A Lightweight Path Validation Scheme in Software-Defined Networks

Bing Hu[†], Yuanguo Bi[†], Kui Wu[§], Rao Fu[†], Zixuan Huang[†]
[†]School of Computer Science and Engineering, Northeastern University, Shenyang, China
[§]Department of Computer Science, University of Victoria, B.C., Canada

IEEE INFOCOM 2024 - IEEE Conference on Computer Communications | 979-8-3503-8350-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOM52122.2024.10621099

Abstract*—Software-Defined Networks (SDN) revolutionize traditional networks by separating control and data planes for enhanced agility and programmability. This separation, however, also opens up vulnerabilities, allowing adversaries to manipulate data plane forwarding and breach security policies. To counter this, we propose a Lightweight Path Validation Scheme (L-PVS) specifically designed for SDN environments. Our approach uses a simple validation scheme for packet forwarding paths that verifies the paths traversed by packets. Then, we further amplify the scheme with a network flow path validation to boost the validation efficiency. To reduce storage demands on switches during flow path validation, we develop a storage optimization method that aligns switch storage overhead with network flows rather than individual packets. Furthermore, we formulate a path partition scheme and present a Greedy-based KeySwitch Node Selection Algorithm (GKSS) to pinpoint optimal switches for path partition, significantly reducing overall data plane storage usage. Lastly, we design a technique using temporary KeySwitch nodes to identify anomaly switches when the controller encounters path validation failure. Evaluation results verify that L-PVS facilitates path validation with a reduced validation header size while minimizing the impact on processing delay and switch storage overhead.

***Index Terms*—software-defined networks, forwarding behavior anomaly, forwarding path validation, storage optimization.**

I. Introduction

THe original network protocols are not designed to authenticate packets' transmission path. This limitation cannot ensure packets are transmitted along their intended paths. Numerous cases have shown that routers are susceptible to compromise [1], [2], leading to packet forwarding paths that diverge from network policy [3]. This deviation presents a significant security risk: compromised routers can maliciously redirect packets, deviating from network policy and potentially enabling unauthorized access to sensitive data or disrupting network operations. Non-compliant packet forwarding can also result in network congestion, packet loss, or delay, significantly degrading network performance and user experience [4].

To address the above problem, path validation empowers network devices to verify that packets follow paths compliant with network policies, thus enabling swift detection of abnormal packet delivery behaviour. Significant research efforts have been devoted to implementing path validation in traditional IP networks [4]–[9]. Nevertheless, modern network applications like data centers and intelligent manufacturing often require frequent modification and customization of network traffic policies. This is a hard-to-meet demand for traditional IP networks [10]. For ease of network management and operation, Software-Defined Networks (SDN) decouple the network's control plane and data plane [11], [12]. The controller in the control plane forms the traffic policies and implements forwarding rules into the switches in the data plane via SDN protocols like OpenFlow [13]. The switches then forward packets based on these rules. Network operators can alter traffic policies through the controller to dictate how the switches forward packets. While this separation gives SDN many advantages over traditional networks, it invalidates the existing path validation mechanisms designed for traditional networks. Also, it makes the switches more vulnerable to attacks [14], [15].

Unfortunately, the SDN protocols are not designed to validate packet paths, allowing compromised switches to divert packets from their intended switches undetected by the controller. The issue of validating packet forwarding paths in SDN is challenging. Since a logically centralized controller governs all switch-forwarding behaviour, transmitting every packet path information to the controller could significantly consume switch-to-controller bandwidth, impairing the controller's capacity to distribute flow rules to the data plane. One solution involves the controller implementing path validation by gathering packet counts based on the corresponding matching rules [16]–[18]. However, the accuracy of the packet count-based method is inherently limited by the precision of packet statistics. Another approach adds a validation header to each packet to record the switch information along the transmission path [19]–[22]. Although the last-hop switch can compress packet validation headers, thereby reducing switch-to-controller bandwidth consumption, it still incurs significant switch storage resources when handling a large number of packets. Furthermore, during the validation process, the controller must compute the update procedure for each packet's validation header (packet-based validation in the controller), thereby consuming considerable computational resources. Consequently, it is *a substantial challenge to concurrently ensure low switch-to-controller bandwidth overhead, low switch storage consumption, low controller validation cost, and high validation accuracy.*

We propose a Lightweight Path Validation Scheme (L-PVS) for SDN to tackle the above challenges. Our path validation scheme is distinct from all existing solutions, as illustrated in Fig. 1. The key contributions of this paper include:

- We design a lightweight packet forwarding path valida-

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.

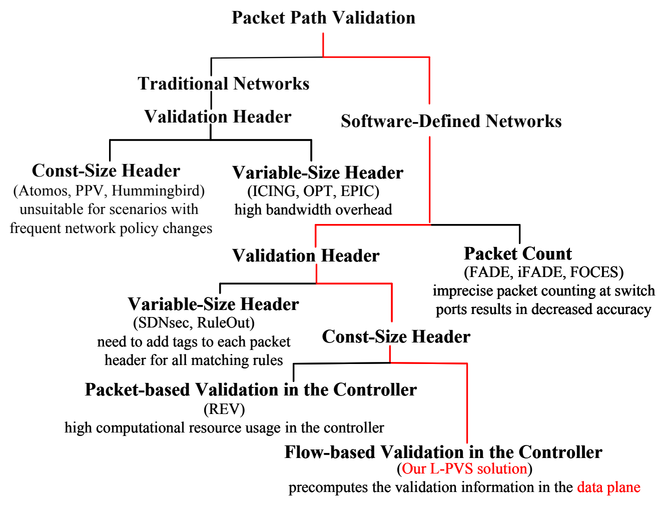

Packet Path Validation
Traditional Networks
Validation Header
Software-Defined Networks
Const-Size Header
(Atomos, PPV, Hummingbird)
unsuitable for scenarios with frequent network policy changes
Variable-Size Header
(ICING, OPT, EPIC)
high bandwidth overhead
Validation Header
Packet Count
(FADE, iFADE, FOCES)
imprecise packet counting at switch ports results in decreased accuracy
Variable-Size Header
(SDNsec, RuleOut)
need to add tags to each packet header for all matching rules
Const-Size Header
Packet-based Validation in the Controller
(REV)
high computational resource usage in the controller
Flow-based Validation in the Controller
(Our L-PVS solution)
precomputes the validation information in the data plane

Fig. 1. Classification of packet path validation.

tion scheme for SDN and provide an in-depth feasibility analysis of the validation process. To enhance validation efficiency, we extend the packet path validation to network flow path validation.

- We propose a method for optimizing the storage procedure, which precomputes the validation information at the last-hop switch. This method reduces both the storage overhead on switches and the validation overhead on the controller during the validation process.
- To minimize overall data plane storage overhead, we partition long paths into multiple sub-paths and validate overlapping sub-paths of different network flows simultaneously. We propose a heuristic algorithm, GKSS, to identify suitable switches (KeySwitch nodes) for path partition. In addition, we use temporary KeySwitch nodes to pinpoint anomaly switches when the controller encounters path validation failure.
- We implement the proposed scheme and evaluate its performance. Evaluation results confirm that L-PVS effectively reduces the size of validation headers while minimizing processing delay and switch storage overhead.

II. Related Work

A. Path Validation in Traditional Networks

In traditional networks, a common path validation method is adding a validation header to each packet. The ICING protocol [5] inserts Proofs of Provenance (PoPs) into headers for path validation. Routers verify packet paths by examining the PoPs. To reduce the storage and computation overhead in routers during the path validation process, the OPT protocol [6] uses dynamically recreatable keys (DRKey) for origin path tracing. EPIC [4], introduced by Legner *et al.*, achieves a 3-5 times reduction in communication overhead compared to schemes like DRKey and ICING. Nevertheless, including proofs from every router along the transmission path in the validation header still substantially increases bandwidth overhead, particularly for longer paths.

To reduce the bandwidth overhead in the validation process, Atomos [7] introduces a solution using a constant-sized validation proof. This approach prevents the increase of packet header-space overhead as the path length grows. In addition, both PPV [8] and Hummingbird [9] propose a probabilistic packet marking strategy to reduce computational overhead.

B. Path Validation in SDN

1) Packet Count-Based Path Validation: In SDN, the controller can collect data plane status information. As packets pass through each hop and undergo rule matching at each switch, the match count for a network flow should be consistent across all hops. To identify forwarding anomalies, the FADE method [16] installs dedicated rules to gather flow statistics. These statistics are then verified to detect irregularities. Building upon this approach, the iFADE method [17] reduces the number of required rules by aggregating network flows that follow the same sequence of switches. Furthermore, Zhang et al. propose FOCES [18], which constructs a Flow Counter Matrix (FCM) within the controller. Packet path validation is achieved by checking the consistency between the FCM and the flow counters in the data plane.

2) Validation Header Based Path Validation: Another approach for path validation in SDN involves adding a validation header to each packet. SDNsec [19] proposes that switches embed their proofs into the validation header, which can then be sent to the controller for path validation. In RuleOut [20], a tag sequence is added to each packet, corresponding to the rules the packet should match. This allows switches to verify each packet's forwarding correctness by checking the associated tag. The Verifying Rule Enforcement (REV) technique [21], [22] enhances path validation by adding switch identification to the validation header through encryption after packet reception. REV also uses a compressive message authentication code to reduce the size of the validation header for the same flow at the last switch.

All existing solutions have shortcomings. Packet count-based methods are sensitive to packet losses and network congestion because these factors may result in inaccurate packet counts in SDN switches. Validation header-based methods incur substantial bandwidth overhead because they include information about all switches along the path in the validation header. Compressing the validation header in the last hop switch can reduce switch storage and switch-to-controller bandwidth usage, but the controller would face high computational overhead. This is because it must compute the update process of the validation headers for each packet at every hop, leading to considerable computational demands for packet-based validation.

L-PVS is unique, as illustrated in Fig. 1. We list a qualitative comparison between existing path validation schemes and L-PVS in Table I.

III. System and Attack Models

We consider an SDN architecture where a logically centralized controller manages switches. The controller installs flow

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON OF EXISTING PATH VALIDATION SCHEMES VS. OUR L-PVS SCHEME

solution \ specification	Traditional Networks / SDN	Granularity	Method	Header Size	Validation Cost	Theoretical Accuracy
EPIC [4], ICING [5], OPT [6]	Traditional Networks	packet	validation header	variable	packet-related	100%
PPV [8], Atomos [7], Hummingbird [9]	Traditional Networks	packet	validation header	const	packet-related	100%
FADE [16], iFADE [17], FOCES [18]	SDN	flow	packet count	-	flow-related	About 99.5%
SDNsec [19]	SDN	packet	validation header	variable	packet-related	100%
RuleOut [20]	SDN	packet	validation header	variable	packet-related	100%
REV [21], [22]	SDN	flow	validation header	const	packet-related	100%
L-PVS	SDN	flow	validation header	const	flow-related	$1-10^{-6}$

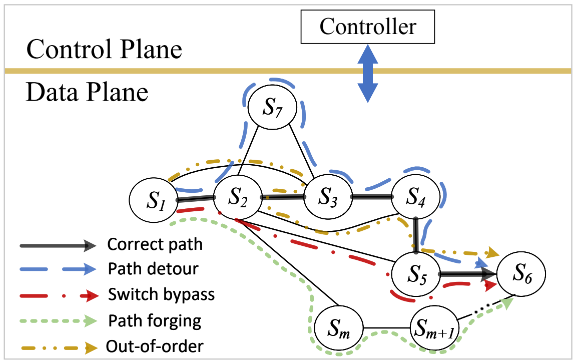


Fig. 2. Forwarding anomalies in the data plane.

rules into the flow tables of switches, which forward packets following the rules. Each rule consists of a matching field and an action field, with the former determining which packets to process and the latter determining how to process them.

The security of the SDN architecture might be compromised if an adversary exploits vulnerabilities of the switch OS [14], [15]. Adversaries can modify the output ports of flow rules on the compromised switches [22]. For example, as illustrated in Fig. 2, the correct forwarding path of a packet is $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \rightarrow S_6$. The adversary, however, may reroute the packet along a different path, resulting in a forwarding anomaly. There are four types of forwarding anomalies:

- **Path detour:** A malicious switch reroutes packets to a switch that is not the intended next-hop in the forwarding path, but later the packets return to the designated next-hop switch. In Fig. 2, S_2 sends packets to S_7, and then S_7 sends packets to S_3.
- **Switch bypass:** A malicious switch selectively forwards packets to a specific switch along the path, bypassing other switches in the path. In Fig. 2, switch S_2 forwards packets to S_5, but skips switches S_3 and S_4.
- **Path forging:** A malicious switch reroutes packets to a switch that diverges from the intended path, resulting in the packets being forwarded along another path. For instance, in Fig. 2, S_2 forwards packets to S_m, causing the forwarding path becomes $S_1 \rightarrow S_2 \rightarrow S_m \rightarrow S_{m+1} \rightarrow ... \rightarrow S_6$.
- **Out-of-order traversal:** Packets are being forwarded by switches that are part of the intended path but not in the correct sequential order. In Fig 2, packets are forwarded along the path $S_1 \rightarrow \mathbf{S_3} \rightarrow \mathbf{S_2} \rightarrow S_4 \rightarrow S_5 \rightarrow S_6$ instead of following the designated path.

Note that the above are all the forwarding anomalies known to the public. Handling other forwarding anomalies, if any, is left for our future work. Also, we only consider anomalies that affect the forwarding path of packets. Anomalies that do not alter the forwarding path, such as packet modification, packet dropping, or packet replay [23], are out of the scope of the paper. We assume that the controller and most of the switches can be trusted because, otherwise, the attacker can take down the whole network.

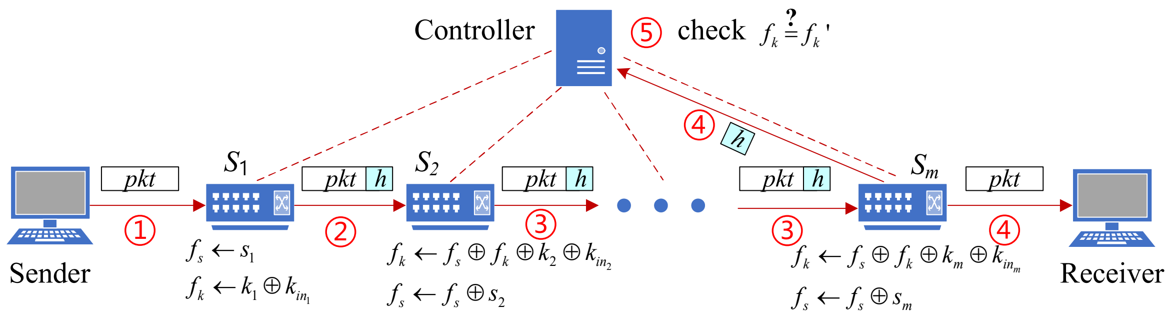
IV. METHOD FOR PACKET PATH VALIDATION

We first introduce our packet path validation method, which helps readers easily understand L-PVS's flow path validation process presented in the next section. When a switch connects to the controller, it receives a unique public identification. In addition, the switch receives keys, one for the switch itself and the rest for its ports. Note that keys are used to record the switch ports a packet traverses, while identifications are used to track the switch sequence. We classify switches into four distinct categories:

1) InSwitch: These switches receive packets directly from the source hosts.
2) RelaySwitch: These switches forward packets received from one switch to another.
3) OutSwitch: These switches send packets to the destination hosts.
4) KeySwitch: These switches divide paths into sub-paths, a process further detailed in Section V-C.

Operations of InSwitch: The InSwitch node attaches a validation header to each packet to track its intended path. For instance, as shown in Fig. 3, when a packet pkt enters the network, the InSwitch node, S_1, attaches the validation header h to pkt. The validation header comprises two fields: f_s and f_k. Inspired by [7], we utilize an additive operation to update the validation header as the switch processes the packet. This approach enables the validation header to maintain a constant

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.



Controller
⑤ check $f_k \overset{?}{=} f_k'$
④ h
Sender
pkt
①
S_1
$f_s \leftarrow s_1$
$f_k \leftarrow k_1 \oplus k_{in_1}$
pkt h
②
S_2
$f_k \leftarrow f_s \oplus f_k \oplus k_2 \oplus k_{in_2}$
$f_s \leftarrow f_s \oplus s_2$
pkt h
③
③
pkt h
S_m
$f_k \leftarrow f_s \oplus f_k \oplus k_m \oplus k_{in_m}$
$f_s \leftarrow f_s \oplus s_m$
pkt
④
Receiver

Fig. 3. Packet path validation.

size regardless of the path length. The additive operation is defined as

$$(f_s, f_k)\vec{\oplus}(s, k) = (f_s \oplus s, f_s \oplus f_k \oplus k), \tag{1}$$

where $\oplus$ denotes the XOR (exclusive OR) operation. We use the symbol $\vec{\oplus}$ to highlight the asymmetric property of the additive operation, meaning that the order of operands matters.

The InSwitch node S_1 initializes the validation fields of pkt before sending it to the next hop. This process is expressed as

$$f_s \leftarrow s_1,\ f_k \leftarrow k_1 \oplus k_{in_1}, \tag{2}$$

where s_1 refers to the identification of S_1, k_1 represents the key assigned to S_1 by the controller, and k_{in_1} represents the key assigned to the port from which S_1 received the packet[1]. We will use different subscripts to represent the above information in different switches. For instance, s_i represents the identification of switch S_i.

Operations of RelaySwitch: If a RelaySwitch node S_i (referring to $S_2, S_3, \ldots,$ or S_{m-1} in Fig. 3) receives a packet, it utilizes the additive operation in (1) to update the validation fields f_s, f_k of the packet, which can be expressed as

$$(f_s, f_k) \leftarrow (f_s, f_k)\vec{\oplus}(s_i, k_i \oplus k_{in_i}). \tag{3}$$

Operations of OutSwitch: When an OutSwitch node (e.g., S_m in Fig. 3) receives the packet, it firstly updates the validation fields using (3), then it sends the validation header to the controller.

Packet path validation in controller: The controller utilizes its knowledge of the network-wide packet forwarding process to validate the packet transmission path. Denote the path the packet pkt takes as $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_m$, where m is the length of the path. To validate the path that pkt has traversed, the controller computes the value of the validation field f_k. The notations in the controller are represented by adding a prime symbol to the corresponding notations used by the switch. For example, the computed result of the validation field is denoted as f_k'. The computation process can be expressed as

$$f_k' = \bigoplus_{j=1}^{m-1} \bigoplus_{i=1}^{j} s_i' \oplus \bigoplus_{j=1}^{m} (k_j' \oplus k_{in_j}'), \tag{4}$$

[1]We assume that the keys have the same length. Otherwise, we can pad the shorter keys with zeros without causing any problem.

where $\bigoplus_{i=1}^{n}$ represents the cumulative XOR operation. For example, $\bigoplus_{i=1}^{n} x_i = x_1 \oplus x_2 \oplus \cdots \oplus x_n$. The controller checks whether or not f_k and f_k' are equal. If yes, the validation succeeds; otherwise, the validation fails, implying the packet did not follow the designated path.

Correctness Analysis: If the packet follows the designated path, the validation field f_k received from the data plane should equal the validation field f_k' computed in the controller. With (2) and (3), the validation header in the data plane can be computed as

$$(f_s, f_k) = (s_1, k_{s_1})\vec{\oplus}(s_2, k_{s_2})\vec{\oplus}\cdots\vec{\oplus}(s_m, k_{s_m}), \tag{5}$$

where $k_{s_i} = k_i \oplus k_{in_i}$. According to (1) and (5), the validation field f_k can be computed as

$$f_k = \bigoplus_{j=1}^{m-1} \bigoplus_{i=1}^{j} s_i \oplus \bigoplus_{j=1}^{m} (k_j \oplus k_{in_j}). \tag{6}$$

Comparing (4) and (6), we can see that f_k is equal to f_k' if pkt follows the designated path.

Remark 1: The XOR operation prevents a switch from inferring the keys assigned to other switches. For instance, based on the value of f_k in (2), it is computationally infeasible [24] to infer the value of k_1 or the value of k_{in_1} since both k_1 and k_{in_1} are secure keys, which are considered as random bits.

V. Flow Path Validation with L-PVS

In packet path validation, the data plane must send a validation header to the controller for each packet, which significantly taxes switch-to-controller bandwidth. By extending packet path validation to network flow path validation, we validate entire flow sequences instead of individual packets, significantly reducing switch-to-controller bandwidth consumption. In this context, a network flow is defined as a sequence of packets that undergo processing based on the same sequence of flow rules.

A. Network Flow Path Validation Process

We denote the packets of the network flow as $pkt_1, pkt_2, ..., pkt_i, pkt_n$, and the designated path for this flow is $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_m$. In the network flow path validation process, the OutSwitch node S_m operates

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.

differently: Upon receiving a packet, S_m updates the validation field and keeps the validation header locally instead of sending it to the controller immediately.

Let VI^i denote the ith validation header stored at the OutSwitch node. We use subscripts to denote the different fields of VI. For example, VI^i_k represents the field in VI^i that corresponds to the validation field f^i_k in pkt_i. If the controller needs to validate the path of a network flow, it sends a validation request to the corresponding OutSwitch node. The OutSwitch node S_m forwards the locally stored validation information, VI^1, VI^2, ..., VI^n, to the controller. The request frequency is based on network needs. We adopt a time-slice polling method to send validation requests to the data plane within a specified time interval. For each validation information VI^i, the controller computes $VI^{i'}_k$ as

$$VI^{i'}_k = \bigoplus_{j=1}^{m-1}\bigoplus_{l=1}^{j} s'_l \oplus \bigoplus_{j=1}^{m}(k'_j \oplus k'_{in_j}). \tag{7}$$

The flow's forwarding path can be verified as

$$\text{Validation result} = \begin{cases}\text{success}, & \text{if } \sum_{i=1}^{n} VI^{i'}_k = \sum_{i=1}^{n} VI^i_k,\\ \text{failure}, & \text{otherwise},\end{cases} \tag{8}$$

where n denotes the total number of packets that have traversed the path.

Correctness Analysis: Validating the path of a network flow entails validating the path of all packets in the flow, that is,

$$\forall i \in [1,n], VI^{i'}_k = VI^i_k. \tag{9}$$

Clearly, if (9) is true, then the flow path validation succeeds. Nevertheless, we cannot claim the flow path validation fails if (9) is false. In other words, our flow path validation has no false negatives but may, *in theory*, have false positives, i.e., (9) is false, but the flow path validation succeeds.

The probability of false positives, however, is negligible. Denote the set of packets that do not follow their designated path as U. For each packet $i \in U$, we can calculate the difference between $VI^{i'}_k$ and VI^i_k, denoted as $\delta_i = VI^{i'}_k - VI^i_k$. Note that the value of each δ_i is considered to be a random number because the secure keys involved in (7). Then, we can split U into two subsets, U_1 and U_2, such that if $\delta_i > 0$ then packet $i \in U_1$ otherwise, packet $i \in U_2$. The only condition for false positives to occur is

$$\sum_{i \in U_1} \delta_i + \sum_{j \in U_2} \delta_j = 0. \tag{10}$$

Equation (10) implies that the sum of a sequence of random numbers equals the sum of another sequence of random numbers. The probability for (10) to hold is nearly zero. Indeed, we have never observed false positives in our later experiments.

B. Storage Optimization

The flow path validation method places a heavy storage burden on OutSwitch nodes by requiring them to store validation headers for all packets in a network flow. We propose a novel Storage Optimization (SO) method to compress the validation headers of flow packets into one validation information (VI). For this purpose, firstly, the controller assigns each path a prime number p and one of p's primitive element g. These values are sent to the corresponding OutSwitch node of the path. Next, we incorporate a validation field f_t into the validation header and modify the validation header initialization in (2) as

$$f_s \leftarrow s,\ f_t \leftarrow t,\ f_k \leftarrow k \oplus k_{in}, \tag{11}$$

where t is the timestamp of the switch when the packet was received. The operations of RelaySwitch nodes remain unchanged. Upon receiving a packet, the OutSwitch node updates its corresponding flow's validation information (VI). Denote the validation fields of the ith received packet pkt_i as f^i_s, f^i_k and f^i_t. The OutSwitch node first updates the validation fields f^i_s, f^i_k using (3) before sending pkt_i to the destination host. Subsequently, it updates the local validation information fields VI_s, VI_k and VI_t as

$$\begin{aligned} &VI_s \leftarrow f^i_s,\ VI_k \leftarrow (g^{(f^i_k+f^i_t)\%\varphi(p)}\%p + VI_k)\%p,\\ &VI_t \leftarrow (g^{f^i_t\%\varphi(p)}\%p + VI_t)\%p,\end{aligned} \tag{12}$$

where $\%$ represents the modulus operator, and VI_s, VI_k, and VI_t are the fields of VI, which are initialized to zero in the OutSwitch node. $\varphi(p)$ is the Euler's totient function [25] of p, which is utilized to reduce the exponential part in (12), thereby improving computational efficiency without altering the computing result.

When the controller needs to validate the path of a network flow, it sends a validation request to the corresponding OutSwitch node. The OutSwitch node then returns the locally stored validation information VI to the controller. Upon receiving VI, the controller computes the field VI'_k as

$$VI'_k = (VI_t * g^{(\Delta_m(s')\oplus k')\%\varphi(p)})\%p, \tag{13}$$

where m represents the number of switches in the path, $\Delta_m(s') = \bigoplus_{j=1}^{m-1}\bigoplus_{i=1}^{j} s'_i$, and $k' = \bigoplus_{j=1}^{m}(k'_j \oplus k'_{in_j})$. The flow path validation is expressed as

$$\text{Validation result} = \begin{cases}\text{success}, & \text{if } VI'_k = VI_k,\\ \text{failure}, & \text{otherwise.}\end{cases} \tag{14}$$

The above SO method significantly reduces the storage overhead in the OutSwitch, aligning switch storage overhead with network flows rather than individual packets. As a result, it also reduces the switch-to-controller bandwidth consumption while sending the validation information to the controller.

Correctness Analysis: Due to the space limit, we omit the detailed analysis, which is similar to the analysis in Section V-A but leverages the special features of modulus

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.

operation and the Euler's totient function [25]. To be more specific, these properties are

$$(x+y)\%p = (x\%p + y\%p)\%p, \tag{15}$$

$$(x*y)\%p = [(x\%p)*(y\%p)]\%p, \tag{16}$$

$$g^{x\%\varphi(p)}\%p = g^{x}\%p, \tag{17}$$

where x, y are positive integers, p is a prime number, g is one of p's primitive elements, and $\varphi(p)$ is the Euler's totient function. Similar to the analysis in Section V-A, the SO method also has false positives. The probability of false positives in the SO method is close to $\frac{1}{p}$, which is not an issue when p takes a large value.

C. Path Partition

To support large-scale deployment, we propose a path partition scheme that divides the path at intersections of multiple network flow paths. This enables merging and jointly validating overlapping sub-paths from different flows, thereby reducing both overall data plane storage overhead and the total number of paths requiring validation.

Firstly, we select some switches that handle a large number of network flows as KeySwitch nodes. This can be achieved by leveraging the controller's network-wide topology view. Then, the controller assigns a virtual key, denoted by k_v, to each KeySwitch node. Finally, we use KeySwitch nodes to partition the transmission paths of network flows. When a KeySwitch node S_i receives a packet pkt of a network flow, it firstly utilizes (3) to update the validation header of pkt and utilizes (12) to update the local validation information fields. Then the KeySwitch node uses k_v to initialize the validation header of pkt before sending it to the next hop:

$$f_t \leftarrow t,\; f_s \leftarrow s_i,\; f_k \leftarrow k_i \oplus k_v. \tag{18}$$

Example 1: We use a simple example to illustrate how path partitioning can reduce the storage overhead of the data plane. Consider a star network topology where the central switch S is connected to m edge switches $S_1, S_2, ..., S_m$, and the hosts connected to these edge switches communicate with each other. When validating the path for all network flows, each edge switch needs to validate $m-1$ paths. For example, the switch S_1 needs to validate paths $S_2 \rightarrow S \rightarrow S_1, S_3 \rightarrow S \rightarrow S_1, ..., S_m \rightarrow S \rightarrow S_1$. The total number of paths that need to be validated in the data plane is $m(m-1)$. By designating the switch S as the KeySwitch node and partitioning paths at this node, S needs to validate m paths $S_1 \rightarrow S, S_2 \rightarrow S, ..., S_m \rightarrow S$, while each edge switch only needs to validate one path, e.g., S_1 validates path $S \rightarrow S_1$. This way, the total number of paths to be validated is reduced to $2m$. Here, we have two different path validation policies. The term *path validation policy* defines which paths or path segments that need to be validated. We emphasize that **the paths under a path validation policy may not be the same as the paths used in actual data transmission**. For instance, the paths under the second path validation policy are: $\{S_1 \rightarrow S, S_2 \rightarrow S, \ldots, S_m \rightarrow S, S \rightarrow S_1, \ldots, S \rightarrow S_m\}$, while the actual paths used for data transmissions are $S_2 \rightarrow S \rightarrow S_1$ and so on.

We propose a Greedy-based KeySwitch node Selection algorithm (GKSS) to identify suitable KeySwitch nodes. Inspired by betweenness centrality [26], we introduce a novel metric in GKSS known as policy centrality (pc), which reflects the significance of a switch based on the total number of paths passing the switch under the *current path validation policy*. Let $pc(S)$ denote the number of paths traversing the switch S under the current path validation policy, i.e.,

$$pc(S) = \sum_{(e_i,e_j)} \sigma_{e_i,e_j}(S), \tag{19}$$

where m denotes the total number of edge switches, e_i and e_j represent the edge switches ($1 \leq i, j \leq m$), and function $\sigma_{e_i,e_j}(S)$ returns the number of paths from e_i to e_j passing through the switch S under *current path validation policy*. GKSS works as follows:

- **Step 1:** The controller computes the path set $\mathbb{P} = \bigcup_{\forall i,j}\{\mathcal{T}_{ij}\}$, where $\mathcal{T}_{ij}$ denotes the set of paths from the edge switch e_i to e_j in *the current validation policy*. Define a set $\mathbb{S}$ from $\mathbb{P}$ by

  $$\mathbb{S} = \{S | S \in \mathcal{T}, \mathcal{T} \in \mathbb{P}\}, \tag{20}$$

  where $S \in \mathcal{T}$ means that the path $\mathcal{T}$ passes through the switch S under the *current path validation policy*. Since initially $\mathbb{P}$ contains all paths, $\mathbb{S}$ initially includes all the switches (because otherwise, the switches not in $\mathbb{S}$ will not be used by any flow). For each switch $S \in \mathbb{S}$, we use set $\mathbb{T}_S$ to record all paths passing S under *the current validation policy*:

  $$\mathbb{T}_S = \{\mathcal{T} | \mathcal{T} \in \mathbb{P}, S \in \mathcal{T}\}. \tag{21}$$

- **Step 2:** We evaluate the switches in descending order of their $|\mathbb{T}_S|$ values to ascertain their suitability as KeySwitch nodes. Here, $|\mathbb{T}_S|$ equals the policy centrality value of S as defined in (19).
- **Step 3:** We select a switch as a KeySwitch node, one by one following the descending order in Step 2. We check if using the selected node as KeySwitch can reduce the total number of validation paths under the current path validation policy. If no, do nothing; otherwise, we add it to the KeySwitch set $\mathbb{V}$, and paths in $\mathbb{T}_S$ are then removed from the path set $\mathbb{P}$.
- **Step 4:** The controller then recalculates sets $\mathbb{S}$ and $\mathbb{T}_S$. This means the update of the current path validation policy. Go to Step 2. Repeat until none of the switches in $\mathbb{S}$ are suitable as KeySwitch nodes. The final KeySwitch nodes are recorded in set $\mathbb{V}$.

Since using all edge switch pairs in GKSS is computationally costly, we can sample edge switch pairs from the data plane using a uniform sampling method and use these samples to estimate the policy centralities of the data plane switches. The predicted policy centrality of the switch S is represented

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.

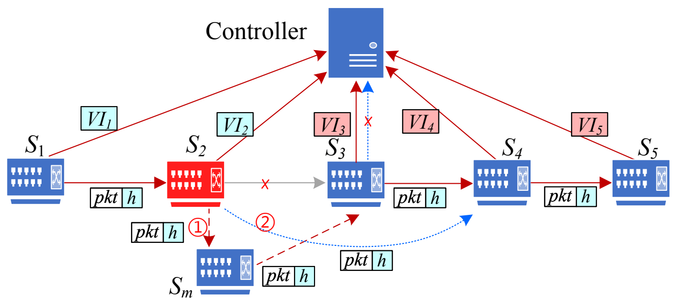


Fig. 4. An illustration of anomaly switch localization.

as $\tilde{pc}(S)$. As delineated in [27], to attain a prediction approximation $|pc(S) - \tilde{pc}(S)| \leq \epsilon$ with a probability of at least $1-\delta$, the required sample size N must satisfy:

$$N = \frac{c}{\epsilon}(d + \ln\frac{1}{\delta}), \tag{22}$$

where c is a positive constant, typically set to 0.5, and d represents the network diameter.

VI. Security Analysis and Anomaly Switch Localization

A. Security Analysis

A malicious switch may divert packets to alternate routes instead of the designated path. We assume that the final hop switch directs packets to the correct port. Otherwise, packets would not reach their intended destination, which would be quickly detected by end-to-end transmission protocols.

Path detour: As shown in Fig. 4 by the red dashed lines, S_2 forwards the packet to S_m instead of S_3. Subsequently, S_m transmits the packet to S_3. No matter whether or not S_m follows the validation header update defined by L-PVS, the path validation in the controller will fail because S_3's header update has involved an incorrect incoming port.

Switch bypass: As shown in Fig. 4 by the blue dotted lines, S_2 forwards the packet directly to S_4, bypassing S_3. Consequently, the validation header lacks information about S_3, resulting in a path validation failure in the controller.

Out-of-order traversal: Suppose that the intended forwarding path for a packet is $S_1 \to S_2 \to S_3 \to S_4 \to S_5$. However, the packet follows the path $S_1 \to \mathbf{S_3} \to \mathbf{S_2} \to S_4 \to S_5$ instead. Since the order of operands affects the result of the validation information in (1), the altered switch processing order results in different values in the validation header, leading to a path validation failure in the controller.

Path forging: If the switch S_2 forwards the packet to switch S_m, and the packet doesn't return to the original forwarding path until the last hop. The altered path will produce distinct values in the validation header, leading to a path validation failure in the controller.

B. Anomaly Switch Localization

A path validation failure in the controller indicates that some packets violated the network forwarding policy. We utilize temporary KeySwitch (TKeySwitch) nodes to locate

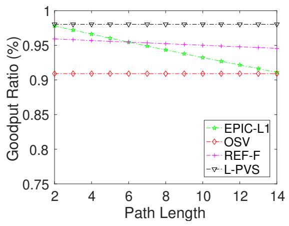


Fig. 5. Packet goodput ratio with the path length (packet payload = 800 bytes).

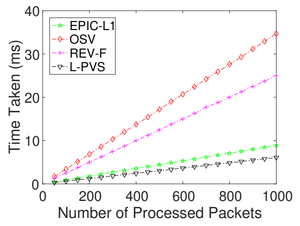


Fig. 6. Extra packet processing time in a path.

the anomaly switches. Compared with a KeySwitch node, a TKeySwitch node only cares about packets that belong to a specific path rather than all packets passing through the switch.

To locate the anomaly switch, the controller sets all the switches on the anomalous path as TKeySwitch nodes (referring to S_1, S_2, S_3, S_4, S_5 in Fig. 4), constructs a test packet that can be transmitted on this path, and sends it to the first hop switch. The validation header of the test packet is initialized and only known to the controller. When a switch receives the test packet, it updates the packet validation header and the local validation information using (3) and (12), respectively. Then it delivers the packet to the next hop while sending the local validation information (VI) to the controller. The controller employs (13) and (14) to validate each VI.

In Fig. 4 indicated by the red dashed lines, S_2 did not send the packet to the designated next hop S_3 but to S_m instead. This results in the validation failure of VI_3, VI_4, VI_5 at the controller. Based on the previous analysis, the controller can locate the anomaly switch as the last switch that has correct validation information (referring to S_2 in Fig. 4). In the case of switch bypass or path forging, the skipped switches (referring to S_3 in Fig. 4) will not receive the test packet and will not send their validation information to the controller. So the controller sets a timer to detect if it has not received validation information from a TKeySwitch node within a specified timeout period. The timer can be set to the sum of the current network's RTT between the controller and a TKeySwitch node plus the RTT of the path. Missing validation information by the timeout indicates a validation failure.

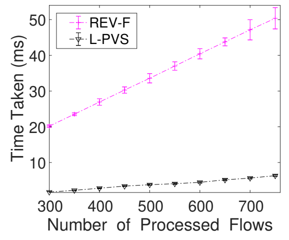
The above process is denoted as a localization round. To check whether there are more anomaly switches on this path, the controller extracts a sub-path from the next hop of the anomaly switch to the last hop switch and starts a new localization round on this sub-path (referring to $S_3 \to S_4 \to S_5$ in Fig. 4). The process of anomaly localization terminates, when all information received by the controller is successfully validated, or when only one switch remains in the sub-path.

VII. Implementation and Evaluation

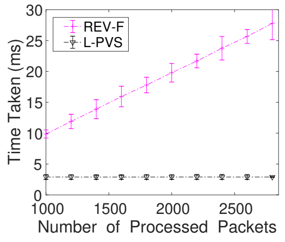
We conduct extensive experiments to evaluate the performance of L-PVS. We implement the data plane functions of L-PVS using Open vSwitch version 2.13.7 [28], while the control plane functions are implemented in RYU [29]. We emulate[2]

[2]We did not use P4 switches for our experiments because the cost is too high.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.



(a) Validation time with the number of flows.



(b) Validation time with the number of packets.

Fig. 7. Time taken for path validation in the controller.

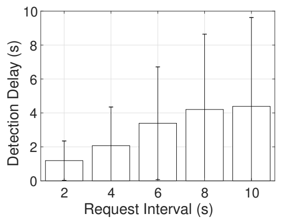


Fig. 8. Average detection delay with validation request intervals.

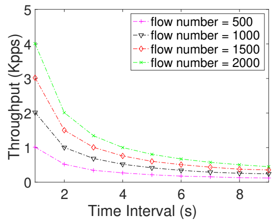


Fig. 9. Switch-to-controller link throughput with validation request intervals.

SDNs on an Ubuntu server with an Intel(R) Xeon(R) Gold 5117 CPU @ 2.00GHz and 32G RAM. We utilize several topologies from the Internet Topology Zoo [30], an ongoing project that gathers over 250 real-world network topologies. The validation header consists of 16 bytes, encompassing f_s with an allocation of 2 bytes, f_t with 4 bytes, f_k with 8 bytes, and an extra field len of 2 bytes to signify the length of the validation header.

A. Validation Header Size Comparison

TABLE II
VALIDATION HEADER SIZE COMPARISON

	OSV	**REV-F**	**EPIC-L1**	**L-PVS**
Size (byte)	constant	$m + 32$	$5m + 8$	constant
$m = 2$	80	34	18	16
$m = 6$	80	38	38	16
$m = 10$	80	42	58	16
$m = 14$	80	46	78	16

We compare the validation header size between L-PVS, OSV [31], flow-level REV (REF-F) [22], and level 1 EPIC (EPIC-L1) [4]. The comparison results are presented in Table II, where the path length is denoted by m. Using REV-F as a baseline, L-PVS achieves reductions of 52%, 61%, and 65% in the validation header size for path lengths of 6, 10, and 14 hops, respectively. The comparison results of the packet goodput ratio[3] are shown in Fig. 5. Overall, **L-PVS has the smallest validation header size and achieves the highest packet goodput ratio.**

[3]The packet goodput ratio is defined as the ratio of payload size over the total size of the payload and validation header. Here, the payload is a general term referring to all content in the packet excluding the validation header.

B. Processing Delay Comparison

1) Extra Packet Processing Delay in a Path: During the packet forwarding process, updating the validation header requires extra packet processing time in the switch. We compare the extra packet processing time of different schemes under the same hardware environment. In L-PVS, the time includes updating the validation header and storing the validation headers locally. The path length is set to 10 (similar results were observed for other path lengths), and the prime number p in L-PVS is set to 1000003, ensuring a false positive rate lower than $\frac{1}{10^6}$.

The comparison results in Fig. 6 demonstrate that our method surpasses so far the fastest path validation scheme by achieving **about** 25% **reduction in validation header processing time** when considering 1000 processed packets. The observed improvement is due to our method's use of an exponentiation-and-modulo operation at the last hop switch and leveraging Euler's totient function for faster validation processing. In contrast, encryption-based methods need encryption at every hop, increasing computational overhead and time consumption.

2) Time Consumption in the Controller: We compare the validation time consumption in the controller between L-PVS and REV-F. We do not consider EPIC and OSV since they are designed for traditional networks that do not have a controller. The path length is set to 10. Fig. 7(a) shows that the validation time varies with the number of network flows, each with 10 packets. Fig. 7(b) shows that the validation time varies with the number of packets when the number of flows is set to 500. The results demonstrate that our method is faster. This is because our SO method precomputes the validation information in the data plane, and the precomputed results can be directly used by the controller in the validation process, improving the validation speed. In addition, with the SO method, the controller only needs to perform path validation once for each flow path. The utilization of Euler's totient function also helps in improving the computation speed.

C. Ablation Study

1) Impact of Validation Request Interval on Detection Delay and Switch-to-Controller Link Throughput: Fig. 8 shows that the average controller detection delay increases with longer request intervals. Fig. 9 demonstrates a decrease in switch-to-controller link throughput with longer request intervals, attributed to fewer requests per unit time due to our polling-based validation request from the data plane.

2) Impact of Path Partitioning on Data Plane Storage Overhead: Fig. 10 presents the storage overhead of the entire data plane under three scenarios: without path partition (Without GKSS), utilizing sampled edge switch pairs to compute KeySwitch nodes (With GKSS), and utilizing all switch pairs to compute KeySwitch nodes (Optimal). We used Chinanet, Cynet, IBM, and Oxford topologies [30], and calculated the diameters of these topologies using the method proposed in [27]. We assign one host to each switch and enable them to communicate with each other to generate network flows.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.

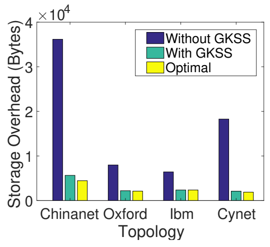


Fig. 10. The storage overhead of the data plane.

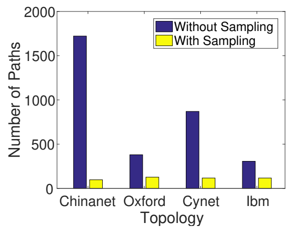


Fig. 11. The number of paths required to compute KeySwitch nodes.

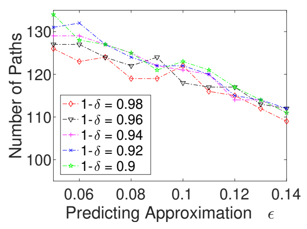


Fig. 12. The number of paths with sampling parameters.

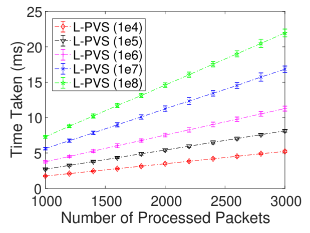
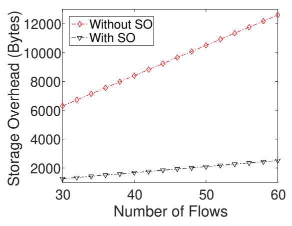
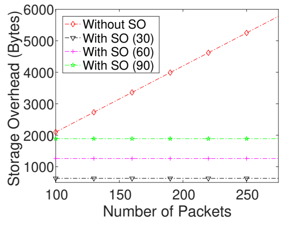


Fig. 13. Extra packet processing time. “L-PVS (1e4)” means that the prime value p is set to the smallest prime number greater than 10^4.



(a) Storage overhead with the number of flows, each with 10 packets.



(b) Storage overhead with the number of packets. “With SO (30)” means that we adopted SO and the number of network flows is 30.

Fig. 14. Storage overhead of the OutSwitch node.

The predicting approximation ϵ and predicting accuracy $1-\delta$ in (22) is set to 0.05 and 0.99, respectively. The results shown in Fig. 10 suggest that our path partitioning method can significantly reduce the data plane storage cost.

3) Benefit of Sampling in GKSS: Fig. 11 displays the number of paths required in the GKSS algorithm with and without the sampling method. The predicting approximation ϵ is set to 0.05 and the predicting accuracy $1-\delta$ is set to 0.99. The results in Fig. 11 indicate that the sampling method can significantly reduce the number of paths required in GKSS. Fig. 10 indicates that the sampling-based GKSS approach achieves comparable storage overhead to the optimal GKSS approach. Combining Figs. 10 and 11, we can conclude that the sampling method reduces the computation overhead without obvious degradation in path partition performance.

Fig. 12 shows the number of paths that need to be validated for different predicting accuracies $(1-\delta)$ and different predicting approximations (ϵ) over the Oxford topology. We omit the results for other topologies to save space but point out that other topologies lead to a similar trend. The results indicate that the number of required paths is mainly determined by the predicting approximation ϵ, while the impact of predicting accuracy $1-\delta$ is not significant.

4) Impact of Different Prime Values (p) on Extra Packet Processing Time: Fig. 13 shows the extra packet processing time in an OutSwitch node with different prime values. A larger p offers a lower false positive rate but results in longer validation information processing time. The results in Fig. 13 indicates that L-PVS (1e6) can achieve low extra packet processing delay and low false positive rate simultaneously.

5) Impact of Storage Optimization (SO): Fig. 14(a) shows that the storage overhead increases with increased number of network flows. Fig. 14(b) demonstrates how the storage overhead varies with the number of processed packets. From the two figures, we conclude that SO leads to significant storage savings in the OutSwitch node. This is attributed to the precomputation of validation information by the OutSwitch node during the SO process, which compresses all validation headers within a network flow into one validation information. In other words, with SO, the storage cost is purely determined by the number of flows rather than the number of packets.

VIII. Conclusion

We have presented L-PVS, a novel approach for path validation in SDN, which offers several key features to enhance the path validation process: (1) **Constant validation header**– This is achieved through the utilization of a novel asymmetric additive operation. (2) **Path validation at the flow level**– Building upon packet path validation, we extended our approach to network flow path validation by temporarily storing the validation information at the last hop switch along the path. To reduce storage cost at the last hop switch, we proposed a storage optimization (SO) method, which retains only one compressed validation information for each network flow path rather than storing information for each individual packet path. (3) **Path partitioning**– This method enables concurrent validation of multiple overlapping paths, reducing switching storage overhead overall. To identify suitable switches for path partitioning, we proposed GKSS, a greedy-based method and designed a sampling technique that further reduces the computational overhead of GKSS. (4) **Anomaly switch localization**– We have proposed a method to locate anomaly switches when path validation fails at the controller.

Compared to the baselines, L-PVS consumes a constant storage overhead in the data plane and a constant verification cost in the controller for a network flow. It achieves reductions of 52%, 61%, and 65% in the validation header size for path lengths of 6, 10, and 14 hops, respectively. Furthermore, L-PVS surpasses the existing fastest path validation scheme by achieving a 25% reduction in validation header processing time during the transmission process when considering a path length of 10 and 1000 processed packets.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.

References

[1] G. Chalhoub and A. Martin, "But is it exploitable? exploring how router vendors manage and patch security vulnerabilities in consumer-grade routers," in *Proceedings of the 2023 European Symposium on Usable Security, Copenhagen, Denmark*. ACM, 2023, pp. 277–295.

[2] Y. Zhang, W. Huo, K. Jian, J. Shi, H. Lu, L. Liu, C. Wang, D. Sun, C. Zhang, and B. Liu, "Srfuzzer: an automatic fuzzing framework for physical SOHO router devices to discover multi-type vulnerabilities," in *Proceedings of the 35th Annual Computer Security Applications Conference , San Juan, PR, USA*. ACM, 2019, pp. 544–556.

[3] K. Bu, A. Laird, Y. Yang, L. Cheng, J. Luo, Y. Li, and K. Ren, "Unveiling the mystery of internet packet forwarding: A survey of network path validation," *ACM Comput. Surv.*, vol. 53, no. 5, pp. 104:1–104:34, 2021.

[4] M. Legner, T. Klenze, M. Wyss, C. Sprenger, and A. Perrig, "EPIC: every packet is checked in the data plane of a path-aware internet," in *29th USENIX Security Symposium*, 2020, pp. 541–558.

[5] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, "Verifying and enforcing network paths with icing," in *Proceedings of ACM CoNEXT, Tokyo, Japan*, 2011, pp. 30–42.

[6] T. H. Kim, C. Basescu, L. Jia, S. B. Lee, Y. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *ACM SIGCOMM 2014 Conference, Chicago, IL, USA*, 2014, pp. 271–282.

[7] A. He, K. Bu, Y. Li, E. Chida, Q. Gu, and K. Ren, "Atomos: constant-size path validation proof," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 3832–3847, 2020.

[8] B. Wu, K. Xu, Q. Li, Z. Liu, Y. Hu, M. J. Reed, M. Shen, and F. Yang, "Enabling efficient source and path verification via probabilistic packet marking," in *26th IEEE/ACM IWQoS, Banff, Canada*, 2018, pp. 1–10.

[9] A. He, X. Li, J. Fu, H. Hu, K. Bu, C. Miao, and K. Ren, "Hummingbird: Dynamic path validation with hidden equal-probability sampling," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 1268–1282, 2023.

[10] B. Hu, Y. Bi, M. Zhi, K. Zhang, F. Yan, Q. Zhang, and Z. Liu, "A deep one-class intrusion detection scheme in software-defined industrial networks," *IEEE Trans. Ind. Informatics*, vol. 18, no. 6, pp. 4286–4296, 2022.

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

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

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[14] M. Antikainen, T. Aura, and M. Särelä, "Spook in your network: attacking an SDN with a compromised openflow switch," ser. Lecture Notes in Computer Science, vol. 8788, 2014, pp. 229–244.

[15] P. Chi, C. Kuo, J. Guo, and C. Lei, "How to detect a compromised SDN switch," in *Proceedings of the 1st IEEE Conference on Network Softwarization, NetSoft, London, United Kingdom*, 2015, pp. 1–6.

[16] C. Pang, Y. Jiang, and Q. Li, "FADE: detecting forwarding anomaly in software-defined networks," in *ICC, Kuala Lumpur, Malaysia*, 2016, pp. 1–6.

[17] Q. Li, Y. Liu, Z. Liu, P. Zhang, and C. Pang, "Efficient forwarding anomaly detection in software-defined networks," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 11, pp. 2676–2690, 2021.

[18] P. Zhang, F. Zhang, S. Xu, Z. Yang, H. Li, Q. Li, H. Wang, C. Shen, and C. Hu, "Network-wide forwarding anomaly detection and localization in software defined networks," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 332–345, 2021.

[19] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "Sdnsec: forwarding accountability for the SDN data plane," in *25th International Conference on Computer Communication and Networks, ICCCN, Waikoloa, HI, USA*, 2016, pp. 1–10.

[20] S. Xi, K. Bu, W. Mao, X. Zhang, K. Ren, and X. Ren, "Ruleout forwarding anomalies for SDN," *IEEE/ACM Trans. Netw.*, vol. 31, no. 1, pp. 395–407, 2023.

[21] P. Zhang, "Towards rule enforcement verification for software defined networks," in *INFOCOM, Atlanta, USA*, 2017, pp. 1–9.

[22] P. Zhang, H. Wu, D. Zhang, and Q. Li, "Verifying rule enforcement in software defined networks with REV," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 917–929, 2020.

[23] A. Nehra, M. Tripathi, and M. S. Gaur, "Global view in sdn: existing implementation, vulnerabilities & threats," in *Proceedings of the 10th International Conference on Security of Information and Networks, SIN, Jaipur, IN, India*, 2017, pp. 303–306.

[24] F. M. LEWIN. (2012) All about xor. [Online]. Available: https://accu.org/journals/overload/20/109/lewin_1915/

[25] B. Kaliski, "Euler's totient function," in *Encyclopedia of Cryptography and Security, 2nd Ed*, 2011, p. 430.

[26] T. Hayashi, T. Akiba, and Y. Yoshida, "Fully dynamic betweenness centrality maintenance on massive networks," *Proc. VLDB Endow.*, vol. 9, no. 2, pp. 48–59, 2015.

[27] M. Riondato and E. M. Kornaropoulos, "Fast approximation of betweenness centrality through sampling," *Data Min. Knowl. Discov.*, vol. 30, no. 2, pp. 438–475, 2016.

[28] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of Open vSwitch," *login Usenix Mag.*, vol. 40, no. 2, 2015.

[29] R. P. Team *et al.*, *RYU SDN Framework-English Edition*. RYU project team, 2014. [Online]. Available: https://ryu.readthedocs.io/en/latest/

[30] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[31] H. Cai and T. Wolf, "Source authentication and path validation in networks using orthogonal sequences," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2016, pp. 1–10.

Authorized licensed use limited to: UNIVERSIDADE FEDERAL DE OURO PRETO. Downloaded on February 10,2025 at 19:32:22 UTC from IEEE Xplore. Restrictions apply.