Real-Time Link Verification in Software-Defined Networks

Sanaz Soltani, Mohammad Shojafar, *Senior Member, IEEE*, Habib Mostafaei, *Member, IEEE*, and Rahim Tafazolli, *Senior Member, IEEE*

***Abstract*—Software-defined networking (SDN) has been widely adopted in different networks, such as datacenter and service providers. The SDN controller has the entire network view and is responsible for managing it. To obtain such a view of the network, the controller employs link discovery protocols, which are vulnerable to attacks such as *link fabrication attacks (LFAs)*. *TopoGuard* and *TopoGuard+* are two major systems detecting LFAs. This paper introduces a *link latency attack (LLA)* that can bypass the defence mechanism of both systems. LLA can poison the view of the SDN controller from the network topology and causes outages, resulting in poor quality of service (QoS) or quality of experience (QoE). To mitigate this, we develop *two* machine learning-based defence systems, namely machine learning-based link guard (MLLG) and real-time link verification (RLV), to preserve the required defence for LLA. The MLLG works when the network topology rarely updates, while RLV can support frequent updates. Furthermore, RLV trains itself over a link latency dataset (LLD)– including latency data of fabricated and normal links– that is captured from the ongoing packets in the network. It also implements outlier detection techniques to identify a dynamic threshold for link latency. We test both systems on different scenarios using Mininet and show that they achieve reasonable results compared with current defence algorithms. Specifically, RLV presents the highest detection performance (F1-score) to 70% at less than 0.2% false-positive rate. The system also supports the robustness features when the attack rates vary from 3% to 7% in our simulated network.**

***Index Terms*—Software-defined networking (SDN), link fabrication attacks (LFAs), link latency attack (LLA), machine learning, link latency dataset.**

I. Introduction

SOFTWARE-DEFINED network (SDN) simplifies the management of the network devices, such as switches, using an interface with a logically centralised controller. This simplification leads to the widely used application of the SDN in different domain networks, including wired and wireless. In SDN, we can program the forwarding behaviour of devices in the data plane, which gives us more flexibility to implement our desired functionalities [1]. The OpenFlow protocol is one of the most popular realisations of SDN in commercial networking equipment [2].

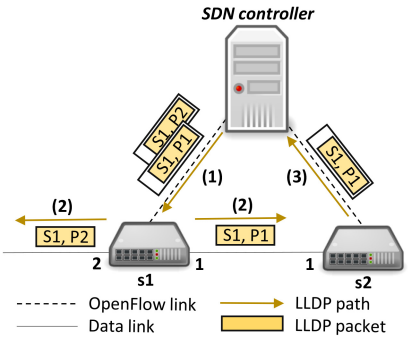


Fig. 1. An example of the link discovery cycle in the OFDP.

In the OpenFlow-based SDN, the controller utilises the OpenFlow discovery protocol (OFDP) [3] to form the logical view of the network. The OFDP involves periodic transmissions of link layer discovery protocol (LLDP) [4] messages from the controller to OpenFlow switches. Fig. 1 presents an example of link discovery process in SDN with two OpenFlow switches, namely, s_1 and s_2. In this process, the controller creates LLDP messages and sends them to the switch s_1 in the data plane (see step (1) in Fig. 1). Each LLDP message includes the data path ID (DPID) of the switch with a Port ID. As LLDP packets are received via `Packet-Out` massages, s_1 distributes them across all its interfaces (see step (2) in Fig. 1). By reaching the message to the destination switch s_2, it encapsulates LLDP as a `Packet-In` message and sends it back toward the controller (see step (3) in Fig. 1). After receiving LLDP, the controller recognises a link between two switches and updates the network topology.

Link discovery procedure can be the target of several security attacks, like topology poisoning attack [5]. Such an attack falsifies the view of the controller to the network topology. Link fabrication attack (LFA) [6] is an example of the topology poisoning attack in which the adversary intends to add a fabricated link between two switches, corrupting the view of the controller [7]. The SDN market value grows every year and will reach 32+ billion USD by 2025, according to the report in [8]. This huge market value confirms the adoption of

Manuscript received 28 October 2022; revised 18 January 2023; accepted 18 January 2023. Date of publication 23 January 2023; date of current version 9 October 2023. This work was funded by University of Surrey, 5GIC & 6GIC (http://www.surrey.ac.uk/ics) and the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A). The associate editor coordinating the review of this article and approving it for publication was J.-H. Cho. *(Corresponding author: Mohammad Shojafar.)*

Sanaz Soltani, Mohammad Shojafar, and Rahim Tafazolli are with 5GIC & 6GIC, Institute for Communication Systems, University of Surrey, GU2 7XH Guildford, U.K. (e-mail: s.soltani@surrey.ac.uk; m.shojafar@surrey.ac.uk; r.tafazolli@surrey.ac.uk).

Habib Mostafaei is with the Department of Mathematics and Computer Science, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands (e-mail: h.mostafaei@tue.nl).

Digital Object Identifier 10.1109/TNSM.2023.3238691

SDN by many enterprises opening the door to several adversaries to attack the network of such businesses, e.g., LFA. We need to analyse LFA since the cost of such an attack is high. According to the report in [9], the average cost of a Cyberattack is 3.86 million USD per incident.

A. Motivations

There are several defence mechanisms to mitigate the risk of LFA, e.g., *TopoGuard* [6], *TopoGuard+* [10], and *SPHINX* [11]. Specifically, *TopoGuard* [6] monitors the LLDP packets to detect LFA. *TopoGuard+* [10] introduces *Port Amnesia* attack that bypasses the defence systems of *TopoGuard* and *SPHINX*. *TopoGuard+* uses the attack to reset the port type of the device – used by *TopoGuard* to detect the link advertisements– and relay the LLDP toward the controller. Motivated by this, we introduce a link latency attack (LLA) that confirms the vulnerability of the *TopoGuard+*. The adversary can use this attack to add a fabricated link into the network and disturbs the view of the controller from the network topology. We report that *TopoGuard* and *TopoGuard+* are unable in preventing LLA.

B. Contributions

We present machine learning-based link guard (MLLG) and real-time link verification (RLV) to preserve the required defence for LLA and LFA. The MLLG works when the network topology rarely updates, while RLV can support frequent updates. Both systems can protect the network topology from link fabrication attacks, including LFA and LLA. In designing MLLG and RLV, we address the following key challenges. Given the diverse link fabrication attacks, detecting LFA and LLA launched through an out-of-band channel is challenging. In such scenarios, the adversary initiates the attacks without manipulating LLDP packets. This makes attack detection a complicated problem. To address this issue, we take the benefits of machine learning (ML) algorithms in implementing outlier detection techniques to identify a dynamic threshold for link latency. To train our proposed ML detection model, we create a comprehensive dataset which captures the link latency values. The network topology can change by adding a set of switches or links, and the RLV adapts itself to the new topology conditions. To do this, we capture the LLDP `Packet-Out` and `Packet-In` messages sent and received by the controller and analysis the link latency values in the presence of the various number of the switches. The MLLG is suitable when there are no changes in the network topology. However, the RLV detects attacks when the network topology updates and leads to the following goals:

- *Scalibility:* When the network size changes, RLV scales and achieves the highest detection rate in large-scaled SDNs.
- *Robustness:* The RLV system is robust for changes in the number of fabricated links, i.e., attack rates.
- *Adaptable:* RLV uses a wide range of ML techniques to detect the LFA and LLA.

This work is an extended version of [12], where we introduced the LLA and developed and implemented MLLG to detect and prevent link fabrication attacks. Differently from [12], in this paper, we add the following key contributions.

- Designing and developing RLV as a real-time, robust and scalable ML-based detection system against LFA and LLA.
- Creating a comprehensive dataset on large-scale network topology in SDN using a real-time time-series database.
- Enhancing our detection model by adding extra features related to `Packet-Out` and `Packet-In` messages.
- Applying weighted ML classifiers to reduce the impact of the unbalanced dataset and achieve an effective model and a more realistic result.
- Evaluating the performance of our systems, RLV and MLLG, in different scenarios implemented in Mininet [13] and Floodlight [14] controller.
- Making our systems open-source and our results repeatable.

The rest of the paper is as follows. In Section II, we present LLA and the weakness of *TopoGuard+* in attack detection using an example. Section III presents our proposed MLLG and RLV systems. The performance evaluation of both systems comes in Section IV. We discuss the capabilities and limitations of LLA and our proposed detection systems in Section V. The related work is presented in Section VI. Finally, Section VII concludes the paper.

II. LLA: Link Latency Attack

In this section, we introduce a new type of link fabrication attack called *link latency attack (LLA)*. To do so, we explain our threat model in Section II-A. After that, we state that the attack occurs in the network in Sections II-B and II-C.

A. Threat Model

Our threat model assumes that the adversary compromises one or more hosts. The adversary can relay LLDP messages through an out-of-band communication channel which is provided in a wire or wireless connection between two compromised hosts. We call this attack LLA. The adversary intends to add a fabricated link between two switches by relaying the LLDP message over the out-of-band communication channel. The adversary exploits the end hosts to inject unwanted traffic, such as ARP, to increase the packet processing time of the switches. Consequently, the response time of the switches to the controller packets, such as probe packets, increases. The adversary can relay LLDP messages among switches using this long response time and insert the corresponding fabricated link between switches. The network performance can be negatively impacted by misleading the traffic. Consequently, this traffic detouring results in poor quality of service (QoS) or quality of experience (QoE), to name a few [15], [16].

We now describe how the adversary can bypass through LLA the defence mechanism of *TopoGuard+*. *TopoGuard+* includes a link latency inspector (LLI) module to monitor link latency values in the network. LLI can detect fabricated links by measuring the latency of links caused by out-of-band channels when transmitting LLDP packets. The LLI periodically

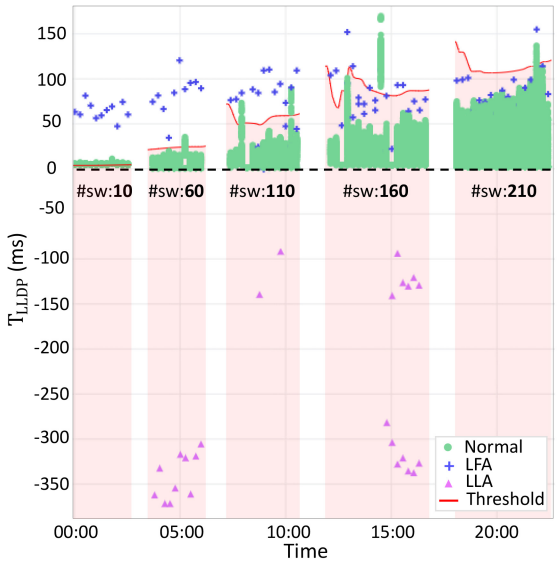


Fig. 2. An example of the weakness of *TopoGuard+* in detecting LLA.

issues probe packets to measure and capture the round trip time (RTT) between the switches and the controller. Every switch responds to the received LLDP packets. Consider two switches, called s_1 and s_2, connected to the controller. The link latency of the probe packets sent to s_1 and s_2 is also given by T_{p_1} and T_{p_2}, respectively. The LLI calculates link latency T_l between switches based on eq. (1),

$$T_l = T_{LLDP} - T_{p_1} - T_{p_2}, \tag{1}$$

where T_{LLDP} represents the propagation latency of the LLDP message. The controller adds a timestamp to the sent LLDP packet to the switches and keeps track of the difference upon receiving the packet. Furthermore, LLI calculates a link latency threshold T_h based on the values of inter-switch latency T_l for previous LLDPs, as represented in eq. (2),

$$T_h = q_3 + 3 * (q_3 - q_1), \tag{2}$$

where q_1 and q_3 indicate the lower and upper quartiles of latencies, respectively. LLI verifies the link's validity by comparing latency T_l and threshold T_h and raises a security alarm in case of suspicious delay, i.e., $T_l > T_h$.

Running Example: The adversary can bypass the *TopoGuard+* by launching LLA. We configure a time series database using InfluxDB [17] to store and visualise link latency data. Fig. 2 is an example view from InfluxDB, which presents the weakness of *TopoGuard+* in detecting LLA. Specifically, the green points show latency values for normal links, and the red line indicates the calculated threshold T_h using eq. (2). *TopoGuard+* can correctly detect all blue '+' marks above the red line as attack points. However, it fails to detect abnormal link latency values located in the highlighted area, which contains LFA '+' marks below the red line and all LLA purple triangle points.

The proposed LLA consists of two phases, namely, *overload phase* and *relay phase*. In the former phase, the adversary injects a huge amount of ARP traffic into the network, while in the latter phase, it relays the received LLDP packets from the switches via the out-of-band channel. The adversary leverages

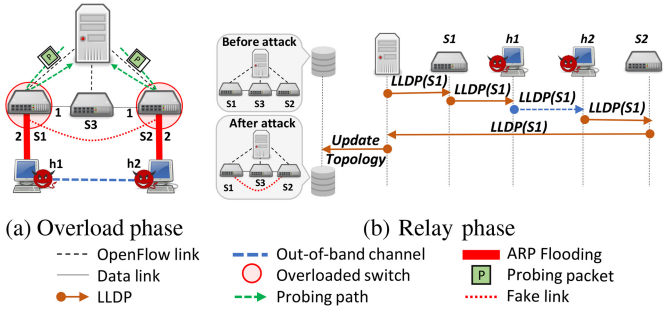


Fig. 3. The considered schematic of LLA. Fig. 3(a) includes compromised hosts send huge ARP traffic toward s_1 and s_2 to increase the RTT of probing packets. Fig. 3(b) explains that compromised hosts relay the LLDP packet through the out-of-band channel.

at least two compromised hosts for this purpose and frequently switches between two phases based on the LLDP propagation interval. To measure the LLDP packet interval, the adversary keeps the time difference between two consecutive packets.

The compromised hosts take two different roles during the attack; *flooder* and *listener*. In the *overload phase*, they play the role of *flooder* and send ARP floods to the switches. In the *relay phase*, as a *listener* role, they both listen to LLDP packets and relay them toward each other and the peer switches. Fig. 3 shows how the attack takes place in our scenario.

B. Overload Phase

During this phase, *flooder* hosts send ARP flooding traffic to switches s_1 and s_2 (see Fig. 3(a)). This traffic significantly increases the number of table-miss entries on the switches and directs a huge number of *Packet_In* messages toward the controller. Handing such an amount of packets results in increasing resource usage of the Open vSwitch (OVS) daemon from the switches. Consequently, the daemon pushes the incoming packets into the queues to be processed later. This results in either growing the probing packet's RTT or even dropping the packets due to the congestion on the ingress port of the switch. The increment in the RTT of the probe packets is enough for the adversary to launch the *relay phase*.

C. Relay Phase

In this phase, both hosts, i.e., h_1 and h_2, stop flooding ARP packets and change their role to *listener* for the incoming LLDP packets. When the controller issues LLDP packets, the adversary changes the attack phase from the *overload phase* to *relay phase*. Fig. 3(b) shows that upon receiving the LLDP packet in the *relay phase* by host h_1, it forwards this packet to host h_2 through a dedicated link. Host h_2 does the same task and forwards the LLDP packet to switch s_2. At this point, in the view of the switch s_2, this is a new LLDP packet from a switch, and it has to forward this packet to the controller.

The controller receives the LLDP response packet, finds a change in the network topology, and updates it. To do so, it performs a check on the threshold and received LLDP packet latency using eq. (1) and eq. (2). Here, the values of T_{p_1} and T_{p_2} are high compared with the normal LLDP packets since they experience high latency in the *overload phase*.

However, by applying eq. (1), the latency of the extra link between switches s_1 and s_2, i.e., T_l, stays in the valid range from the controller point of view, i.e., $T_l \leq T_h$. Even in some cases, the calculation shows a negative value for T_l. Hence, the LLI module in *TopoGuard+* fails to detect the LLA, and finally, the controller updates its view of the network topology by adding an extra link between switches s_1 and s_2.

By deeper investigation through the *TopoGuard+* source code, we realised that the implementation strategy used in this framework for measuring the control link latency leads our proposed LLA more cost-efficient for the adversary. By initiating the first *overload phase*, control link latency, i.e., T_{p_1} and T_{p_2}, increases to a high value. However, the abnormal observation is that *TopoGuard+* freezes on this value and never decreases it even after stopping the *overload phase*. It means that the adversary does not need to sustain or repeat the *overload phase* to keep the latency values high. This vulnerability in *TopoGuard+* implementation is because the controller does not initiate a new probing packet toward the switch without receiving the answer for the previous one.

Example Scenario: LLA can be applied in real-world attack scenarios such as SDN-based vehicular network [18]. In such scenarios, OpenFlow switches take the role of roadside units (RSUs). Hosts could connect to the RSUs and can be surveillance computers, roadside control platforms and edge servers. These hosts communicate with each other through wired or wireless channels. LLA creates a fabricated link that misleads the shortest path decision between two RSUs. It detours the traffic to a different path for the vehicles.

III. Countermeasures

This section describes our two proposed countermeasure systems. We first explain the architecture of MLLG in Section III-A. Then, we introduce RLV architecture in Section III-B and describe how RLV can protect network inter-switch links in real time against the LLA and LFA threats.

A. MLLG: Machine-Learning Link Guard

MLLG leverages ML techniques to protect the network from the LFA and LLA. This system classifies LLDP packets through ML models using a predefined dataset that contains different link fabrication attacks. The controller of MLLG verifies the received LLDP packets by checking their corresponding link latencies with the issued time. This verification results in either dropping the packets or updating its topology graph. We refer to [12] for the internal architecture of MLLG.

MLLG tracks received LLDP packets using a dataset. To do so, it checks `Packet-In` messages and records their propagation time which is the input for ML classifiers. Then, packets are classified using well-known classification algorithms, including random forest (RF), multi-layer perceptron (MLP), K-nearest neighbours (K-NN), support vector machine (SVM), logistic regression (LR), and Naive Bayes (NB). We train the classifiers using 80% of packets in the dataset and the remaining ones for testing. We refer to [12] for more details on the system and its implementation.

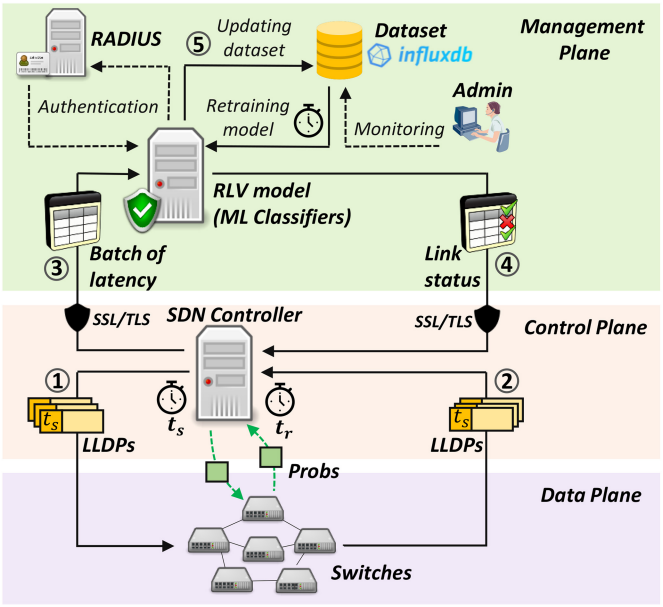


Fig. 4. The general architecture of RLV system.

B. RLV: Real-Time Link Verification

Our proposed system, called real-time link verification (RLV), uses ML techniques to detect LLA and LFA in SDN. In what follows, a detailed description of RLV architecture and ML model configuration and implementation is given.

1) RLV System Architecture: Fig. 4 illustrates the hierarchical architecture of the RLV system, including three planes: i) data plane, ii) control plane, and iii) management plane. The overall workflow of the system works in the following sequence. ① The SDN controller periodically generates LLDP and probe packets in a specific time interval. Then, it forwards the generated packets to the data plane switches. ② Upon receiving the LLDP packet by the switches, they issue a response to each LLDP packet and send it back to the controller. The controller collects LLDP response packets, extracts required metrics, and vectorizes them in batch form. ③ The controller makes a batch of latency values provided by switches–when it needs to verify the validity of the values with their associated delays using RLV. Using the batch technique reduces the communications overhead of the system. ④ RLV analyses each vector based on the ML classification model and forwards its decision toward the controller. The controller either drops the LLDP packet or updates the topology database using the outcome of RLV. ⑤ The classification results and new LLDP data are stored in the dataset. We use InfluxDB to store our dataset. Moreover, human analysts play an important role in monitoring the accuracy of imported data to the dataset.

We deploy RLV on a separate server equipped with high processing capacity to minimise the processing time required for link verification and model regeneration. However, this deployment might raise several security concerns, such as spoofing or information leakage risks. To mitigate these risks, we implement a mutual authentication of the controller to the RLV server and the RLV server to the controller. To this end, we utilise a certificate-based mechanism verified by a remote

access dial-in user service (RADIUS) server [19]. In addition, we encrypt the communication channel between the controller and the RLV server using SSL/TLS.

2) Link Latency Analysis: The main focus of the RLV is on link latency, which is calculated based on LLDP messages and T_{LLDP}. Some network situations might impact LLDP propagation time, i.e., T_{LLDP} and consequently increase the link latency. To observe the effects of network size on LLDP processing time, we examine T_{LLDP} in the sense of different network environments. To this end, we first formulate how network size can impact the CPU usage of the controller and T_{LLDP}. Then, we investigate the impact of network topology and background traffic on T_{LLDP}.

Network Size: Assume an SDN graph $\mathcal{G}(\mathrm{V}, \mathrm{E})$ with V switches and E inter-switch links, where each switch has maximum *M* ports. Let $\mathcal{P}^t_{out}$ and $\mathcal{P}^t_{in}$ be the number of LLDP `Packet-Out` and LLDP `Packet-In` messages at *t*th discovery cycle, respectively (see Fig. 1). Using Eq. (3), we calculate $\mathcal{N}^t_{\mathcal{G}}$ as the total number of LLDP messages the controller needs to process.

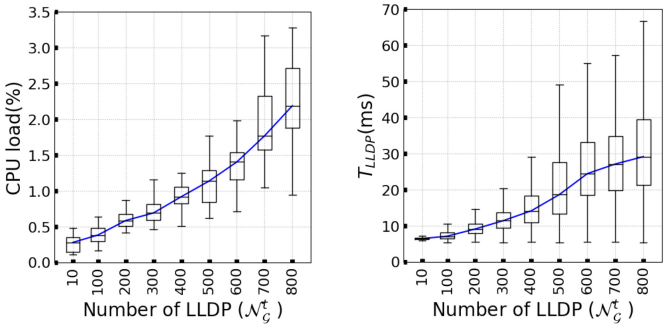
$$\mathcal{N}^t_{\mathcal{G}} = \mathcal{P}^t_{out} + \mathcal{P}^t_{in} = \sum_{i=1}^{V}\sum_{j=1}^{M} p_{ij} + 2E, \tag{3}$$

where

$$p_{ij} = \begin{cases} 1, \text{ If port } j \text{ is active on switch } i, \\ 0, \text{ else.} \end{cases} \tag{4a}$$

Eq. (3) shows that any update in network size, i.e., the number of switches (*V*), active ports (a subset of *M*), and links (*E*), can change $\mathcal{N}^t_{\mathcal{G}}$. Fig. 5 illustrates the impact of changing $\mathcal{N}^t_{\mathcal{G}}$ on the CPU load of the controller and T_{LLDP}. Each experience is repeated 40 times, and the average is computed with 95% confidence intervals. The result in Fig. 5(a) shows that as $\mathcal{N}^t_{\mathcal{G}}$ increases, CPU load also increases. For example, for $\mathcal{N}^t_{\mathcal{G}} = 800$, CPU load grows $\approx$ 8x compared with $\mathcal{N}^t_{\mathcal{G}} = 10$. The reason is that the controller must process more LLDP packets in one discovery cycle. Moreover, the higher CPU utilization can impact the average T_{LLDP} since the controller needs more time to process the incoming LLDP packets and calculate T_{LLDP} based on extracted timestamp. As shown in Fig. 5b, T_{LLDP} increases with $\mathcal{N}^t_{\mathcal{G}}$ where it reaches average $\approx 30ms$ for $\mathcal{N}^t_{\mathcal{G}} = 800$.

Network Topology: According to Eq. (3) and Fig. 5, we realised that T_{LLDP} could be influenced by the network size regardless of how switches are connected to each other. Fig. 6 shows the Cumulative Distribution Function (CDF) of average T_{LLDP} for two network typology types with different network sizes. We consider a linear topology, commonly studied in previous works on LFA [6], [20], and a fat-tree topology one of the most popular network typologies in large data centres [21]. The results show that despite using two different types of network typologies, the T_{LLDP} intervals are approximately the same until the typologies have the same network size or $\mathcal{N}^t_{\mathcal{G}}$. For example, T_{LLDP} intervals for fat-tree with $k = 4$ and linear with $n = 30$, are same and less than $\approx 40ms$ for nearly 100% LLDP messages, when we have $\mathcal{N}^t_{\mathcal{G}} = 145$. By updating the network typology, for example from $k = 4$ to $k = 6$ in fat-tree, we observed that the T_{LLDP} interval increases because of $\mathcal{N}^t_{\mathcal{G}}$ growth to 500.



(a) CPU load (b) T_{LLDP}

Fig. 5. Impact of increasing the number of LLDP messages, $\mathcal{N}^t_{\mathcal{G}}$, on average CPU load (a) and T_{LLDP} (b).

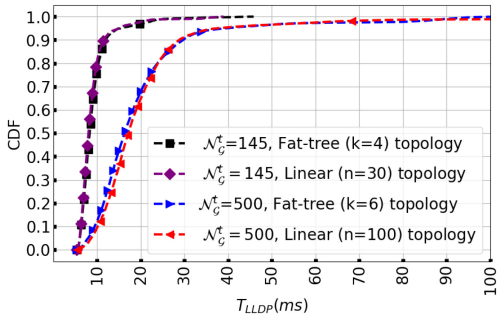


Fig. 6. CDF of T_{LLDP} for four different network topologies.

Background Traffic: We injected background traffic to see the impact of it on T_{LLDP}. We use the traffic generator [22] to generate traffic flows according to Websearch workload [23]. The generated traffic has a Poisson Distribution. The inter-switch link capacity of the links is 1*Gbps*, and we use 50% of the link load for the background traffic. Then, we captured the LLDP packet and calculated T_{LLDP} values. Fig. 7 illustrates the CDF of average T_{LLDP}. We only show some representative results for two network sizes, including 40 and 140 switches. The results show that for each network size, T_{LLDP} presents approximately the same patterns with and without background traffic. Some minor difference is observed in the network with the high number of switches. For example, for switch=140 and without background traffic, nearly 100% of T_{LLDP} are less than 90 *ms*. By imposing the background traffic, we observed that only less than nearly 1% of LLDP has latency higher than 90 *ms*. The reason is that in large-scale networks, the background traffic can slightly impact the switches' buffer and LLDP queuing time in the controller, which could increase T_{LLDP} in less than 1% cases.

3) Dataset Preparation: In order to train our proposed detection method, we need to maintain a large set of labelled network datasets with link properties. The dataset should contain data of fabricated and normal links which are extracted from LLDP packets. The link latency dataset should be representative of real-world network behaviours, which typically have more data for the normal links and less data for the fabricated ones. We created a real and large-scale link latency

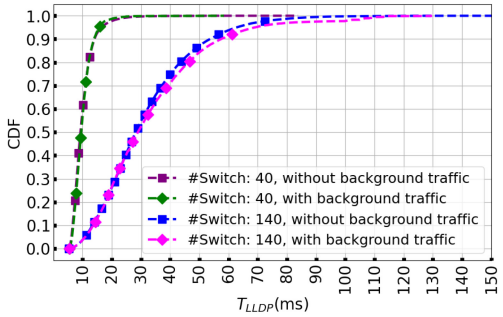


Fig. 7. CDF of T_{LLDP}, without and with background traffic (500 *Mbps* with Poisson distribution) for the different number of switches.

TABLE I
SUMMARY DESCRIPTION OF LLD

Samples	1024885
Attribute	6
Class type	(Normal link, Fake link)
Class percentage per type	(98%, 2%)
Imbalance ratio	(50,1)
Imbalance degree	High

dataset (LLD) that contains 1,024,885 LLDP packets in total. The dataset contains information from two different classes, where the normal links belong to class 0, and fabricated links are part of class 1. We considered $\approx 2\%$ of LLDP for class 1 and $\approx 98\%$ for class 0. Following this approach to maintain the dataset, the imbalance ratio (IR) in LLD is 50 : 1. Therefore, it can be considered a highly imbalanced dataset [24]. Table I summarises the data employed in LLD, including the number of samples, number of attributes, name of minority and majority classes, class distribution, IR and imbalance degree.

We modify the Floodlight controller to extract the following timing features in collecting LLDP packets. This procedure happens on each link discovery cycle of the controller, and we record these features in a nanosecond time scale.

- T_{LLDP}*:* Upon arriving a `Packet-In` message, we capture LLDP propagation time (T_{LLDP}) by taking the difference between *sendTime* and *receiveTime* parameters of each LLDP packet.
- T_{p_1} *and* T_{p_2}*:* We capture the round trip time between the controller and switches to measure the control link latency of the ingress (T_{p_1}) and egress (T_{p_2}) ports.
- N_{PO}*,* N_{PI} *and* N_{SW}*:* We count the number of LLDP `Packet-Out` and `Packet-In` messages sent/received by the controller to update the values of N_{PO} and N_{PI}. As discussed in Section III-B2, the network size updates can be tracked by accounting for these messages. In addition, we capture the number of switches in N_{SW} based on the topology information database.

The LLD provides the following two major capabilities.

- *Diversity:* To enrich the diversity of the link latency value in dataset, a set of network topologies is considered by varying the number of switches from 20 to 200. In addition, we launch different link fabrication attack scenarios categorised into two major groups: LFA and LLA. We also launch several attacks imposed at the same time to capture the link latency value with a different number of attacks.
- *Scale:* We run the controller for nearly three days to collect 1,024,885 LLDP packets including 10,095 (1%) for the LFA, 10,095 (1%) for the LLA, and 1,004,695 (98%) for the normal inter-switch links.

TABLE II
DATA DISTRIBUTION IN LINK LATENCY DATASET

# Switch	Duration (hour)	# Total LLDP	Class 0	Class 1	
			# Normal (98%)	# LFA (1%)	# LLA (1%)
20	2	24695	24209	243	243
40	2	43263	42407	428	428
60	2	58163	57015	574	574
80	2	77712	76168	772	772
100	2	115960	113672	1144	1144
120	2	118673	116335	1169	1169
140	2	133124	130502	1311	1311
160	2	145156	142300	1428	1428
180	2	147635	144735	1450	1450
200	2	160504	157352	1576	1576
Total		1024885	1004695	10095	10095

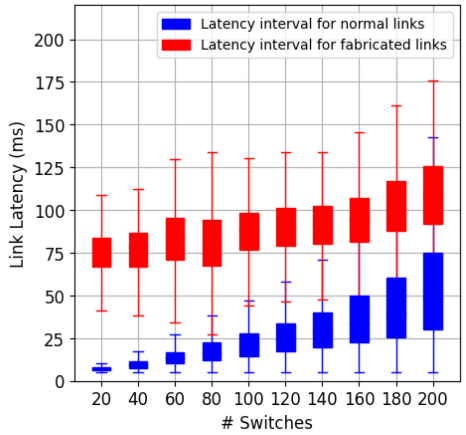


Fig. 8. Impact of adding new switches to the network on the link latency distribution.

Table II and Fig. 8 present the detail of captured data and the latency interval distribution in LLD. In our experiments, we have separated the dataset into 80% for training and 20% for the testing phase.

4) ML Training Algorithms: We define an imbalanced binary decision problem [25] where the trained ML algorithm identifies a link as either fabricated (class 1) or normal (class 0) while the minority class (fabricated link) in the dataset outnumbers the majority class (normal link). A highly imbalanced dataset could significantly impact the performance of ML algorithms. The standard ML classifiers consider the same weights for two classes. As a result, the model ignores the minority class and biases toward the majority class. To do this, we can implement both oversampling and undersampling strategies. Oversampling strategy injects artificial data into the dataset to oversample the minority class. At the same time, in the undersampling one, the real data is removed from the dataset to undersample the majority class. However, applying these approaches to create a balanced dataset threatens the trustworthiness of the original dataset and causes losing

real data, which could lead to mistraining the model. Another approach to coping with the imbalanced dataset problem is the weighted ML classifier, which assigns specific weights to different classes based on their proportions. In this method, the importance of the minority class increases by giving it a high weight, while a low weight is assigned to the majority class. In our scenario, we train our ML model using several weighted ML classifiers to increase the impact of the minority class (fabricated link) and avoid the bias phenomenon model for the majority class (normal link) due to the significant proportion.

Classification Models: In general, ML-based classification models can be categorised into single and ensemble models [26]. For a fair comparison, we select KNN and SVM algorithms as single classifiers and RF and MLP algorithms as ensemble classifiers. Moreover, these four algorithms were the top-performing classifiers in our previous work in detecting fabricated links [12]. They also proved their efficiency and effectiveness in many pieces of research in network security [27]. In the following section, we describe the capabilities and advantages of each algorithm.

- *K-Nearest Neighbour (KNN):* The KNN algorithm works based on distance functions such as Euclidean distance. To classify a new data sample, the KNN identifies the *K* nearest neighbours and then computes the average values of such neighbours. Compared with other classifiers, the KNN is simple, easy to implement and has the lowest computational complexities of $O(n \log K)$ [27] where n is the number of training instances.
- *Support Vector Machines (SVM):* The SVM algorithm seeks to find the best possible hyperplane which separates data instances between two classes. Using kernel functions, the SVM improves classification performance through nonlinear solutions. The complexity of SVM is $O(n^2)$ [27], which is slightly higher than the other classifiers. However, the SVM is proved to be useful in several applications, specifically when the data are not linearly separable [28].
- *Random Forest (RF):* The RF classifier relies on ensemble learning, which often performs better than other classifiers. In this learning method, RF first constructs t independent decision trees using different samples of the training data and a random subset of attributes. Then an average outcome is calculated based on estimations from such trees. RF attracts many researchers' attention due to its low computational complexity and high accuracy. Due to the multi-threading capability of RF, the complexity of this classifier is $O((tmn \log n)/h)$, where m is the number of attributes, and h is the maximum number of threads [27].
- *Multi-Layer Perceptron (MLP):* MLP is the most popular model in artificial neural networks (ANN). It uses a back-propagation training algorithm to improve the accuracy of predictions. It consists of a large number of interconnected nodes, which is called *neuron*. MLP is commonly implemented in two hidden layers with a feed-forwarding structure. The computational cost for training MLP is $O(mnu)$, where u is the number of neurons.

Hyperparameter Optimisation: We use the grid search (GS) algorithm [29] as a hyper-parameter optimisation technique. GS algorithm tunes all possible combinations of parameters and identifies the optimal set. The accuracy measure is generally considered as the objective function to be maximised. However, in a highly imbalanced dataset, the accuracy can be misleading. In this work, we consider F_1-score (F_1) as the optimisation objective. For each model, the best parameter value maximises the F_1 in the training dataset, which is computed as follows.

$$\max_{l} F_1(l) = \max_{l} \frac{2 \times TP_l}{2 \times TP_l + FP_l + FN_l} \tag{5}$$

where l identifies the set of parameters, true positive (TP) indicates the number of attack attempts correctly classified as fabricated links, false positive (FP) denotes the number of normal links misclassified as fabricated links, and false negative (FN) is the number of attack attempts misclassified as a normal link.

To simplify the process, the parameters that have a less significant impact on the model are fixed, and only the most significant parameters are tuned. In addition, when the parameter space is increased beyond a certain number, there is almost no performance gain; instead, the model is more complex, requiring a longer training time. Based on previous literature and our own experience, we select the following most important parameters and relevant parameter space for each algorithm to be tuned.

- *KNN:* We consider different values for *K*, ranging between 1 and 100 with step of 10. It is observed that the best value for *K* is 60.
- *SVM:* We consider different SVM kernels, including linear, polynomial, radial basis function (RBF), and sigmoid. As expected, RBF-based SVM performs relatively better because our dataset is not linearly separable. The regularization parameter denoted by *C* allows controlling the penalty assigned to misclassified samples. We tune *C* with various values, ranging between 0 to 5000, with a step of 500. For the considered dataset, $C = 1000$ performs the best.
- *RF:* We perform an analysis with RF using different values for the number of trees and the maximum depth, ranging between 1 to 20. The splitting criterion including *Gini Index* and *Cross-entropy* is also tuned. The classifier shows the best results of 5 for estimators with a maximum depth of 6 and *Cross-entropy* as the splitting criterion.
- *MLP:* We tune the number of hidden layers between 1 and 10. The learning rate is also tuned based on the set {0.005, 0.001, 0.0005, 0.0001}. We also vary the activation function used in each neuron considering set {*Identity, Sigmoidal, Tanh, Relu*}. The best parameters are obtained with 2 hidden layers, learning rate=0.001, and ReLu as an activation function.

In summary, the optimised configurations are shown in Table III, which are used throughout the rest of the paper.

5) RLV Implementation: We implement RLV in the Floodlight controller. Our system consists of two main procedures to verify the validity of the links: (1) collecting the

Algorithm 1 Real-Time Link Verification (RLV)

1: **procedure** LLDPLISTENER
2: **while** RecievedLLDP==*ture* **do** ▷ Continue until all LLDP packets are received
3: t_r =getCurrentTime() ▷ Capturing the current time as received time of LLDP
4: $t_s = \text{getTimestamp}(lldp)$ ▷ Extracting timestamp of LLDP
5: $T_{lldp} = t_r - t_s$
6: $s_I = \text{getIngressId}(lldp)$ ▷ Extracting DPID of egress switch
7: $s_E = \text{getEgressId}(lldp)$ ▷ Extracting DPID of ingress switch
8: $T_{p_1} = \text{getProb}(s_I)$ ▷ Calculating link latency between the controller and ingress switch
9: $T_{p_2} = \text{getProb}(s_E)$ ▷ Calculating link latency between the controller and egress switch
10: $lldpList.add(T_{lldp}, T_{p_1}, T_{p_2})$ ▷ Storing received LLDP information into a list
11: **end while**
12: sw =getSwitchNum()
13: pkt_{in} =getPktInNum()
14: pkt_{out} =getPktOutNum()
15: LINKSVERIFICATION($lldpList$, sw, pkt_{in}, pkt_{out})
16: **end procedure**
17:
18: **Input:** *List of LLDP packets* ($lldpList$), *Number of switches* (sw), *Number of LLDP* `Packet-In` (pkt_{in}), *Number of LLDP* `Packet-Out` (pkt_{out})
19: **procedure** LINKSVERIFICATION(*lldpList*, sw, pkt_{in}, pkt_{out})
20: $batchQuery = GenerateBatchQuery(lldpList, sw, pkt_{in}, pkt_{out})$
21: $model = getModel()$ ▷ Loading pre-trained ML model
22: $linkStatus = getPredictions(model, batchQuery)$ ▷ querying the model for links validation
23: $lldpList.setLinkStatus(linkStates)$ ▷ Relating link status to each LLDP packet in the list
24: **for each** $lldp \in lldpList$ **do**
25: **if** $lldp.LinkStatus == 1$ **then** ▷ Detecting fabricated link
26: $raiseSecurityAlarm(lldp)$
27: **else**
28: $updateTopology(lldp)$ ▷ Updating topology graph only if the link is valid
29: **end if**
30: **end for**
31: $updateDataset(lldpList, sw, pkt_{in}, pkt_{out}, linkStatus)$
32: **end procedure**

TABLE III
ML ALGORITHMS AND OPTIMAL HYPERPARAMETERS

Alg.	Parameter	Optimal value	Parameter space
KNN	No. neighbours (K)	60	[1,100]
SVM	Kernel function	RBF	{Linear, polynomial, RBF, sigmoid}
	Regularization (C)	1000	[0,5000]
RF	No. trees	5	[1,20]
	Maximum depth	6	[1,20]
	Splitting criterion	Entropy	{Gini, Entropy}
MLP	No. hidden layer	2	[1,10]
	Learning rate	0.001	{0.005, 0.001, 0.0005, 0.0001}
	Activation function	Relu	{Identity, Sigmoidal, Tanh, Relu}

LLDP packets and (2) querying the validity of the links. Algorithm 1 presents the pseudo-code of our system. The RLV system works when the controller waits for the response of issued LLDP packets. At this moment, the LLDPLISTENER procedure is active and collects information regarding the received packets. We now explain this procedure in more detail.

Lines 1-16 (Extracting LLDP Information): In each link discovery cycle, the controller extracts all required information from the received LLDP packets and collects them into a list. First, it calculates the T_{LLDP} using the timestamp field. Then, it measures the link latency between switches and the controller, i.e., T_{p_1} and T_{p_2}. Since the network size can be changed between two LLDP intervals, the controller also measures the number of switches, the number of generated LLDP `Packet-Out` messages, and the number of incoming LLDP `Packet-In` messages in each interval. Now, the controller can verify the validity of the link. The controller passes the values of these parameters to the LINKSVERIFICATION module and waits for the verification results.

Lines 18-32 (Query ML Model for Link Verification): Instead of sending separate link verification queries toward the defence model, the controller creates a batch of queries using all LLDP parameters. This could help to reduce the overall query response time significantly. For each LLDP, if the model detects the fabricated link, it first drops the LLDP, then informs the network administrator by raising a major security alarm and sets the status of the link to `FakeLink`. Otherwise, it updates the topology database of the network and sets the status of the link to `ValidLink`.

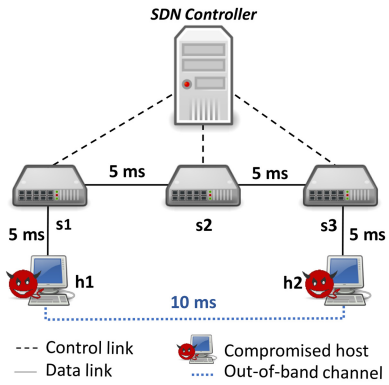
ML models can be adapted to network changes by retraining the models according to the most recent information.

6) RLV Overhead: As we discussed, the SDN controller triggers the link discovery process in a specific time interval. For example, the link discovery interval is 15 seconds in the Floodlight controller. The computation time of the link discovery process at *t*th discovery cycle is equal to eq. (6),

$$L(t) = N_t.\sigma_t \tag{6}$$

where N_t is the total number of LLDP packets traversing the switches in cycle t and σ_t indicates the average processing time for each LLDP in cycle t.

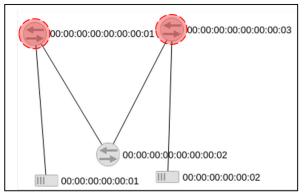
By implementing the RLV defence module, the controller still keeps sending and receiving the LLDP packets with the



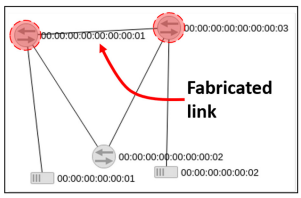
(a) The simulated network topology.

```
1 : 20:08:58.997 INFO  link latency is okay. latency:5ms, threashold:12ms
2 : 20:08:58.998 INFO  link latency is okay. latency:7ms, threashold:15ms
3 : 20:09:14.023 INFO  link latency is okay. latency:6ms, threashold:15ms
4 : 20:09:14.027 INFO  Link latency between controller and switch 3 :1795us
5 : 20:09:14.027 INFO  Link latency between controller and switch 1 :2054us
6 : 20:09:14.053 ERROR  Deteced suspicious link discovery: an abnormal link latency
during LLDP propagation
7 : 20:09:14.053 ERROR  link latency is abnormal. latency:33ms, threashold:15ms
8 : 20:09:29.110 INFO  link latency is okay. latency:8ms, threashold:15ms
9 : 20:09:29.110 INFO  Link latency between controller and switch 1 :1342us
10: 20:09:29.111 INFO  link latency is okay. latency:7ms, threashold:15ms
11: 20:09:29.111 INFO  link latency is okay. latency:7ms, threashold:17ms
12: 20:09:29.112 INFO  Link latency between controller and switch 3 :1261us
13: 20:09:29.112 INFO  link latency is okay. latency:8ms, threashold:27ms
14: 20:09:29.572 INFO  Packet in rate is 18181, enable throttling on OFSwitchBase
15: 20:09:29.662 INFO  Packet in rate is 10204, enable throttling on OFSwitchBase
16: 20:09:30.583 INFO  Packet in rate is 1058, enable throttling on OFSwitchBase
17: 20:09:44.121 INFO  Link latency between controller and switch 3 :90831us
18: 20:09:44.122 INFO  Link latency between controller and switch 1 :92555us
19: 20:09:59.220 INFO  Inter-switch link detected: Link [src=00:...:01 outPort=1,
dst=00:...:03, inPort=1]
20: 20:09:59.220 WARN  Link added: Link [src=00:...:01 outPort=1, dst=00:...:03,
inPort=1]
21: 20:09:59.649 INFO  updateAttachmentPoint: ap [AttachmentPoint [sw=3, port=1]]
22: 20:09:59.682 INFO  updateAttachmentPoint: ap [AttachmentPoint [sw=1, port=1]]
```

(b) Floodlight logs before and after launching LLA.



(c) Before Link Latency Attack



(d) After Link Latency Attack

Fig. 9. An example of LLA using a simple topology on the Floodlight controller.

previous computation time. The only difference is LLDP processing time because each LLDP also needs to be sent toward RLV for security purposes. Hence, the computation time of the link discovery process in the case of using RLV is equal to eq. (7),

$$\hat{L}(t) = N_t.\sigma_t + \rho_t + \gamma_t \tag{7}$$

where ρ_t indicates the processing time to query the link status prediction from the model (line 22), and γ_t is the processing overhead to update the dataset with the LLDP batch (line 31). As a result, implementing RLV as defence system in SDN controller imposes the extra overhead of $\rho_t + \gamma_t$.

IV. Performance Evaluation

In this section, we investigate the security and network performance for the LLA attack and RLV defence system. First, we present the network setup to run LLA in Section IV-A and show the accuracy of the LLA attack in Section IV-B. The performance of our proposed ML models is evaluated in Section IV-C. We also discuss the capability of RLV in the dynamic network size.

A. System Setup

We run our experiments on a physical server equipped with an Intel Xeon CPU E5-2667 3.3GH VM with 32 CPU cores and 190 GB RAM running Ubuntu server 18.04. We use Mininet for the simulation and build the network topology of Fig. 9(a). All the links connecting the hosts to the switches have 5 milliseconds (ms) of delay, including the inter-switch links.

We state that extending the size of the network results in more LLDP packets being sent and received by the controller, which can negatively impact the network bandwidth, and consequently increase the LLDP propagation interval. To minimise these negative impacts, we consider the maximum bandwidth of our experimental network to be equal to 1Gb/s. We run the Floodlight controller, including the RLV defence system, on the same server to protect the network. We also configured a time series InfluxDB [17] database to store the collected LLDP data. InfluxDB is an open-source time-series platform designed for real-time analytics. We implement the ML classification part of the RLV in Python 3.7 using the Scikit-learn library [30].

B. Running LLA Tests

We run LLA on simulated network topology to show the weakness of *TopoGuard+* in detecting the attack. For launching LLA, we build the network topology of Fig. 9(a) adopted from [6]. There are two compromised hosts, i.e., h_1 and h_2, connected to switches s_1 and s_3. They play the role of the *flooder* and *listener* in the *overload* and *relay* phases of the LLA. The two compromised hosts communicate via a dedicated out-of-band channel with 10ms of link delay. To launch the *overload phase* of the LLA, we use *arping* to send 1 million ARP request messages with 1-μs interval toward the switches. For *relay phase*, we use *scapy* [31] library in Python to relay the received LLDP packets via the out-of-band channel toward s_1 and s_3.

LLA Security Performance: To inspect the behaviour of *TopoGuard+* against LLA, we activated the Floodlight system log. In this experiment, our goal is to demonstrate that *TopoGuard+* cannot detect LLA. Fig. 9(b) depicts the log of *TopoGuard+* before and after launching LLA. In normal operation, all link latency values are less than the calculated threshold value, which is ≈7 ms (see Fig. 9(b) lines 1-3). Afterwards, we run the LFA and found that *TopoGuard+* detects the attack (see Fig. 9(b) lines 6-7). It is because the latency of received LLDP, e.g., 33*ms*, is greater than the threshold, i.e., 15*ms*. Normally, the control link latency value is around 1*ms* (see Fig. 9(b) lines 9 and 12).

In our next step, we launch the LLA through the *overload phase*. We observe a spike in control link latency value to ≈ 90*ms* (see Fig. 9(b) lines 17-18). To prevent overloading, the Floodlight controller uses a simple throttle strategy. Since we

issue the ARP packets for a short time, the controller cannot detect floods. We apply the *relay phase* at this point (see Fig. 9(b) lines 19-22) and observe that *TopoGuard+* fails to detect the attack. The result is a bidirectional fabricated link between s_1 and s_3 that is added as a normal link to the network topology.

Running Example: The adversary can insert the fabricated link using LLA even in the presence of *TopoGuard+*. We now show a running example of LLA using a simple topology on the Floodlight controller. We take a snapshot of our network topology from the WebUI of the Floodlight controller in Fig. 9 before and after launching the LLA. Fig. 9(c) presents the controller view of the current links in which there is no link between switch s_1 with DPID [00:00:00:00:00:00:00:01] and switch s_2 with DPID [00:00:00:00:00:00:00:03]. However, after launching the attack, the controller misleads into believing a direct link between these two switches (see Fig. 9(d)).

C. RLV Performance Evaluation

To assess the performance of our proposed defence model, first, we run RLV on the Floodlight controller. Then, the different scenarios of link fabrication attacks, including LLA and LFA, are launched on a set of linear network topologies where the number of switches varies from 20 to 200. We consider a linear topology commonly studied in various previous works on link fabrication attack [6], [20] to evaluate the performance of the defence algorithms. In addition, some compromised hosts are connected to several switches to initiate the mentioned attack scenarios. Next, the real-time decisions of RLV in distinguishing between the normal and fabricated links are analysed. We evaluated the performance of RLV by measuring a broad range of ML metrics, including numerical metrics, such as F1 and Cohen's kappa (Ka) [32], and rank metrics, such as receiver operator characteristic (ROC) and precision-recall (PR) curves. The objective is to identify effective ML algorithms as the RLV model. Finally, we test the scalability and robustness of the RLV system and investigate its performance in the presence of background traffic.

1) Numerical Evaluations: True positive rate (TPR) and false-positive rate (FPR) are two numerical metrics which are widely used to evaluate the performance of ML-based attack detection models. TPR, which is also known as recall or detection rate (DR), identify the fraction which is correctly classified as fabricated links, while FPR, which is also called false alarm (FA), presents the rate of normal links that are misclassified as fabricated links, measuring using Eq. (8) and Eq. (9),

$$TPR = \frac{TP}{TP + FN} \tag{8}$$

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

where true negative (TN) stands for the number of links correctly classified as normal links. Our proposed RLV algorithm aims for high TPR while minimising FPR to detect more fabricated links and prevent the unwarranted removal of the normal link. In Section IV-D1, we discuss the trade-off between TPR and FPR in more detail and show the effectiveness of RLV in balancing these two metrics.

TABLE IV
THE DETECTION PERFORMANCE COMPARISON (%)

Method	Alg.	TPR	FPR	Pr	F1	Ka
RLV	MLP	56.95	0.14	88.99	69.45	68.97
	RF	67.22	0.60	69.21	68.2	67.57
	KNN	38.68	0.19	80.24	52.2	51.58
	SVM	32.66	0.15	81.13	46.57	45.95
MLLG [12]	MLP	99.03	52.5	4.03	7.75	4.11
	RF	99.70	55.36	4.31	8.26	4.65
	KNN	93.47	56.69	4.46	8.52	4.94
	SVM	96.77	58.9	4.53	8.65	5.07
TopoGuard+ [6]	LLI	26.65	0.66	44.81	33.42	32.43

We also evaluate the performance of our proposed RLV algorithm by measuring precision (Pr) and F1 using Eq. (10) and Eq. (11).

$$Pr = \frac{TP}{TP + FP} \tag{10}$$

$$F1 = \frac{2}{Pr^{-1} + TPR^{-1}} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{11}$$

Pr values indicate the number of links classified as fabricated links that are truly fabricated links. F1 considers a weighted average of the TPR and Pr to provide a high attack detection rate while reducing the false alarm rate.

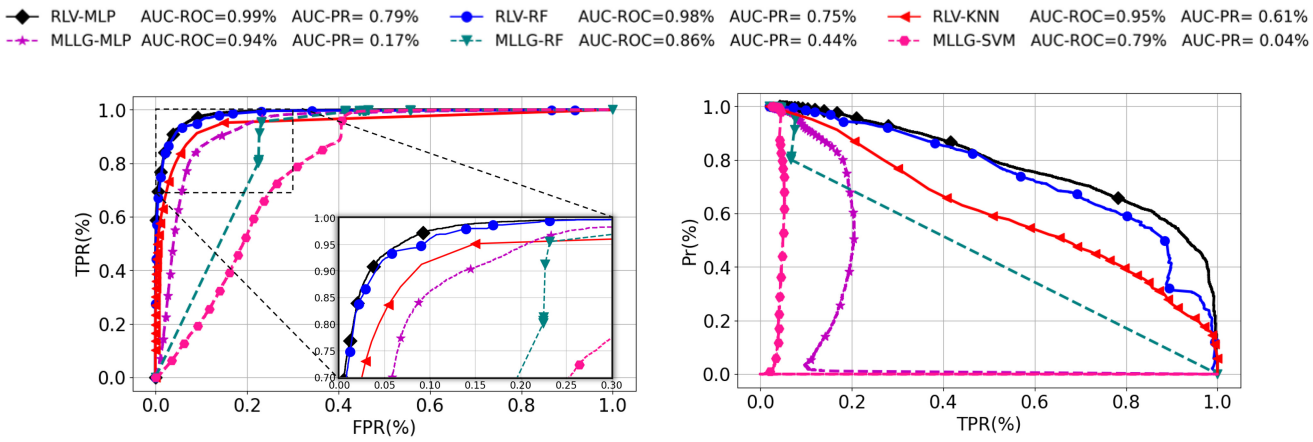
As we mentioned in Section III-B4, the RLV classifier is trained on the highly imbalanced dataset where the number of fabricated links in class 1 is much less than the number of normal links in class 0. The skewed data problem might bias ML performance metrics, such as accuracy, and mislead the final evaluation [33]. The high accuracy is not a reliable result in using a highly imbalanced dataset because the classifier is under-trained with the minority class data. It misclassifies nearly all negative samples into the positive class. Ka metrics [32] is an alternative measure to the accuracy, particularly in the imbalanced dataset. It measures the accuracy of the classifier while penalising random predictions, i.e., all-positive or all-negative predictions, using Eq. (12)

$$Ka = \frac{A_o - A_c}{1 - A_c} \tag{12}$$

where A_o and A_c represent the observed accuracy and random chance accuracy, respectively.

F1 is recommended as the appropriate evaluation metric [34], [35]. It is a combination of TPR and *Pr* metrics which is influenced by a balanced impact from the majority and minority classes. We evaluate the performance of our proposed RLV defence by measuring TPR, FPR, Pr, F1 and Ka. Then, the obtained results are compared with those of running *TopoGuard+* [6] and MLLG [12] to show the effectiveness of the RLV system. Table IV shows the performance of the different classifiers of RLV against the classifiers in the MLLG and LLI module in *TopoGuard+*. We find that among the above three defence algorithms, RLV classifiers present the best results in terms of F1, Pr, FPR and Ka.

MLP and RF classifiers in RLV achieve a reasonable TPR $\approx$ 60% and promising FPR, less than 0.6%, which leads



(a) Comparing AUC-ROC for ML algorithms

(b) Comparing AUC-PR for ML algorithms

Fig. 10. AUC-ROCs and AUC-PRs for RLV and MLLG on different ML algorithms.

to the highest rate of $\approx$ 68% for F1. The reason is that RF classifies an incoming link based on judgments received from several independent decision trees. This approach can reduce the impact of high variance latency data in large-size network typologies, as shown in our dataset in Fig. 8. RF mitigates the effect of outliers in classification, particularly when the latency intervals of the normal and fabricated links overlap. Furthermore, MLP calculates the error rates for each prediction through the backward propagation algorithm, which leads to more accurate detection and the highest performance.

Compared with MLP and RF algorithms, KNN shows a great FPR of 0.19% but a lower TPR, i.e., 38.68%. This is because KNN classifies each incoming link by looking at K nearest neighbours. However, this approach fails to work correctly when the fake link point is mostly surrounded by normal link points. Similar behaviour is observed for SVM, which results in the lowest TPR and F1 rate of 32.66% and 46.57%, respectively. The MLLG shows the highest TPR, more than 90%, at the cost of the significant rate for FPR $\approx$ 52% and the negative impact of lowest Pr, F1 and Ka. Although MLLG and RLV use similar ML algorithms, we observed that MLLG shows the lowest performance in dynamic network size. The main reason is that the MLLG model is trained based on link latency values without considering the network size. Therefore, MLLG is useful in networks with less frequent topology updates. *TopoGuard+* shows the lowest TPR, i.e., 26.65%, while it has the better F1 rate compared with MLLG. The reason is that the LLI module of *TopoGuard+* could slightly update the latency history based on the new arrival latency values [10].

2) Rank Evaluations: We now report the performance of the RLV classifiers using ROC and PR curves. PR curves have been considered an alternative to ROC curves for a highly imbalanced dataset with skewed class distribution [36]. The ROC curve illustrates TPR versus FPR in which the area under the curve (AUC) is calculated to determine which classifier best predicts the fabricated or normal links. In PR space, the x-axis shows recall (same as TPR), and the y-axis stands for Pr values. Figs. 10(a) and 10(b) show ROC curves and PR curves of RLV and MLLG classifier models, respectively. RLV-MLP (RLV-based MLP classifier) and RLV-RF (RLV-based RF classifier) curves dominate other ROC space and PR space curves and achieve the highest AUC-ROC of 0.987 and 0.983 and also highest AUC-PR of 0.789 and 0.746, respectively. Different presentations of the curves in the ROC and PR spaces provide a more informative view of the ML algorithm's performance. In the ROC space, RLV-MLP and RLV-RF curves are approximately close to optimal (the upper-left-hand corner), while in the PR space, there is still considerable room for improvement to meet the optimal (upper-right-hand corner). PR curves illustrate differences between RLV-MLP and RLV-RF algorithms that are not apparent in ROC curves. Looking at RLV-MLP in RP curves can expose a clear advantage over RLV-RF.

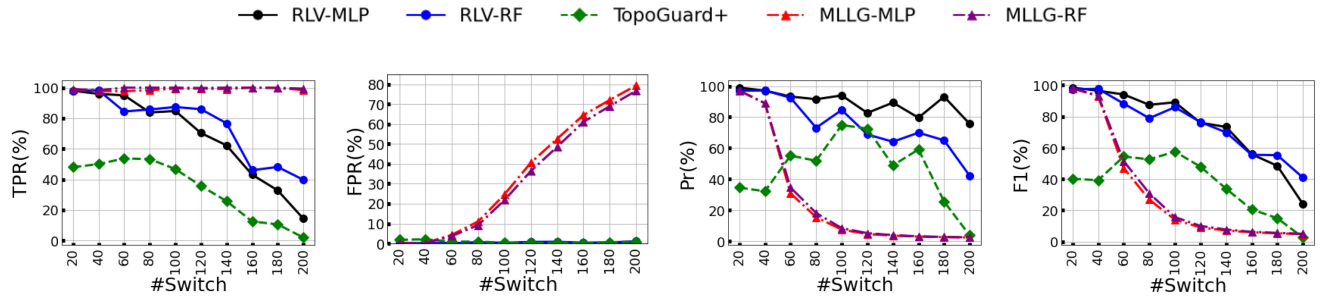
D. RLV Performance in Different Network Scenarios

We selected MLP and RF as the best models for RLV and MLLG. The reason is that Table IV and Fig. 10 show that MLP and RF algorithms perform better than SVM and KNN algorithms in terms of numerical evaluation results, such as the F1 measure and the rank evaluation results, like AUC-ROC and AUC-PR. We summarise reasons for the superiority of these two classifiers over other algorithms listed below, as discussed in Sections IV-C1 and III-B4.

- RF works based on ensemble learning which could mitigate the impact of high variance latency data, particularly in large-scale network topology.
- MLP utilises the backward propagation algorithm to improve the prediction results.

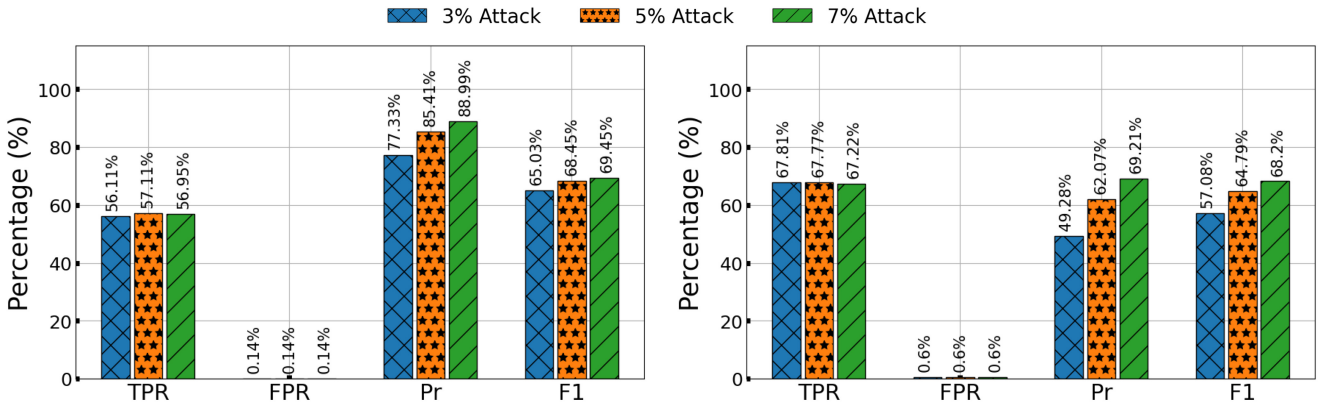
For the rest of this paper, we focus on RF-based RLV and MLP-based RLV to investigate the RLV performance in different network scenarios.

1) Scalability: In this section, we report how RLV scales when the network size changes by adding a set of switches and links. For this purpose, we vary the number of switches from 20 to 200, where the largest topology has 200 switches with 198 inter-switch links. Fig. 11(a) shows that MLLG has a perfect TPR rate, nearly 100%, while for RLV and *TopoGuard+* a downward trend is observed, ends in 200



(a) Attack detection (b) False alarm (c) Precision (d) F1-score

Fig. 11. Study of RLV, MLLG and *TopoGuard+* performance for different number of switches.



(a) RLV-MLP (b) RLV-RF

Fig. 12. Impact of different attack rates on RLV-MLP and RLV-RF performance.

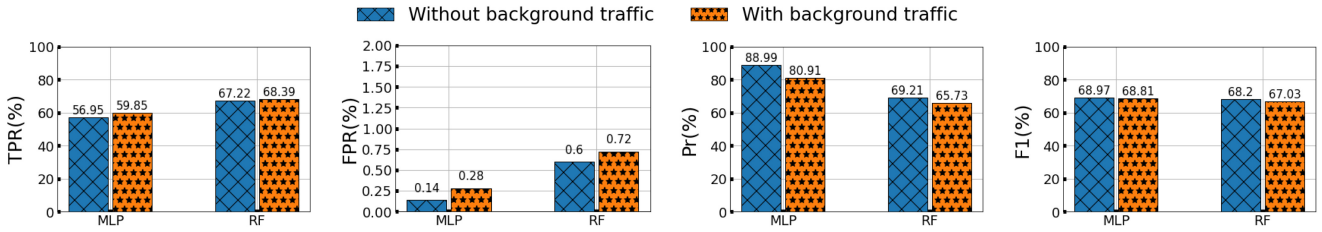
switches with $\approx$ 40%, 17%, and 0% TPR for RLV-RF, RLV-MLP, and *TopoGuard+*, respectively. Looking at the FPR rate in Fig. 11(b) exposes a critical trade-off between providing a high attack detection rate, i.e., TPR, and keeping a low false alarm rate, i.e., FPR. Despite the best performance in the TPR rate, MLLG experiences a dramatic upsurge in FPR. However, *TopoGuard+* and RLV achieve a low false alarm rate of less than 1%. In addition, RLV-RF and RLV-MLP achieve a high TPR rate compared with *TopoGuard+* while they experience nearly the same false alarm rate. Hence, RLV has an advantage over MLLG and *TopoGuard+*.

The downward trend of TPR in RLV and *TopoGuard+* is that by increasing the network size, the latency interval of the fabricated link overlaps with the interval of normal link latency. It makes attack detection more challenging and consequently decreases the TPR rate. However, the TPR rate in MLLG remains stable when the number of switches increases. This is because the MLLG model is trained based on a single topology, and the attack interval can be fixed as the network size is changed. As a result, by increasing the number of switches, almost all normal links are misclassified as fabricated links. This leads to the perfect attack detection rate while destroying normal links in the network.

The RLV exhibited superior Pr and F1 performance is compared with other algorithms as shown in Fig. 11(c) and Fig. 11(d), respectively. It means a trade-off between the gain of attack detection (TPR) and the cost of false alarm (FPR). RLV provides the best balance and achieves higher values in Pr and F1.

2) Robustness: In the second experiment, the robustness ability of RLV under different attack rates is investigated. To this end, we increase the number of positive samples (fabricated links) from 3% attack to 5%, and 7% attack in our simulated network. Let α, β, and γ represent 3%, 5%, and 7% attack rates, respectively. Fig. 12 shows TPR, FPR, Pr, and F1 measures for RLV-MLP and RLV-RF on different attack rates. For a fixed number of negative samples (normal links), we have $FP_\alpha = FP_\beta = FP_\gamma$ and $TN_\alpha = TN_\beta = TN_\gamma$. Thus, we conclude $FPR(\alpha) = FPR(\beta) = FPR(\gamma)$ (see Eq. (9)). Despite increasing the attack rate, we observed that RLV still maintained its TPR rate with less than 1% changes. For example, RLV-RF resulted TPR rate with 67.81%, 67.77% and 67.22% for 3%, 5% and 7% attack, respectively (see Fig. 12(b)). Thus, $TPR(\alpha) \approx TPR(\beta) \approx TPR(\gamma)$. This result shows that the RLV provides a robustness algorithm for attack detection under different attack rates. As we expected, Pr metric exposes a minor increase between three attack rates where the Pr for 7% attacks has advantage over other rates, i.e., $Pr(\alpha) \leq Pr(\beta) \leq Pr(\gamma)$. This increase exists because we have $TP_\alpha \leq TP_\beta \leq TP_\gamma$ (see Eq. (10)). By considering the value of Pr and TPR, it is obvious that $F1(\alpha) \leq F1(\beta) \leq F1(\gamma)$ as we can see in obtained result in Fig. 12.

3) Background Traffic: In this section, we investigate the accuracy of the RLV system with various levels of background traffic. We measure TPR, FPR, Pr, and F1 parameters to show the consequences of Poisson-arrived traffic on the RLV model performance. The results show that, in general, background



(a) Attack detection (b) False alarm (c) Precision (d) F1-score

Fig. 13. Impact of Poisson arrived traffic on RLV-MLP and RLV-RF performance.

traffic flows have a negligible impact on the performance of RLV, and it still could work effectively compared with the network without background traffic (see Fig. 13). In more detail, slight decreases by only $\approx 0.16\%$ and $\approx 1.17\%$ are reported in F1 for MLP and RF, respectively (see Fig. 13(d)). The reason is that the Poisson-based traffic might impose minor queuing delays in some switches and the controller, which causes growth in T_{LLDP} associated with fabricated and normal links. This minor increase could cause a bit growing in FPR (see Fig. 13(b)), and TPR (see Fig. 13(a)). Consequently, due to Eq. (10), Pr slightly decreases (see Fig. 13(c)).

V. Discussion

This section discusses different concerns and limitations of our attack scenario and proposed countermeasures.

LLA: To place LLA in the network of a business or enterprise, the adversary must compromise two end hosts connected through an out-of-band channel or a dual-homed host. In addition, the adversary needs to issue some probe packets over different intervals. This helps the attacker to gain some information from the network regarding the probe packet intervals issued by the controller. The frequency of issuing probe packets by the controller of *TopoGuard+* is five seconds. Moreover, most network intrusion detection systems, e.g., Snort and Bro, fail in detecting ARP floods when the false positive rate is low [6]. Attackers exploit such vulnerabilities to initiate the *overload phase* in LLA and bypass the defence mechanism of the SDN controller.

RLV deployment: We implement MLLG and RLV systems in Floodlight controller by extending the defence system of *TopoGuard+* [6]. MLLG detects LFA and LLA when the network topologies are stable over time without any updates. However, it is a proactive solution that needs a dataset of link latency information before taking action. In contrast, RLV is a reactive solution that can create its dataset during network operation. This feature of RLV makes it a ready to deploy solution for many networks.

As we described in Section III-B, the RLV is considered to be an independent component located outside the Floodlight controller. The main reason behind such placement is that RLV can be integrated with the other SDN controllers with minimal effort. However, implementing RLV inside the controller imposes a considerable effort to customise the controller's source code in case of using other SDN controllers rather than Floodlight.

RLV performance: The RLV has high precision, meaning that when it labels a link as fake, there is a high level of certainty that it is fake. However, the recall (or TPR) is modest, meaning the RLV may overlook some instances of fake links (see Table IV). Note that in some applications, such as the SDN IP backbone [40], which contains hundreds of high bandwidth links, the RLV system must ensure high precision, even at the expense of a modest recall, to prevent the unwarranted removal of a normal link. We can improve the modest recall in RLV through subsequent rounds of model training and testing. In this case, the dataset would be refined and completed iteratively. The repeated training, testing, and updating of data would eventually identify fake links that were not identified during the initial training and testing. Additionally, combined supervised machine-learning techniques and sophisticated hyperparameter optimisation algorithms [41], [42] might help to improve the modest recall and deserve further investigation. However, we need to align the achieved performance with the available resources.

The RLV-MLP and RLV-RF models can produce superior results in different scenarios. However, some factors make these two algorithms different. First, training in MLP algorithm is time-consuming and computationally intensive, requiring parallel processing across multiple CPU cores. In contrast, the RF algorithm is fast to train and requires less computation [27]. In this paper, the training time for RLV-RF was much less than RLV-MLP (a few seconds compared to minutes). Therefore, we must consider the trade-off between performance and computational resources before choosing the best forecast model. For example, in networks with real-time constraints and limited resources, RLV-RF can be a better choice than RLV-MLP. Second, the MLP requires greater expertise to tune the hyperparameters. In contrast, the RF does not require much fine-tuning, and one can often obtain the best results with default parameters.

If the burden of the neural network can be tolerated, autoencoders (AEs) [43] could be used to detect unsupervised anomalies [44]. In the AE, compressed encoding is obtained from input, and decoding is used to reconstruct the data. Normal input tends to have lower reconstruction errors since it is close to the training data, while abnormal input tends to have higher reconstruction errors. However, this assumption may fail in our LLDP dataset, which contains outliers and noise. The reason is that as the network size grows, the LLDP data complexity increases, which means that extracted features, such as T_{LLDP}, are more likely to be shared between normal

TABLE V
COMPARISON OF THE PROPOSED SYSTEMS WITH THE PREVIOUS DEFENCE ALGORITHMS AGAINST LFA

Countermeasure	Technique	Advantages	Disadvantages
TopoGuard [6]	Port labelling strategy to classify ports based on traffic	✓ Disallow hosts from propagating LLDP	× Ineffective solution in a dynamic network size × Failed to detect PAA
[37]	Using a baseline threshold to compare distributions of link latency values	✓ Nearly real-time approach	× Significant false alarm during peak hours × Using a single threshold
TopoGuard+ [10]	Comparing link latency interval with a single threshold	✓ Protect against both in-band and out-of-band attacks	× Removal of normal links during peak hours × Using a single threshold × Port monitoring overhead × Failed to detect LLA
SPV [38]	Verification of link validity through the transmission of probe packets per new link	✓ Nearly real-time detection ✓ Recognise known and unknown LFA	× Communication overhead × Sharing TLS keys explicitly
MLLG [12]	Classifying link latency values using supervised ML	✓ High attack detection rate ✓ Dynamic latency threshold	× Overhead of training model × Offline evaluation × Lack of scalability for large SDN
[18]	Using deep reinforcement learning in vehicular network	✓ self-recovery mechanism	× Lack of attack detection mechanism × Overhead of learning model
Hybrid-Shield [39]	Using a verification method to verify MHL and blocking the fake MHLs	✓ A lightweight countermeasure ✓ High accuracy rate in Hybrid SDN	× Ineffective in LLDP discovery process
RLV	Scaled and real-time classification on link latency values using weighted supervised ML	✓ Real-time approach ✓ High performance in large scaled SDN ✓ Robustness on attack rate	× Overhead of learning model × Retraining model for different latencies

and abnormal LLDPs (see Fig. 8), resulting in mixed feature values. In this situation, we can reconstruct data from both normal and anomalous LLDPs properly. Denoising autoencoder [45] is one method to address this challenge, which requires a source of clean, noise-free data to train. However, real-world networks rarely provide such information. In this case, the RLV model might need to isolate noise and outliers from the input values, and the encoder is trained after this isolation. We leave this aspect for future research.

RLV application: RLV works for networks with bounded link latency when the latency variation is not high. However, its applications in geographically distributed networks need revisiting. To solve this problem, we could consider distributed and centralised strategies. In distributed RLV, we can partition the network into multiple small networks and run RLV for each. Each part of the network owns a local RLV model, which is trained based on the local link latency data. In this case, the system will have multiple controllers, and their functionalities can be synchronised. Different techniques can be applied here, such as hot or cold synchronisation in distributed systems. The centralised RLV divides the network into domains, and the latency data aggregates from all domains to one centralised controller. The latency data for each partition needs to be labelled with domain ID before sending it to the controller. A centralised RLV model is deployed in the controller and trained based on the received data, which has a global view and full information of the whole network. However, the RLV model needs to reconfigure the domain ID as a new feature in this case. We leave this aspect of our system for the future.

VI. RELATED WORK

The work [6] introduced LFA. The adversary can attack the network without knowing the vulnerability of the controller or the control plane. The goal of LFA is to add a fabricated link between two switches in the network. *TopoGuard* [6] utilises a port labelling strategy to detect LFAs. The controller classifies ports based on the received packet using three labels: SWITCH, HOST, and ANY. The adversary can compromise a host by pretending to be a switch so that it relays LLDP packets in *TopoGuard*. The authors of [10] introduced a new type of LFA called *port amnesia attack (PPA)*. The authors demonstrated that bypassing port-labelling with *TopoGuard* could be accomplished by switching the port status of the compromised host from down to up during the propagation of the LLDP protocol. In addition, they designed *TopoGuard+*, which includes a link latency inspector (LLI) module for detecting fabricated links. Using the LLI, *TopoGuard+* can calculate a link latency and compare it to a latency threshold to detect the attacks. However, the threshold cannot be adjusted according to traffic patterns, and as a result, normal links might be removed.

The work in [20] gradually increases the latency threshold to exceed the latency of the out-of-band communication channel. The attack in [20] requires preparation for hours. Additionally, they offer no specific protection against attack. The study in [37] also presented a threshold-based algorithm that compares latency samples from LLDP messages with a threshold. However, the approach may cause dramatic false-positive predictions, similar to that of [20]. Using the worm-hole attack proposed in [46], a packet is relayed over the fabricated link without using any out-of-band communication channels.

The study in [38] proposed a stealthy probing-based verification (SPV) system that sends probe packets to the switches to discover potentially fabricated inter-switch links. SPV is less secure due to its integration issues with the SDN controllers. The tool in [47] can give us more insight into the attack roots. The work in [48] proposed a secure architecture to counter the threat of attacks caused by malicious hosts. Nevertheless, this architecture fails in detecting all types of LFA, similar to [11].

In our recent work [12], we introduced the link latency attack (LLA) that can bypass the current defence system, such

as *TopoGuard+*. We also proposed a machine learning-based link guard (MLLG) algorithm that can protect SDN topology from LLA and LFA attacks. However, the efficiency of MLLG reduces in large-scale SDN.

Table V shows a comparison between the existing defence systems against LFA. The studies in [18], [49] used deep reinforcement learning (DRL) to enhance topology poisoning defence in the SDN-based vehicular networks. Their solutions, however, focus on the failure recovery mechanisms rather than providing a detection system for TPAs. The study in [39] proposed a multi-hop link (MHL) fabrication attack employed in hybrid SDN networks that include both traditional switches and SDN switches. The hybrid SDN controller uses broadcast domain discovery protocol (BDDP) to discover the multi-hop link between two switches. The research also presented a Hybrid-Shield defence system to protect the Hybrid SDN. The Hybrid-Shield defence employs a verification method to identify fake MHLs. However, the solution is not applicable where LLDP messages are propagated among OpenFlow switches.

VII. Conclusion

This paper introduced Link Latency Attack (LLA) and proposed two machine learning-based defence mechanisms called MLLG and RLV. The MLLG work when the network topology rarely updates, while RLV is a real-time and scalable defence mechanism to detect LFA and LLA. RLV creates its dataset to train the model using classification algorithms and can update the dataset constantly. This system is suitable for networks that dynamically add/remove links. We plan to realise the implementation of RLV in the programmable networks. Also, we plan to extend the RLV to have a sophisticated mechanism to update its dataset, such as using elastic time windows methods.

References

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

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

[3] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in software defined networks," in *Proc. Int. Conf. Signal Process. Commun. Syst. (ICSPCS)*, 2014, pp. 1–8.

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

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

[6] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2015, pp. 8–11.

[7] N. Kaur, A. K. Singh, N. Kumar, and S. Srivastava, "Performance impact of topology poisoning attack in SDN and its countermeasure," in *Proc. 10th Int. Conf. Security Inf. Netw. (SIN)*, 2017, pp. 179–184.

[8] "Software-defined networking market by component (SDN infrastructure, software, and services), SDN type (open SDN, SDN via overlay, and SDN via API), end user, organization size, enterprise vertical, and region—Global forecast to 2025." Research and Market. 2021. [Online]. Available: https://bit.ly/3pDWlJK

[9] "Cost of a data breach report 2020." Ponemon Institute. 2021. [Online]. Available: https://bit.ly/3l0AjR4

[10] R. Skowyra et al., "Effective topology tampering attacks and defenses in software-defined networks," in *Proc. Int. Conf. Depend. Syst. Netw. (DSN)*, 2018, pp. 374–385.

[11] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting security attacks in software-defined networks," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2015, pp. 8–11.

[12] S. Soltani, M. Shojafar, H. Mostafaei, Z. Pooranian, and R. Tafazolli, "Link latency attack in software-defined networks," in *Proc. Conf. Netw. Serv. Manag. (CNSM)*, 2021, pp. 187–193.

[13] "MININET: An instant virtual network on your laptop (or other PC)." MININET. Accessed: Jan. 2023. [Online]. Available: http://mininet.org/

[14] "Floodlight." Open SDN controller. Accessed: Jan. 2023. [Online]. Available: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller

[15] T. Arnold et al., "Beating BGP is harder than we thought," in *Proc. Workshop Hot Topics Netw.*, 2019, pp. 9–16.

[16] Z. Akhtar et al., "Oboe: Auto-tuning video ABR algorithms to network conditions," in *Proc. Conf. Spec. Interest Group Data Commun.*, 2018, pp. 44–58.

[17] "InfluxDB OSS 2.4." InfluxDB. Accessed: Jan. 2023. [Online]. Available: https://docs.influxdata.com/influxdb/v2.4

[18] J. Wang, Y. Tan, J. Liu, and Y. Zhang, "Topology poisoning attack in SDN-enabled vehicular edge network," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9563–9574, Oct. 2020.

[19] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (RADIUS)," IETF, RFC 2865, 2000.

[20] E. Marin, N. Bucciol, and M. Conti, "An in-depth look into SDN topology discovery mechanisms: Novel attacks and practical countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2019, pp. 1101–1114.

[21] W. Xia, P. Zhao, Y. Wen, and H. Xie, "A survey on data center networking (DCN): Infrastructure and operations," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 640–656, 1st Quart., 2017.

[22] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *Proc. Netw. Syst. Des. Implement. (NSDI)*, 2016, pp. 537–549.

[23] M. Alizadeh et al., "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, 2013, pp. 435–446.

[24] A. Fernández, S. García, M. J. del Jesus, and F. Herrera, "A study of the behaviour of linguistic fuzzy rule based classification systems in the framework of imbalanced data-sets," *Fuzzy Sets Syst.*, vol. 159, no. 18, pp. 2378–2398, 2008.

[25] X.-Y. Jing et al., "Multiset feature learning for highly imbalanced data classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 1, pp. 139–156, Jan. 2021.

[26] O. Aouedi, K. Piamrat, and B. Parrein, "Ensemble-based deep learning model for network traffic classification," *IEEE Trans. Netw. Service Manag.*, early access, Jul. 26, 2022, doi: 10.1109/TNSM.2022.3193748.

[27] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 2, pp. 1153–1176, 2nd Quart., 2016.

[28] S. Weerasinghe, T. Alpcan, S. M. Erfani, and C. Leckie, "Defending support vector machines against data poisoning attacks," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2566–2578, 2021.

[29] L. Buitinck et al., "API design for machine learning software: Experiences from the scikit-learn project," 2013, *arXiv:1309.0238*.

[30] F. Pedregosa et al., "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[31] "Scapy: Packet crafting for Python2 and Python3." Accessed: Jan. 2023. [Online]. Available: https://scapy.net/

[32] J. Carletta, "Assessing agreement on classification tasks: The kappa statistic," 1996, *arXiv:cmp-lg/9602004*.

[33] M. Bekkar, H. K. Djemaa, and T. A. Alitouche, "Evaluation measures for models assessment over imbalanced data sets," *J. Inf. Eng. Appl.*, vol. 3, no. 10, pp. 27–38, 2013.

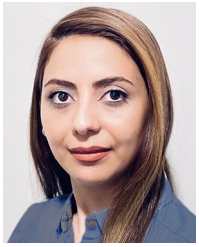
[34] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[35] L. A. Jeni, J. F. Cohn, and F. De La Torre, "Facing imbalanced data recommendations for the use of performance metrics," in *Proc. Humaine Conf. Assoc. Adv. Affect. Comput. (AAAC)*, 2013, pp. 245–251.

[36] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2006, pp. 233–240.

[37] D. Smyth, S. McSweeney, D. O'Shea, and V. Cionca, "Detecting link fabrication attacks in software-defined networks," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2017, pp. 1–8.

[38] A. Alimohammadifar et al., "Stealthy probing-based verification (SPV): An active approach to defending software defined networks against topology poisoning attacks," in *Proc. Eur. Symp. Res. Comput. Security (ESORICS)*, 2018, pp. 463–484.
[39] P. Shrivastava and K. Kataoka, "Topology poisoning attacks and prevention in hybrid software-defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 1, pp. 510–523, Mar. 2022.
[40] G. Nencioni, B. E. Helvik, and P. E. Heegaard, "Including failure correlation in availability modeling of a software-defined backbone network," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 4, pp. 1032–1045, Dec. 2017.
[41] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2013, pp. 847–855.
[42] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6765–6816, 2017.
[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[44] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2017, pp. 665–674.
[45] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 1096–1103.
[46] J. Hua, Z. Zhou, and S. Zhong, "Flow misleading: Worm-hole attack in software-defined networking via building in-band covert channel," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 1029–1043, 2020.
[47] B. E. Ujcich, S. Jero, R. Skowyra, A. Bates, W. H. Sanders, and H. Okhravi, "Causal analysis for software-defined networking attacks," in *Proc. Security Symp.*, 2021, pp. 3183–3200.
[48] V. Varadharajan and U. Tupakula, "Counteracting attacks from malicious end hosts in software defined networks," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 160–174, Mar. 2020.
[49] J. Wang and J. Liu, "Location hijacking attack in software-defined space–air–ground-integrated vehicular network," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 5971–5981, Apr. 2022.



Sanaz Soltani received the master's degree in software engineering from the Amirkabir University of Technology (Tehran Polytechnic), Iran, in 2014. She is currently pursuing the Ph.D. degree with the Information and Communication Systems, 5GIC & 6GIC Innovation Centre, University of Surrey, U.K. Before, she was a Network Specialist with Huawei and MTN telecommunication companies involved in 4G and LTE projects. Her research interests include network softwarization, open-RAN, and network security and privacy.

Mohammad Shojafar (Senior Member, IEEE) received the Ph.D. degree in ICT from Sapienza University of Rome, Rome, Italy, in 2016 with an "Excellent" degree. He is a Senior Lecturer (Associate Professor) in network security and an Intel Innovator, a Professional ACM Member, an ACM Distinguished Speaker, a Fellow of the Higher Education Academy, and a Marie Curie Alumni, working in the 5G & 6G Innovation Centre, Institute for Communication Systems, University of Surrey, U.K. Before, he was a Senior Researcher and a Marie Curie Fellow with the SPRITZ Security and Privacy Research Group, University of Padua, Italy. He secured 310k for the ESKMARALD project funded by GCHQ, U.K., in 2022. He is also a PI of AUTOTRUST, a secure autonomous 5G-based traffic management platform the European Space Agency supported for around €750k in 2021. He was also a PI of the PRISENODE project, a €275k Horizon 2020 Marie Curie project in network security and Fog task/resource scheduling collaborating at the University of Padua. He was also a PI on an Italian SDN security and privacy (€60k) supported by the University of Padua in 2018 and a Co-PI on an Ecuadorian–British project on IoT and Industry 4.0 resource allocation ($20k) in 2020. He contributed to some Italian projects in telecommunications, like GAUChO, SAMMClouds, and SC2. He is an Associate Editor of IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE SYSTEMS JOURNAL, and *Computer Networks*. For additional information: https://www.surrey.ac.uk/people/mohammad-shojafar

Habib Mostafaei (Member, IEEE) received the Ph.D. degree in computer science and engineering from Roma Tre University in 2019. He is currently an Assistant Professor of Computer Science with the Eindhoven University of Technology. Before, he was a Postdoctoral Researcher with Technische Universität Berlin, where he was involved in the BIFOLD-BBDC project from 2019 to 2022. He worked as a full-time Faculty Member with the Computer Engineering Department, Azad University from 2009 to 2015. He is a member of ACM. Currently, his main research fields include networked systems, network management, and distributed systems. For additional information: https://mostafaei.bitbucket.io

Rahim Tafazolli (Senior Member, IEEE) is the Regius Professor and a Professor of Mobile and Satellite Communications. He is the Director of ICS and the Founder and Director of world's first 5G Innovation Centre with the University of Surrey, U.K. He is regularly invited by many governments for advise on mobile communications and in particular 5G technologies. He has given many interviews to International media in the form of television, radio interviews, and articles in international press.