TopoTrust: A Blockchain-based Trustless and Secure Topology Discovery in SDNs

Mohamed Lamine Adjou*, Chafika Benzaïd†, and Tarik Taleb†
* University of Sciences and Technology Houari Boumediene, Algiers, Algeria
† University of Oulu, Oulu, Finland
Emails: madjou@usthb.dz, firstname.lastname@oulu.fi

***Abstract*—The Software Defined Network (SDN) architecture decouples the control functionality from the forwarding devices and implements it in a separate entity known as the controller. This raises new concerns on securing the control messages exchanged between the controller and the forwarding devices. In this paper, we propose *TopoTrust*, a novel fully trustless authenticity and integrity verification mechanism that relies on a Blockchain protocol to detect network topology poisoning attacks, namely Host Tracking Service (HTS) and OpenFlow Discovery Protocol (OFDP). The key merit of TopoTrust is its ability to operate in a zero trust SDN environment where no controller or switch is trusted. The evaluation of our protocol shows that it can successfully detect any spoofing-based and packet tampering attacks; and up to 96% and 100% of Fast Relocation and Link Fabrication attacks respectively within a short detection time, while introducing small overhead to the network.**

***Index Terms*—SDN, Security, Topology Discovery, Blockchain, and Trust.**

I. Introduction

Software-Defined Networking (SDN) came as a response to the rapid increase in network complexity and the emergence of new technologies that require on-demand and application-specific network resource allocation, such as Deterministic Networks (DetNet) [1] and 5G networks. The aim of SDN is to simplify the provisioning and distribution of those resources, thus reducing the capital and operational expenditure [2].

SDN enables unified network management and visibility by introducing centralized control with a special entity called "the controller", but it also creates new attack vectors compared to traditional networks [3]. One of the most dangerous ones is Topology Poisoning, which can have devastating consequences, especially on networks that require very accurate information in order to make critical decisions. The most prominent example is arguably DetNet [4], which is an ongoing effort by the IETF to provide bounds on latency, jitter, packet loss, and bandwidth, in order to transport critical data streams such as industrial processes and machine control, and vehicles.

Topology Poisoning is achieved by altering the discovery processes of the network topology and hosts location, namely, *OpenFlow Discovery Protocol* (OFDP) and *Host Tracking Service* (HTS). Both protocols are inherently non-authenticated [5], and if targeted, the integrity and availability of the network would be at risk.

Several solutions have been proposed regarding Topology Poisoning, some focused on securing the forwarding rules by analyzing flow statistics [6] and flow rules [7], or by using probing techniques [8] [9]. Other solutions addressed specific attacks such as Link Fabrication attack [10]. These solutions focus on specific protocols rather than Topology Poisoning as a whole. Furthermore, they rely on the fact that the network nodes are trusted in regard of the controller which turns it into a single point of failure that runs all the verification processes. In [11], a Blockchain network is used between the SDN controllers to reach consensus on network information; whereas [12] introduces special nodes to verify its integrity. The major drawback of these solutions is that they only include the controllers or special nodes in the control process, which leaves the forwarding devices unverified.

In this paper, we propose TopoTrust, a fully trustless and lightweight verification mechanism based on Blockchain, that detects attacks targeting the topology discovery and host tracking protocols in SDN. A Blockchain network that connects all active nodes (controller and forwarding devices) is introduced, so they can exchange verification information and detect attacks. This solution provides continuous monitoring and analysis of topology-related exchanges in an automatic and distributed manner, which is a particularly crucial task in modern DetNet use cases, such as Industrial machine to machine (M2M), Professional audio and video, among others. The simulation of multiple attack scenarios shows promising results in terms of detection time and detection rate.

The main contributions of this work can be summarized as follows:

- Highlighting the fragility of topology discovery and trust mechanisms in SDN.
- TopoTrust is proposed, a fully integrated and trustless solution to detect Topology Poisoning attacks. The solution is based on a custom implementation of Blockchain that accentuates zero-trust, resiliency, and efficiency.
- Finally, we evaluate TopoTrust in multiple attack scenarios. The obtained results show the high effectiveness of TopoTrust in detecting topology Poisoning attacks with reduced overhead.

The remainder of this paper is organized as follows. Section II provides background information on SDN, topology discovery, and Blockchain, and presents an overview of the related solutions. Section III describes the threat model, followed by Section IV wherein a detailed explanation of TopoTrust

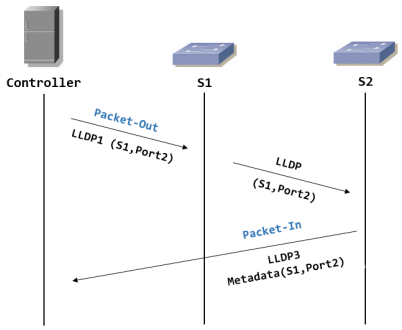


Fig. 1: LLDP discovery steps.

and a security analysis are presented. Section V is dedicated to the implementation and the performance evaluation of the TopoTrust protocol. The paper concludes in Section VI.

II. Related Work

A. Background

1) SDN and Topology Discovery: The SDN controller enables the control of a network through a software acting as the control plane of the entire network. The control plane communicates with the forwarding devices (the data plane) through southbound Application Programming Interfaces (APIs). The defacto control protocol that is implemented in nearly every SDN environment is OpenFlow [13]. The controller also provides abstracted controls using northbound APIs to applications (the application plane). The data plane reports information and statistics regarding the network to the control plane, based on which, the control plane updates its perception of the network state, and responds with the proper instructions in compliance with the application plane policies.

To discover the network, the control plane relies on two main protocols: OFDP for gathering switch neighboring information, and HTS for host discovery and location.

- *Switch neighboring discovery:* OpenFlow controllers use OFDP which is based on standard Link Layer Discovery Protocol (LLDP) to discover forwarding device interconnection. To discover a link between switches S1 and S2, as illustrated in Fig. 1, (1) the controller sends an LLDP packet with S1 ID into a *packet-out* message to S1, (2) the switch S1 then broadcasts this LLDP message to its neighbors. (3) Finally, when S2 receives S1's LLDP, it encapsulates an LLDP packet to send to the controller as *packet-in* the *ChassisID* (i.e., neighbor's ID) and *PortID* which is the neighbor's port from which it sent the LLDP packet. Accordingly, the SDN controller discovers a unidirectional link between S1 and S2.
- *Host Tracking Service:* When a switch receives an IP packet, it refers to its local flow table which associates various flow attributes (e.g., IP source/destination, VLAN ID, and source interface) with actions (e.g., forward, send-to-controller, and drop) to apply on this packet. When no flow rule is matched, the switch encapsulates this packet in a *packet-in* message and sends it to the controller to figure out which action to undertake. The controller then responds with a *packet-out* message containing a FlowMod instruction to add the corresponding flow rule. The controller will then infer that the same IP as the one encapsulated in the *packet-in* as a source IP is reachable via the switch's source interface.

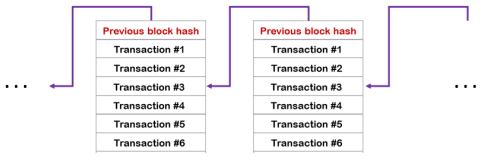


Fig. 2: Blockchain format.

2) Blockchain: Blockchain technology was first introduced as the underlying protocol enabling the first cryptographic monetary system Bitcoin [14]. It provides a zero-trust mechanism that allows secure transactions without the need for a central node; it achieves this by recording all the transactions in a set of digitally-signed blocks [15]. In addition to those transactions, each block contains a hash calculated from the previous block hash and the set of transactions [16]. When the block hash is calculated, the new block is broadcast by the miner to all participating nodes so they can append it to the rest of the blocks (See Fig. 2).

B. Related Solutions

SDN relies heavily on the communication between the control plane and the data plane to effectively draw the correct forwarding state of the network. Consequently, various solutions were developed to secure this exchange. In this section, we discuss and illustrate the main approaches applied therein, categorizing them as 1) Blockchain-less solutions and 2) Blockchain-based solutions.

1) Blockchain-less solutions: The work in SPHINX [7] provides a verification module that analyzes OpenFlow messages and flow statistics, in order to detect topology poisoning attacks. TopoGuard [10] detects Link Fabrication attacks by identifying the type of device connected to a switch using probing techniques and authenticated LLDP packets. SDNsec [9] uses a symmetric key between the switches and the controller to ensure the integrity of forwarding rules. It also embeds a cryptographic tag into every packet of the flow to make sure the packets follow the intended path. DYNAPFV [6] uses flow statistics and *packet-in* hashes collected from switches to detect packet dropping and tampering attacks, thus ensuring that the flows do not deviate from the original path. SDN traceroute [8] uses a probing technique by injecting small packets into the network which triggers a *packet-in* chain reaction for active monitoring of the forwarding state.

2) Blockchain-based solutions: The solution named DistBlockNet [11] uses a Blockchain network between multiple controllers to verify the latest flow rules for a synchronous view of all networks to each controller. The approach used in BLOSTER [17] considers a SDN controller with access granted only to an administrator (i.e., the only entity with the right to write to the Blockchain); a vSwitch node which updates the flow rules and saves the changes in a log file; and a

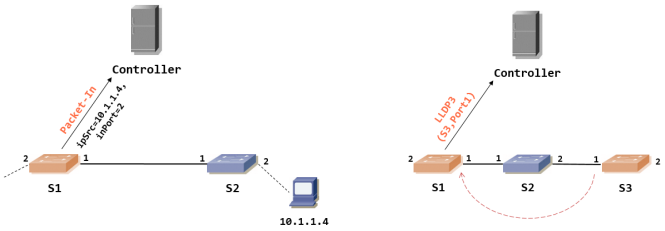
Firewall that checks and compares the flow rules issued by the controller from the Blockchain, with the ones updated in the vSwitch to detect attacks. In [12], the authors propose a trust evaluation system to determine whether a node in the network is reliable by issuing a trust weight from special nodes called "Verifiers". Those nodes share a Blockchain for storing, trust values, historical data, and authorization of new users.

Discussion: The solutions based on collecting switches' statistics as well as probing techniques rely on the fact that the switch-controller connection is established using TLS connection which OpenFlow specification provides. However, the activation of TLS is specified as 'optional' which can be problematic. Moreover, the implementation of TLS is found to be vulnerable to Man-in-the-Middle (POODLE) attack [18]. Finally, the controller still constitutes a single point of failure of the network, meaning that if an attacker successfully intercepts the encryption keys, (s)he can basically impersonate it and wreak havoc into the network. The Blockchain-based solutions do share the verification process over multiple controller instances or dispense it to special nodes, but they technically have to trust the information received from the data plane. Unlike existing solutions that are controller-centric, TopoTrust introduces a novel verification mechanism that involves all active network nodes using a custom-built Blockchain protocol, in such a way that no node (i.e., controller or switch) is individually trusted, and every node can potentially detect attacks. To the best of our knowledge, this is the first work involving Blockchain-based consensus to detect topology poisoning attacks in a fully integrated solution.

III. Threat Model

We consider an attacker who tries to poison the controller's topology view of the network by injecting false flow rules into the data plane to redirect traffic to their desired destination, and from there perform more sophisticated attacks such as: denial of service (by dropping the redirected traffic), data exfiltration (by redirecting traffic through a network gateway), and packet tampering (as the attacker places himself as Man-in-the-Middle). The attacker is assumed to have full knowledge and access to the trusted segments of the network, and can intercept and modify control messages sent from/to the controller, following in this way a "zero-trust" security model [19]. To achieve this goal, the attacker will have to disguise his identity as one or more legitimate nodes of the network (identity spoofing). He can achieve this in one of three ways: i) the first is to join the network as a new node (switch or controller) and establish a connection with the target node, ii) the second is to take control of an already authenticated node, and iii) the third is by intercepting a switch-controller connection and spoof the identity of one or the other. In this last instance, two cases can be envisioned:

- **In case of switch identity spoofing:** the attacker can fabricate *packet-in* messages and send them to the controller to indirectly change the controller's HTS information (See Fig. 3(a)). The attacker can also tamper with the LLDP packets to disrupt with the controller's neighboring information (See Fig. 3(b)).
- **In case of controller identity spoofing:** The attacker can send fake *packet-out* messages containing FlowMod instructions and thus build fraudulent flow rules directly into the targeted switch (See Fig. 4).



(a) Packet-In request tampering (b) LLDP tampering

Fig. 3: Case of a malicious switch.

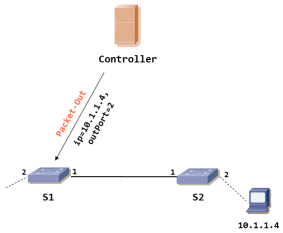


Fig. 4: Case of a malicious controller.

IV. TopoTrust: Proposed Solution

A. Overview

We consider a Blockchain network composed of an OpenFlow-enabled controller managing multiple switches; each node is associated with a pair of ECDSA [20] keys and gets its public key broadcast through the Blockchain network by the controller to all nodes after manual approval by the admin, in order to ensure the authenticity of control messages (transactions). TopoTrust algorithm works in four main steps:

1) *Transaction broadcast*: When a node sends a *packet-in*/*packet-out* message, it also broadcasts a small-sized version of the packet containing only necessary information (switch ID, source IP, etc.) in addition to its signature on the Blockchain network to every online node of the network, thus ensuring that every node keeps the same list of control messages sent over the SDN network;
2) *Block creation*: When certain conditions are met, every node generates a block and calculates the hash of n transactions locally, including the hash of the last block, to ensure the integrity of the transactions (process detailed in Section V-A). After that, a randomly selected node (the miner) broadcasts this hash to every node for comparison;
3) *Hash verification*: Every node checks the correctness of the received hash compared to the one it calculated; if an error is detected, it notifies the admin for possible integrity breach;
4) *Topology verification*: If the hash is correct, the node presumes that the transactions contained in the block are

valid, it then proceeds to topology verification in order to detect other attacks (detailed in Section V-A).

If some nodes went offline for a certain period of time due to accidental disconnections or attacks, they will be detected since they establish a connection with other nodes in the Blockchain network, and therefore will not be taken into account in the verification process till the next hash broadcast. If the the miner has missed some transactions, then it will not broadcast the verification hash; the other nodes will wait *m* seconds counting from the timestamp of the last received transaction and designate another miner.

B. Security Analysis

TopoTrust is based on consensus (i.e., attacks are detected if a majority declares so), so we assume that an attacker cannot compromise more than half of the network's nodes and a minimum of two switches composing the data plane are required in order to reach a majority. Furthermore, we assume that the network nodes are relatively stable; i.e., the majority of nodes are not disconnected at the same time. We consider an attacker who wants to manipulate LLDP and FlowMod messages as described in the threat model (Section III) in a SDN network that implements TopoTrust. The detection model consists of the following parties:

1) The asset being protected: LLDP and FlowMod messages sent by network nodes (controller or switch);
2) The attacker: knows and is connected to the internal network, can intercept those messages, and can spoof the identity of network nodes;
3) The protecting function: consensus-based verification of control messages by all nodes using secure hash exchanges through the Blockchain overlay. The consistency of the hash represents the proof that all transactions in the current and the previous blocks are valid; the blocks are deleted as soon as the hash is calculated.

This approach achieves authenticity (all nodes' identity is verified), integrity (with the hash), and availability (nodes can notify if disconnected with each other). In the following, various attack vectors and their mitigation method are detailed:

1) Attacker fabricates messages (without secret key): Whether the attacker spoofs the controller or switch identity, this attack is quickly detected since the attacker will not have the secret key of the spoofed node, and therefore cannot broadcast messages through the Blockchain network. As a result, spoofing-based attacks will fail.

2) Attacker joins the network as a new node (with self-generated secret key): This case is also easily detectable as the public keys of all legitimate nodes are known to all other nodes at deployment time (see Section IV-A).

3) Attacker takes control of one or multiple legitimate nodes (retrieves other nodes' secret key): In this case, the attacker can broadcast with the legitimate node's private key signature, to all nodes or only a few selected ones through the Blockchain network. In such event, we have three distinct cases:

- **In case of controller**: The attacker can then send fake FlowMod messages to selected switches and broadcast this event in the Blockchain network. This attack is detected if the majority of nodes report that there is no corresponding *packet-in* message broadcast previously.
- **In case of one switch**: Where an attacker can broadcast fake *packet-in* in the Blockchain network. This attack is detected if the majority of nodes report that there are no similar *packet-in* messages broadcast in the same path; the switches can verify this information from their *Complete-View* graph (more details in Section V). Attacks involving LLDP responses to the controller can also be detected in the same manner as the corresponding LLDP request will be reported missing by the majority of nodes.
- **In case of multiple switches**: This situation is much harder to achieve if TopoTrust is implemented since the attacker must hijack at least two switches for their attacks to take place. If (s)he succeeds, (s)he can forge links (i.e., Link Fabrication attack) between switches by tunneling LLDP packets from one switch to another. (S)he can also achieve Host Location Hijacking attack which consists of mimicking a legitimate host traffic in a different location to get the controller to believe that the host has moved to that location. In this case, to detect Link Fabrication attack, the other nodes can calculate timestamps between the first LLDP request and the received response, and compare it to a normal one since tunneling the information takes significantly more time to reach its destination. The same approach is used to detect Host Location Hijacking as a "fast" relocation is considered a probable attack.

V. Implementation and Performance Evaluation

A. TopoTrust Implementation

We implemented TopoTrust using a lightweight module integrated directly into the network's switches and the SDN controller. A SDN network is created with Mininet to simulate a 1000 hosts/vSwitch switches connected to Floodlight [21] controller, using the Openflow protocol for control plane exchanges. We used a VM with 4-core CPU and 32Go of RAM deployed on Cisco UCS chassis [22], simulating a medium data center within a single Blockchain network. The verification module listens to four types of messages, and broadcasts them in the Blockchain network as reduced-size transactions:

1) OFPacketInNoMatch (91 bytes): is a *packet-in* sent when a switch receives a flow that does not match any flow rule.
2) OFPacketOutFlowMod (108 bytes): is a *packet-out* message sent to a switch in response to a corresponding OFPacketInNoMatch.
3) OFPacketOutLLDP1 (75 bytes): is a *packet-out* message sent to a switch containing the switch's *ChassisID*.
4) OFPacketInLLDP3 (77 bytes): is a *packet-in* message sent to the controller when a new link is discovered.

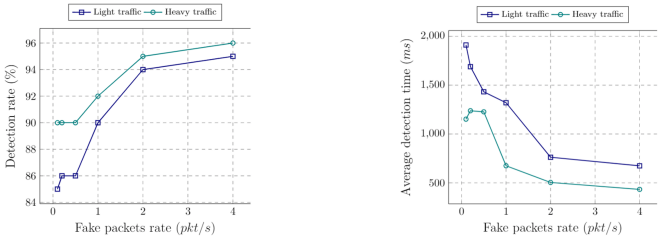
Each node generates a block if: 1) the number of received transactions reaches *n=10*; or 2) when some transactions are received (fewer than *n*) and *p=4* seconds have passed counting from the last transaction's timestamp. After a block is

generated, every node that did not go offline calculates a block hash of the transactions (ordered based on their timestamps). Next, the nodes derive from this hash the *nodeID* that would be the miner, following a proof-of-selection [23] fashion – in order to save time and resources while electing a miner – and broadcasts this hash to all nodes for comparison. After that, the nodes delete the block and retain the hash to include it in next block. In case the nodes do not receive the hash after *m=5* seconds, they elect another miner. If no mismatch is reported, the nodes consider the block's content legitimate, and can proceed to the verification process. It should be noted that the values *n*, *m*, and *p* are chosen experimentally in order to maximize efficiency, meaning that shorter values correlate proportionally with shorter detection times, and higher values with less overhead. The nodes also maintain a *CompleteView* structure that aggregates the switches' interconnections and host location information into a graph to rapidly detect inconsistencies, and updates this graph if no attack is detected. The verification procedure is operated at every block generation by all nodes, and works as follows:

- If an OFPacketInNoMatch transaction is found in the new block: TopoTrust looks for other OFPacketInNoMatch packets' timestamps that have the same path attributes, and compares them with the timestamp of the last known position; if it is considerably shorter than a normal relocation, it notifies the admin. If no attack is detected, it updates *CompleteView*. The normal relocation is defined as an average of previous relocations calculated when no attack is detected.
- If an OFPacketOutFlowMod transaction is found in the new block: the verification process searches for a corresponding OFPacketInNoMatch request within a reasonable timestamp difference. If found, it updates its *CompleteView* structure; otherwise, it notifies the admin for a possible attack.
- If an OFPacketInLLDP3 transaction is found in the new block: it searches for a corresponding OFPacketOutLLDP1 request within a reasonable timestamp difference. If not found, it notifies the admin for a possible attack; else, it updates *CompleteView*. It also compares timestamps between LLDP1 and LLDP3 packets with a latency baseline defined as average latency from previous legitimate LLDP exchanges. If the average latency is substantially longer, it notifies the admin for a possible Link Fabrication attack.

B. Performance Evaluation

In the remainder of this section, we evaluate TopoTrust by running multiple attack scenarios and repeating those scenarios until consistent results are found. The performance of TopoTrust is assessed in terms of (i) *detection rate* which refers to the percentage of fake control messages successfully detected by TopoTrust; and (ii) *average detection time* needed to detect the fake control messages. We also evaluate the computational overhead introduced by TopoTrust. Other work (presented in Section II-B) do not offer comprehensive zero-trust solutions; consequently, the attacks never originate from "trusted" nodes in their experimentation. For this reason,



(a) Detection rate (b) Average detection time

Fig. 5: Fake OFPacketInNoMatch detection results.

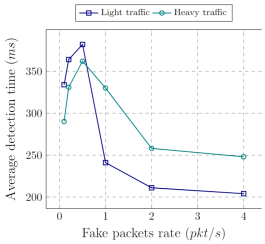
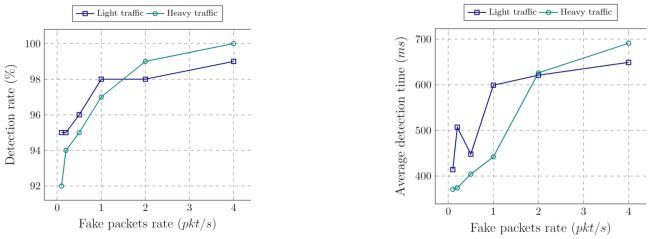


Fig. 6: Fake OFPacketOutFlowMod detection results.

comparing their detection rate directly with ours is inaccurate. Our attack scenarios are the following:

1) An attacker takes control of a switch and sends fake *packet-in* messages. Fig. 5 plots two graphs, showing (a) the detection rate and (b) the average detection time for different fake packet rates, respectively. In the experiments, we consider two different scenarios: light traffic and heavy traffic.
   Discussion: The results show that the higher the fake packet rate, the higher the detection rate and the shorter the detection time, particularly under heavy traffic conditions. That is because the difference between timestamps is smaller – as more *packet-in* messages are generated by switches – which provides a higher probability of attack detection; and more control messages means more blocks are generated, which shortens the detection time. With fake packets rate ranging between 0 to 1 pkts/s, not enough control messages are generated. This leads to a slower block creation, so the waiting time *p=4* still constitutes the deciding factor, meaning that most blocks are generated with less than *n=10* transactions.
2) An attacker controls the controller and sends fake *FlowMod*.
   Discussion: All fake *FlowMod* are detected since corresponding *packet-in* do not exist. Moreover, Fig. 6 shows that the average detection time varies in correlation with the block generation time and verification process runtime. We can see the same effect on block generation of waiting time *p* as in the previous simulation. Processing time also plays an important role; that is because more fake *FlowMod* broadcast means more searching for corresponding *packet-in*, which is more common in case of heavy traffic.
3) An attacker takes control over multiple switches, and sends fake *packet-in* simulating fake switch neighboring.
   Discussion: The results in Fig. 7 show that the probability of correct detection of the attack increases with the increase



(a) Detection rate

(b) Average detection time

Fig. 7: Fake OFPacketInLLDP3 detection results.

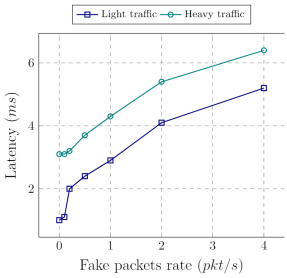


Fig. 8: Computational overhead at verification phase.

of traffic. This is justified by the fact that the normal link latency learned becomes increasingly accurate. Contrary to previous simulations, TopoTrust also consumes more processing time since it searches for corresponding OFPacketOutLLDP1 on the Blockchain, and compares their timestamps with the baseline stored in *CompleteView*, as a result, there is an increase in the detection time.

4) The graph in Fig. 8 represents the latency introduced by TopoTrust, in both heavy and light traffic scenarios while varying the rate of fake packets injected in the network:
Discussion: Both computational and network overhead introduced in the transaction broadcast phase are negligible in our tests because only a small amount of data is being signed and sent to other nodes (see Section V-A). On the other hand, we also measured the computational overhead at block/topology verification time, and the results in Fig. 8 show that a small overhead is introduced, which varies proportionally with fake packet rate. In fact, the more the traffic and attacks are generated, the more block creation and topology verification are carried out.

VI. Conclusion

This paper introduced a completely trustless, Blockchain-based solution, dubbed TopoTrust, to detect Topology poisoning attacks in a SDN. The results of the conducted simulations demonstrated that TopoTrust could successfully detect various attacks with promising detection rates and detection times. This work also exposed the security implications when restricting the role of the data plane to only executing static flow rules, and how various attacks can be launched consequently. That is why researchers have recently began experimenting programmable data planes such as P4 [24] in order to introduce *smartness* into the forwarding devices. As future research work, we will investigate the potential of stateful data planes regarding SDN security challenges.

Acknowledgment

This work was supported in part by the Academy of Finland Project 6Genesis Flagship (Grant No. 346208) and the European Union's Horizon 2020 research and innovation programme under the MonB5G project (Grant No. 871780).

References

[1] H. Yu, T. Taleb, J. Zhang, and H. Wang, "Deterministic Latency Bounded Network Slice Deployment in IP-over-WDM based Metro-Aggregation Networks," *IEEE TNSE*, vol. 9, no. 2, pp. 596 – 607, Apr. 2022.

[2] C. Benzaid and T. Taleb, "AI-driven Zero Touch Network and Service Management in 5G and Beyond: Challenges and Research Directions," *IEEE Network Magazine*, vol. 34, no. 2, pp. 186 – 194, Mar./Apr. 2020.

[3] ——, "ZSM Security: Threat Surface and Best Practices," *IEEE Network Magazine*, vol. 34, no. 3, pp. 124 – 133, May/June 2020.

[4] "Deterministic Networking (detnet)," https://datatracker.ietf.org/wg/detnet/about/, accessed: 2022-04-06.

[5] S. Khan, A. Gani, A. W. Abdul 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, 2017.

[6] Q. Li, X. Zou, Q. Huang, J. Zheng, and P. P. C. Lee, "Dynamic packet forwarding verification in sdn," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 6, pp. 915–929, 2019.

[7] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks," 01 2015.

[8] K. Agarwal, E. Rozner, C. Dixon, and J. Carter, "SDN traceroute: Tracing SDN Forwarding without Changing Network Behavior," *in Proc. of HotSDN'14*, pp. 145 – 150, Aug. 2014.

[9] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig, "SDNsec: Forwarding Accountability for the SDN Data Plane," in *2016 25th ICCCN*, 2016, pp. 1–10.

[10] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures," in *NDSS*, 2015.

[11] P. K. Sharma, S. Singh, Y.-S. Jeong, and J. H. Park, "DistBlockNet: A Distributed Blockchains-Based Secure SDN Architecture for IoT Networks," *IEEE Commun. Magazine*, vol. 55, no. 9, pp. 78–85, 2017.

[12] Y. Liu, B. Zhao, X. Li, S. Wang, B. Zhang, and Z. Liu, "A Trust Chain Assessment Method Based on Blockchain for SDN Network Nodes," in *2019 IEEE SmartIoT*, 2019, pp. 240–245.

[13] "OpenFlow Switch Specification (v1.5.1)," https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, accessed: 2022-04-06.

[14] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[15] C. Benzaïd, T. Taleb, and M. Z. Farooqi, "Trust in 5G and Beyond Networks," *IEEE Network*, vol. 35, no. 3, pp. 212–222, 2021.

[16] O. Hireche, C. Benzaid, and T. Taleb, "Deep Data Plane Programming and AI for Zero Trust Self-Driven Networking in Beyond 5G," *Elsevier J. on Computer Networks*, vol. 203, Feb. 2022.

[17] A. Derhab, M. Guerroumi, L. Maglaras, M. A. Ferrag, M. Mukherjee, and F. Khan, "BLOSTER: Blockchain-based System for Detection of Fraudulent Rules in Software-Defined Networks," 09 2019.

[18] B. Agborubere and E. Sanchez-Velazquez, "OpenFlow Communications and TLS Security in Software-Defined Networks," in *2017 IEEE iThings-GreenCom-CPSCom-SmartData*, 2017, pp. 560–566.

[19] "Zero Trust Architecture," https://csrc.nist.gov/publications/detail/sp/800-207/final, accessed: 2022-04-06.

[20] C. Benzaid, K. Lounis, A. Al-Nemrat, N. Badache, and M. Alazab, "Fast authentication in wireless sensor networks," *Future Generation Computer Systems*, vol. 55, pp. 362 – 375, 2016.

[21] "Floodlight Controller," https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller, accessed: 2022-04-06.

[22] "Cisco Servers - Unified Computing System (UCS)," https://www.cisco.com/c/en/us/products/servers-unified-computing/index.html, accessed: 2022-04-06.

[23] "Proof of Stake Instead of Proof of Work," https://bitcointalk.org/index.php?topic=27787.0, accessed: 2022-04-06.

[24] "P4," https://opennetworking.org/p4/, accessed: 2022-04-06.