1.
5. Pad the n bits until the length of the digests is b bits long.
6. XOR K with the OPAD to give a b-bits output K^*
7. Apply the hash algorithm H to $(K^* \parallel D_1)$, to generate the hash code.
8. The value of the hash code can then be transmitted.

4. Link Fabrication Attack Model

LFA's aim at making the controller perceive that a direct inter-switch link exists, yet it is non-extent. The link is normally through an attacker. It is an exploitation of the link discovery service because of the existence security flaws. Most OpenFlow controllers use OFDP, and OFDP leverages LLDP. Switches in the network observe the flooded LLDP traffic and notify the controller with this traffic that links exist between them. The flooded traffic can be observed by an eavesdropper who can craft LLDP packets and inject them into the network. The controller, upon receiving these packets, processes them and can infer existence links. There could be genuine and fabricated links. The link discovery service in OpenFlow controllers is subject to two conditions.

- The propagation path of LLDP frames can only be contained in OpenFlow enabled switches.
- The integrity of the LLDP frames must be ensured during the topology formation procedure.

An attacker can execute an LFA using at least one malicious host. Case 1 in Figure 7 shows a host H3 directly connected to two switches, hence transparently creating a bridge between two interfaces of the switches. In cases where the physical distance is a hurdle, this can be overcome by using an out-of-band communication channel, such as a wireless one. By using two malicious hosts that communicate through an out-of-band wireless channel, the attacker can avoid some of these physical restrictions and fabricate links between SDN switches which are further apart. Crafted packets can be injected into the network using scapy [23], a packet manipulation tool. In case 2, two compromised hosts, H1 and H2, can be used to infer a link between the SW1 and SW2.

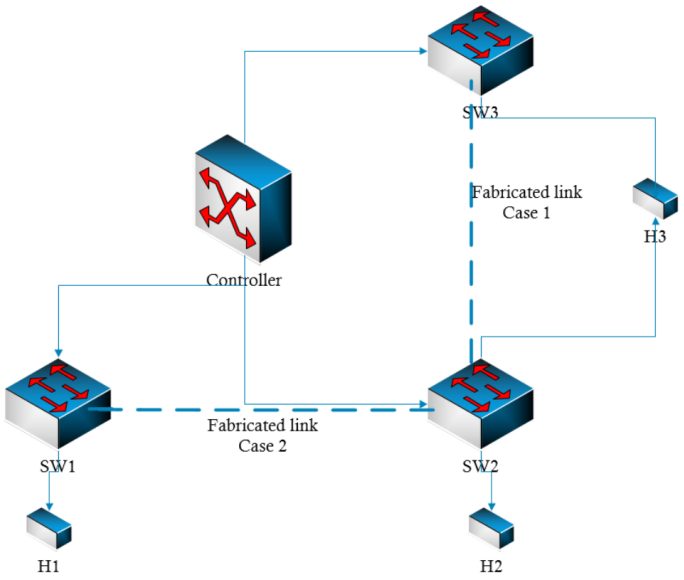

Figure 7. Attack scenarios.

5. Proposed Link Fabrication Attack Mitigation Algorithm (LiFAMA)

As OpenFlow discovery protocol packets sent over the network can be analysed by an attacker connected to any switch within that network, injection of packets is easy. This kind of environment is deemed to be an insecure one. However, the LLDP packet used for the discovery process gives leverage to add custom TLVs. In this study, we propose the addition of authentication information to every packet-out. Addition of this authentication information in turn ensures the integrity of the packet.

As every SDN is run in a programmable environment, other functionality extensions can be installed for various purposes depending on the problem to be solved. In this study, we installed a link verification database. As the name suggests, it is a database for all known and approved links within the network, and all link discovery packets must be checked against it. The introduction of a database provides inter-operability across a wide range of platforms, as databases create abstractions of the logic that simplifies the management of an SDN. Connections to the database are only allowed to come from the controller, as there is no need for external devices to connect, hence creating an extra layer of security for the database. For cases where the database is external to the controller or on a separate host for reasons such as high availability, measures must be taken to ensure that such databases are safe from unauthorised modification of records.

For packet integrity and authentication, we use keyed-hash based message authentication code (HMAC) as the feature to add to the LLDP packet, and for the link verification database we used PostgreSQL, a structured query language-based database. The addition of these measures ensures a secure topology discovery process.

Figure 8 details the link discovery process, from generating the packet-out message to the packet-in message being received and processed by the controller. The process begins with the generation of an HMAC that is to be embedded in the controller's packet-out message as a TLV. The generated HMAC is sent out as part of the discovery packet. For every link, a unique HMAC value is sent.

The packet-in message received by the controller is checked to see if it matches the packet-out, if it does not, the controller discards the packet as a potentially forged packet, but if it does, another layer of validation is in place called a reference database for all links expected within that control domain. If the second condition is not met, the packet-out is discarded. This scenario will normally happen if the link is genuine but the link database has not been updated. If all these checks are passed, the link will then be created. This algorithm will help stop all sorts of link fabrication attacks, including both host-based and switch-based topology tampering methods.

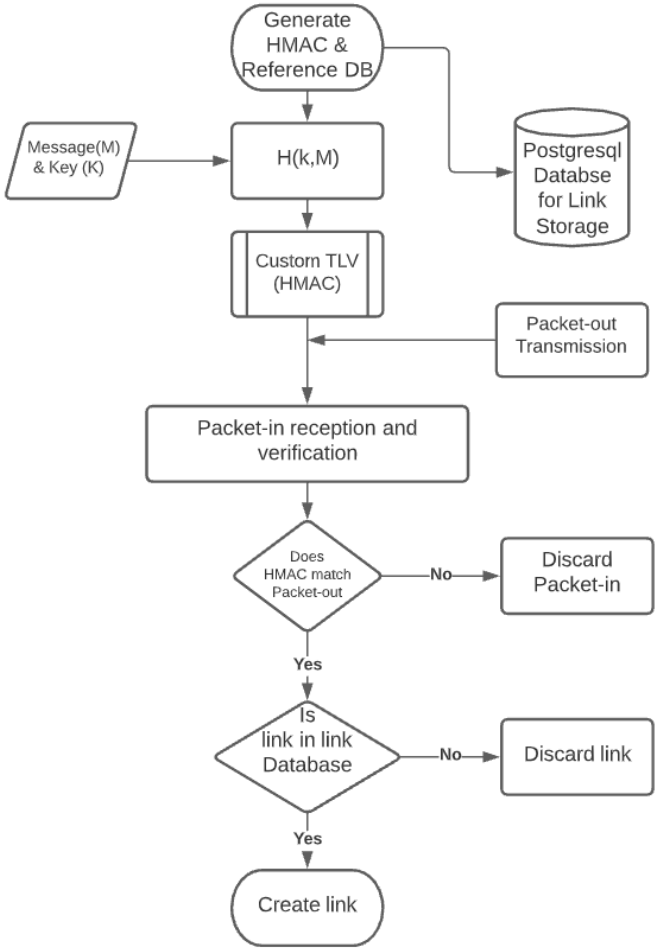

Figure 8. Proposed link fabrication mitigation algorithm (LiFAMA).

The LLDP packet being sent over the network is assumed to be sent over an insecure channel that has no authentication; therefore, a mechanism of verification is required that ensures the LLDP packet-in received at the controller is as a result of a packet-out message sent by the controller. HMAC provides this authentication mechanism and ensures the integrity of the controller's view of the network. HMAC is known to be vulnerable to replay attacks. The approach to prevent this is the use of a dynamic key for every packet-out message sent and making the controller configured to validate every received HMAC against the computed HMAC in the packet-out message. The SHA-1 hashing algorithm is used in the HMAC function.

6. Security Analysis of LiFAMA

In this section, an analysis of the introduction of the HMAC and link database to the SDN environment is presented.

Theorem 1. *No key sharing required.*

One weakness of the HMAC is compromising/revealing the key by/to an adversary. This normally happens during the key-exchange process if it is not securely managed. For verification of authenticity, the receiver needs the key in order to compute the MAC. However, there is no key sharing, since the controller which sends out the LLDP packet is in turn charged with verification of the same message upon reception from the switches.

Theorem 2. *Security of HMAC.*

HMAC has been proved to be a pseudo-random function (PRF), since the underlying compression function is believed to be a PRF. A PRF is a function that generates an output from input

variables that is indistinguishable from a truly random one [24]. To mitigate extension attacks that may exist in ordinary hash functions, consider a message M, a key K concatenated and hashed together by a hash function H(). Hash functions follow the Merkle–Damgard construction [25]. The message M is broken down into equal blocks $m_0, m_1, m_2, \ldots, m_n$. *The hash function is initialised with an initialisation vector (IV), with the first block of message* m_1, *and fed into a compression function* f. *The output of* m_0 *becomes the input of* m_2 *until* m_n. *At the output stage of* m_n, *if the adversary knows the tag, he/she can add one or more extra block and compute the tag(s). Figure 9 shows the normal MAC generation that is susceptible to extension attacks. With this, an adversary can compute the original message concatenated with the padding block. HMAC is not susceptible to these attacks, since the internal hash using the IPAD is not exposed to the adversary due to the two stages of hashing involved.*

HMAC is considered a secure function with the following assumptions. Consider the number of messages X and the Tag space T [26].

$$X << |T|^{1/2}$$

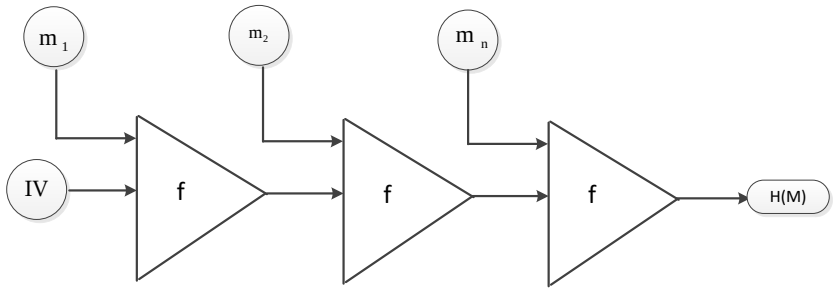

Figure 9. Hash message generation.

The number of messages X tagged is less than the square root of the output tag space, T. As for the message authentication, compromising the HMAC does not compromise traffic that has already been authenticated. This very important to note, especially if the key is not dynamic. In this case, a different key is used for every LLDP discovery, so cracking the key will not be important to the attacker, hence ensuring a secure ecosystem with the SDN environment.

Theorem 3. *The scheme provides LLDP authentication.*

Packet authentication enables verification of the packet-in corresponding to a matching packet-out sent by the controller. An adversary will have to compute a dynamically generated HMAC with a dynamic key to be able to break this security feature.

Theorem 4. *Link Database.*

Creating a link database as a reference database adds an extra layer of security so that an adversary would need to create a fabricated link by also injecting a false link into the PostgreSQL database as well.

7. Experiments

This section shows a summary of the tools and environments that formed the basis of our investigational assessment. A key program used for the experiments was Mininet [27], a Linux-based network emulator. Mininet is a network emulator program used extensively in SDN simulations with the ability to create virtual switches, hosts, controls and links. Mininet can switch support for OpenFlow, a defacto protocol for SDNs, and it has the ability to run code that can be run on real hardware, making the transition from the test bed to the hardware environment easy. The controller used in this research was POX [28], an open-source python-based SDN controller that provides rapid prototyping. The environment was set up Mac OSX operating system running Oracle Virtual Box with Ubuntu 18.04 LTS version (RAM: 3 GB, storage: 20 GB). We limited the speed of links in our experiments to 100 Mbps. Table 3 shows the experimental tool set used to generate the results in this study.

Table 3. Experimental tool set.

Application	Function
Mininet	SDN Emulator
Oracle Virtualbox	Virtualization Software
POX	Controller
Ubuntu	Operating System
Python	Programming Language
PostgreSQL 12	Database

In our experiments, using the environment in Table 3, we simulated three types of topologies. These were the linear, tree and custom topologies with varying numbers of switches. The studied topologies, when implemented, presented different numbers of links for the same numbers of switches. We therefore chose the three topologies for the purpose of exploiting the diversity of possible link fabrication attacks in both simple and complex networks. Generally, the linear topology presents a simpler topological structure compared to the tree and custom structures. An existing LFA mitigation algorithm called SPV [8] was also run on the three topologies to validate our proposed LiFAMA.

For each topology, the number of switches was varied from 2 to 40 in the experiment. The time whereat the first discovery packet was sent out and the time at which the last discovery packet was received were recorded. These were used to calculate the discovery time. For every switch topology, we first populated the database with a list of known links to act as a reference database of known links for that topology. We ran three trials and recorded the discovery times for each of those trials, which we later used to calculate the average discovery time. To simulate and attack, we crafted LLDP packets using [23] a packet generating tool and injected them into the environment.

Figure 10 shows the output of the controller after implementation of our discovery algorithm. The discovery times are shown later. Due to the nature of the OpenFlow protocol, an attacker can easily inject fabricated links within the network. For purposes of visualising our experiment, as seen Figure 10, we prepared a three-switch topology scenario where an LFA was initiated and mitigated.

In Figure 10, it is shown that an attacker injected a fabricated link from switch 3, port 1 into switch 1, port 1. Our discovery algorithm was able to detect that the fabricated link actually had a false HMAC and was non-existent in the link database, hence returning "Detected Fabricated link 00-00-00-00-00-03:1, 00-00-00-00-00-01:1". The fabricated link was detected, and the controller never allowed one to be used. Another link from switch 1, port 2 to switch 2, port 1 was also detected, but unlike the fabricated link, this link met the criteria of a genuine link as per the description in LiFAMA. From this output it can be seen that LiFAMA was successful in detecting fabricated links in SDN and blocking them from being created.

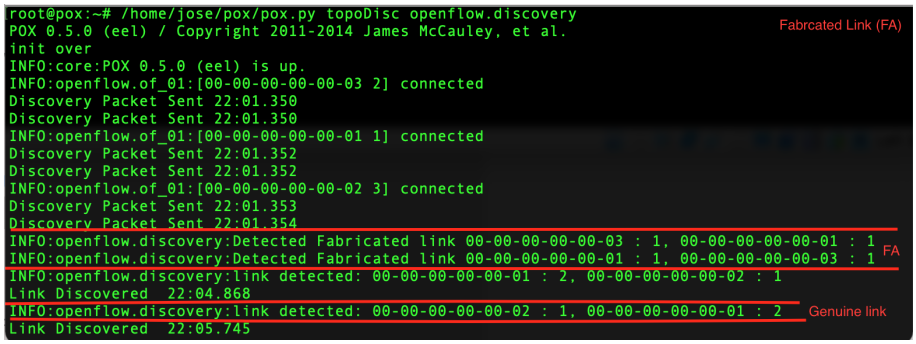

Figure 10. Fabricated link detection.

8. Results

8.1. Performance Metrics

RFC 8456 provides generic methodologies for benchmarking OpenFlow-based SDN controller performance [29]. More parameters to test SDN performance were highlighted by [17], as they carried out an analysis of the discovery process in SDNs. One of the performance benchmarking tests that could be carried out is network topology discovery time, and since the modification is in the topology discovery mechanism, it was imperative to find out how addition of the HMAC and a reference database affected the overall topology discovery time.

The network topology discovery time (TDT) can be obtained by calculating the time difference between the first OFPT-packet-out with an LLDP message received from the controller (T_{m1}) and the last OFPT-packet-in with an LLDP message sent to the controller (T_{mn}) when the comparison is successful, as shown in Equation (3).

$$TDT_1 = T_{mn} - T_{m1} \tag{3}$$

T_{mn} Time at which the last discovery message is received. T_{m1} Time at which the first discovery message was sent out.

The discovery process was repeated for at least three (3) successful attempts and the Average topology discovery time (TDT_{avg}) calculated from Equation (4). A graph of the average topology discovery time against number nodes was plotted for both linear and tree topology discovery times. A graph of discovery time against the number of links was also plotted.

$$TDT_{avg} = (TDT_1 + TDT_2 + TDT_3 + \ldots + TDT_n)/t_n \tag{4}$$

where t_n is the total number of trials and TDT_i is the discovery time after a single trial up to n trials. TDT_{avg} is the average topology discovery time.

CPU utilisation is an important metric used to determine the performance of the SDN. OFDP utilises the CPU to both generate the packet-out and process the packet-in. Addition of the HMAC, which is a cryptographic algorithm, would increase the CPU load as the number of links and switches increases. It is obvious that the number of packets generated depends entirely on the number of switches and the ports in the active ports the switches have. Our LiFAMA, therefore, will naturally exhibit high CPU utilisation because it involves using the HMAC and the database. Our analysis therefore dwells on the discovery time and verification time.

From the graphs in Figures 11–13, it can be noted that the addition of HMAC and a link database has a minimal effect on the discovery time of the network topology, as the difference is about a few milliseconds or seconds for a larger number of nodes. We further note that SPV has a higher discovery time, since at least two packets are required for addition and verification of a single link, hence making it a viable solution to link fabrication attacks.

Figures 11–13 include the standard deviation values, which range from 0.01 to 0.7 s, showing very little variation from the average value of the discovery time for each discovery approach. Discovery time increases with increase in number of nodes and switches. The combination of HMAC and the database opens new avenues for using databases within SDN environments. Based on the computational time required for the discovery, it was also noted that the CPU usage was low, and memory consumption was mainly due to topology creation, since Mininent creates an in-memory topology.

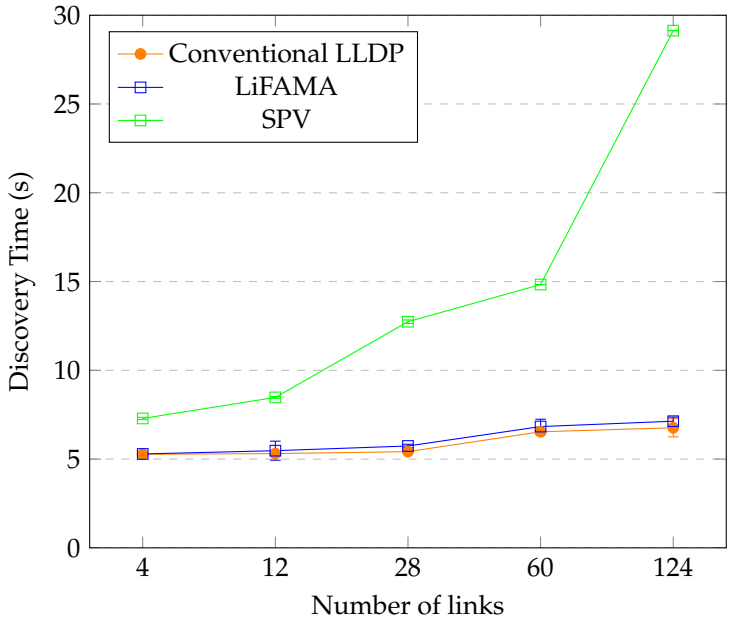

Figure 11. Tree topology performance.

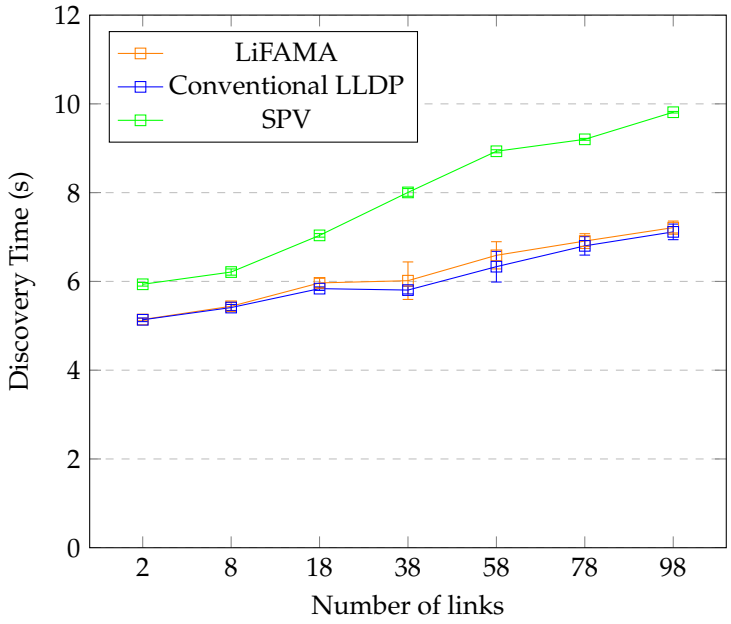

Figure 12. Linear topology performance.

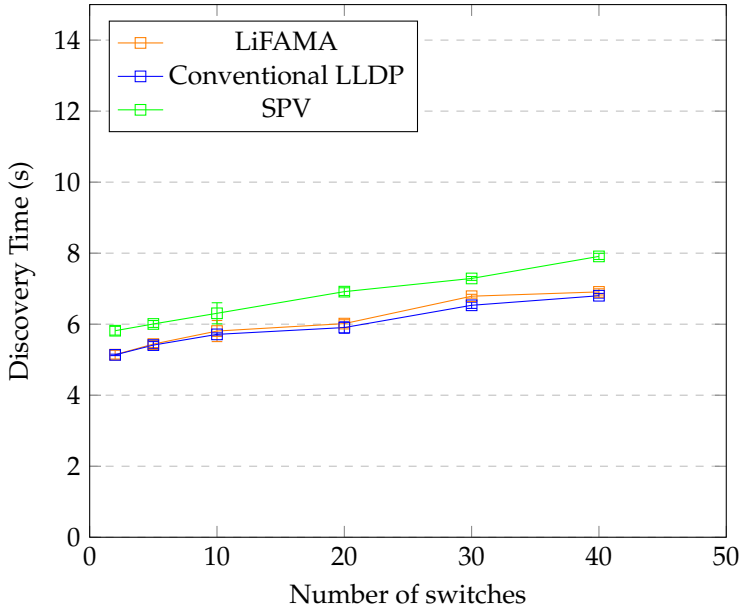

Figure 13. Custom topology performance.

8.2. Discussions

LiFAMA and SPV [8] have the fundamental similarity of both being active algorithms for detecting and mitigating LFA. The major difference between LiFAMA and SPV is that while LiFAMA does not allow creation of a fabricated link before it is discovered, SPV lets the link get created before the probing packet is sent to verify it. For every link created, at least one probing packet is sent out for the link verification. This increases the discovery time for the topology, as more than one packet has to be sent for a particular link—i.e., the packet-out from the controller and at least one probing packet. In the event that the probing packet is dropped, an adversary may be able send traffic through the fabricated link. It was also noted that the probing packet is authenticated using one-way hashing functions that are susceptible to length extension attacks [30]. This difference makes LiFAMA more efficient and faster at detecting fabricated links, though a chance of the probing packet being dropped exists and another packet will have to be resent later. SPV is implemented outside the controller. It can plug into any controller, though this presents a security risk and slows down the controller's efficiency.

From the graphs in Figure 14, it can be noted that the link verification time increases with increase in the number of links and devices. The verification times for all links in the network were noted for the fat tree topology. From the results it was noted that for a low number of devices, LiFAMA and SPV had quite close results, but as the number of devices increased, the verification times for SPV and LiFAMA started differing. LiFAMA recorded a lower verification time than SPV, making it more efficient for verifying large numbers of devices and switches.

In Figure 14, at 40 switches, the verification time for LiFAMA is about 2 s more than that of the conventional LLDP. The difference between the verification times for SPV and conventional LLDP is high because the discovery packet in SPV is sent twice before a link is marked as valid or not. It is also clear that in a simulated environment, the topology is run in memory consuming resources. The controller uses less. This implies that the significant difference when using a simulator will drastically be low when tested on a real platform.

LiFAMA was tested based on in-band control. In Sharma et al. [31], in-band control was tested in the study, and the results strongly recommend in-band control. They proposed a queuing model in which control traffic is served first, preventing competition for network resources with the data traffic. Discovery traffic is control traffic, and in comparison, with LiFAMA, queuing would help reduce the discovery time, especially with large networks. With out-of-band implementation, the control channel is separate from the data traffic.

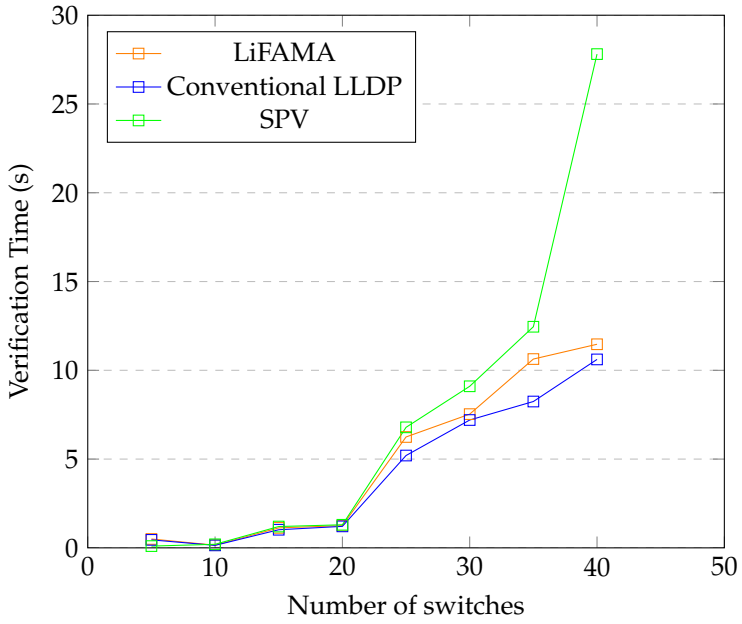

Figure 14. Fat tree topology performance.

8.3. Limitations of LiFAMA

The following are the few limitations LiFAMA presents.

- Management of link database: Introducing a link database as a reference to known links creates an administrative task of updating the database every time there must be link addition or removal. The network administrator in this case is expected to have control of this database to manage inserts, updates and deletions. To improve its security, writes to this database should be independent of the HMAC verified link. Therefore, it would not be a good idea to manage the link database based on links with validated HMACs.
- LiFAMA has only been tested in a simulated environment for proof of concept. Physical environments, however, present more practical and real tests of the true performance of LiFAMA. Topology creation in Mininent is an in-memory process that may use up the random access memory (RAM) of the VM, which might increase the discovery time due to lack of system memory, especially as the topology grows.

9. Conclusions

In this study, an algorithm called LiFAMA for mitigating link fabrication attacks in SDN was proposed and implemented. The mitigation algorithm uses HMAC, a known-links database, and a hashed key authentication mechanism. The results show that for the same network design and configuration, the algorithm increased the discovery time by a few milliseconds for a low number of switches, by about 0.2%, but increased the discovery time with more devices by about 12% for the 1020 links in the tree topology. This is quite negligible compared to the security added by the measure. Though the detection of the fabricated link is performed at packet-in processing, it is almost instant. The proposed measure is an active method that detects and stops the creation of fabricated links. The introduction of the database creates a more dynamic reference and eases implementation of our improvement. In future, we look forward to extending our algorithm to various SDN controllers and carrying out more experiments in other real test beds. We also intend to extend more SDN functionality in a relational database that makes extensions to services easy for developers, since all logic is implemented in the database. We further intend to experiment our LiFAMA in an out-of-band control-based SDN.

Author Contributions: Conceptualization, K.J. and O.S.E.; writing—original draft preparation, K.J. and O.S.E.; writing—review and editing, O.S.E.; P.K., and T.J.O.; supervision, O.S.E. All authors have read and agreed to the published version of the manuscript.

Funding: The APC was funded by Mak-Sida Project 381 at Makerere University, Uganda. This research received no other external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rawat, D.B.; Reddy, S.R. Software defined networking architecture, security and energy efficiency: A survey. *IEEE Commun. Surv. Tutor.* **2016**, *19*, 325–346. [CrossRef]
2. Boussaha, R.; Challal, Y.; Bouabdallah, A. Authenticated network coding for software-defined named data networking. In Proceedings of the 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA), Krakow, Poland, 16–18 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1115–1122.
3. Xiang, S.; Zhu, H.; Xiao, L.; Xie, W. Modeling and verifying TopoGuard in OpenFlow-based software defined networks. In Proceedings of the 2018 International Symposium on Theoretical Aspects of Software Engineering (TASE), Guangzhou, China, 29–31 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 84–91.
4. Wickboldt, J.A.; De Jesus, W.P.; Isolani, P.H.; Both, C.B.; Rochol, J.; Granville, L.Z. Software-defined networking: Management requirements and challenges. *IEEE Commun. Mag.* **2015**, *53*, 278–285. [CrossRef]
5. Smyth, D.; McSweeney, S.; O'Shea, D.; Cionca, V. Detecting link fabrication attacks in software-defined networks. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–8.
6. Dhawan, M.; Poddar, R.; Mahajan, K.; Mann, V. SPHINX: Detecting Security Attacks in Software-Defined Networks. In Proceedings of the Annual Network and Distributed System Security Symposium, San Diego, CA, USA, 8–11 February 2015; Volume 15, pp. 8–11.
7. Hong, S.; Xu, L.; Wang, H.; Gu, G. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures. In Proceedings of the Annual Network and Distributed System Security Symposium, San Diego, CA, USA, 8–11 February 2015; Volume 15, pp. 8–11.
8. Alimohammadifar, A.; Majumdar, S.; Madi, T.; Jarraya, Y.; Pourzandi, M.; Wang, L.; Debbabi, M. Stealthy Probing-Based Verification (SPV): An Active Approach to Defending Software Defined Networks Against Topology Poisoning Attacks. In Proceedings of the 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, 3–7 September 2018.
9. Soltani, S.; Shojafar, M.; Mostafaei, H.; Pooranian, Z.; Tafazolli, R. Link Latency Attack in Software-Defined Networks. In Proceedings of the 17th International Conference on Network and Service Management (CNSM), Izmir, Turkey, 25–29 October 2021.
10. Prasad, A.S.; Koll, D.; Fu, X. On the security of software-defined networks. In Proceedings of the 2015 Fourth European Workshop on Software Defined Networks, Bilbao, Spain, 30 September–2 October 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 105–106.
11. Skowyra, R.; Xu, L.; Gu, G.; Dedhia, V.; Hobson, T.; Okhravi, H.; Landry, J. Effective topology tampering attacks and defenses in software-defined networks. In Proceedings of the 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxembourg, 25–28 June 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 374–385.
12. Kim, H.; Ju, H. Efficient method for inferring a firewall policy. In Proceedings of the 2011 13th Asia-Pacific Network Operations and Management Symposium, Taipei, Taiwan, 21–23 September 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 1–8.
13. Lin, P.C.; Li, P.C. Inferring openflow rules by active probing in software-defined networks. In Proceedings of the 2017 19th International Conference on Advanced Communication Technology (ICACT), Phoenix Park, Pyeong Chang, Korea, 19–22 February 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 415–420.
14. Baidya, S.S.; Hewett, R. Link Discovery Attacks in Software-Defined Networks: Topology Poisoning and Impact Analysis. *J. Commun.* **2020**, *15*, 596–606. [CrossRef]
15. Glaeser, N.; Wang, A. Access control for a database-defined network. In Proceedings of the 2016 IEEE 37th Sarnoff Symposium, Newark, NJ, USA, 19–21 September 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–2.
16. Kaliyamurthy, N.M.; Taterh, S.; Shanmugasundaram, S.; Saxena, A.; Cheikhrouhou, O.; Ben, Elhadj, H. Software-Defined Networking: An Evolving Network Architecture—Programmability and Security Perspective. *Secur. Commun. Netw.* **2021**, *2021*, 9971705. [CrossRef]
17. Wazirali, R.; Ahmad, R.; Alhiyari, S. SDN-OpenFlow Topology Discovery: An Overview of Performance Issues. *Appl. Sci.* **2021**, *11*, 6999. [CrossRef]

18. Yang, L.; Cai, Z.P.; Xu, H. LLMP: exploiting LLDP for latency measurement in software-defined data center networks. *J. Comput. Sci. Technol.* **2018**, *33*, 277–285.
19. Congdon, P. Link Layer Discovery Protocol and MIB. V1. 0 May 20 2002. 2002; pp. 1–20. Available online: https://picture.iczhiku.com/resource/paper/wyIwJyqKAQpuFNNm.pdf (accessed on 7 August 2002).
20. Alharbi, T.; Portmann, M.; Pakzad, F. The (in) security of topology discovery in OpenFlow-based software defined network. *Int. J. Netw. Secur. Its Appl.* **2018**, *10*, 1–16. [CrossRef]
21. Hosoyamada, A.; Iwata, T. On Tight Quantum Security of HMAC and NMAC in the Quantum Random Oracle Model. In Proceedings of the Annual International Cryptology Conference, Virtual Event, 16–20 August 2021; Springer: Cham, Switzerland, 2021; pp. 585–615.
22. Najjar, M. d-HMAC—An Improved HMAC Algorithm. *Int. J. Comput. Sci. Inf. Secur.* **2015**, *13*, 89.
23. Reddy, V.R.; Safwan, M.; Deepamala, N.; Shobha, G.; Premkumar, S.J. Network traffic simulator from real time captured packets. *Int. J. Appl. Eng. Res.* **2017**, *12*, 10134–10137.
24. Krawczyk, H.; Bellare, M.; Canetti, R. Pseudorandom Functions Revisited: The Cascade Construction and its Concrete Security. In Proceedings of the 37th Symposium on Foundations of Computer Science, Burlington, VT, USA, 14–16 October 1996; IEEE: Piscataway, NJ, USA, 1996; pp. 514–523. [CrossRef]
25. Bellare, M.; Canetti, R.; Krawczyk, H. Keying hash functions for message authentication. In Proceedings of the Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 1996 ; Springer: Berlin/Heidelberg, Germany, 1996; pp. 1–15.
26. Leurent, G.; Peyrin, T.; Wang, L. New Generic Attacks Against Hash-based MACs. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, 2–6 December 2013.
27. Kaur, K.; Singh, J.; Ghumman, N.S. Mininet as software defined networking testing platform. In Proceedings of the International Conference on Communication, Computing Systems (ICCCS), Cairns, Queensland, Australia, 10–12 June 2014; pp. 139–142.
28. Noman, H.M.; Jasim, M.N. POX controller and OpenFlow performance evaluation in software defined networks (SDN) using mininet emulator. In *IOP Conference Series: Materials Science and Engineering*; IOP Publishing: Bristol, UK, 2020; Volume 881, p. 012102.
29. Manral, V.; Banks, S.; Monitoring, V.S.S. *Benchmarking Methodology for Software-Defined Networking (SDN) Controller Performance*; Benchmarking: 2018. Available online: https://www.rfc-editor.org/rfc/rfc8456 (accessed on 31 October 2018).
30. Duong, T.; Rizzo, J. Flickr's API Signature Forgery Vulnerability. Available online: https://dl.packetstormsecurity.net/0909-advisories/flickr_api_signature_forgery.pdf (accessed on 28 September 2009).
31. Sharma, S.; Staessens, D.; Colle, D.; Pickavet, M.; Demeester, P. In-band control, queuing, and failure recovery functionalities for openflow. *IEEE Netw.* **2016**, *30*, 106–112. [CrossRef]