# Implementation of a traffic flow path verification system in a data network

Javier Velazquez Martinez*, Mattin Antartiko Elorza Forcada*, Antonio Pastor*, Diego Lopez*
and Jesús A. Alonso-López†
*Telefonica CTIO, Spain
Email: {j.velamar, mattinantartiko.elorzaforcada, antonio.pastorperales, diego.r.lopez}@telefonica.com
†Universidad Complutense de Madrid, Spain
Email: jesusaal@ucm.es

***Abstract*—This paper focuses on one of the recent concerns that has arisen regarding the network softwarization, specifically, traffic attestation in service chaining. The central focus of the paper is the design, development, and evaluation of an implementation of Ordered Proof of Transit (OPoT) as a solution to validate flow paths in the network. This solution uses Shamir's Secret Sharing (SSS) system to add metadata to each packet, updating them at each node or service it traverses until reaching the final destination. This method ensures the validation of services traversed by the packet at the last crossing point, providing an additional layer of security and preventing unauthorized modifications to the flow of data traffic. We report here how a programmable data plane, based on the P4 language, can be used to provide OPoT features dynamically, according to user and network policy requirements. Additionally, a controller will be developed to configure the network nodes, execute OPoT, and monitor the system state.**

***Keywords*—Software Defined Networks, Network Function Virtualization, Service Function Chaining, Ordered Proof of Transit, P4**

## I. Introduction

In recent years, the transformation of the network towards SDN [1] and NFV has led to the development of new innovative technologies but also new concerns regarding security.

On the one hand, P4 [2] has emerged as a significant component in this transformation. P4 allows the programming of network devices independently of the protocol. With P4, engineers can define how network packets should be processed and how routing decisions should be made. This has enabled greater customization and adaptability in the behavior of network devices. Furthermore, P4 also enhances the performance of software-defined networks by enabling more efficient and specific programming.

On the other hand, one of the main concerns in the network evolution, where different services are deployed, is the path taken by packets. Specifically, it is key to validate that the flow travels through the services defined in a specific order, as virtualized environments reduce visibility over the traffic route.

Proof of Transit (PoT) proposes a solution to validate the paths a flow takes in the network by sharing a small segment of metadata added to each packet. This metadata is updated at each node or service crossed along the path until reaching the final destination. A central controller divides a secret into as many parts as there are nodes participating in the system and sends it through a secure channel, following Shamir's Secret Sharing Scheme (SSS) [3]. The set of secrets is managed by the controller and applied by the verifying node, which is the last one on the route. When the last node receives the PoT data, it compares the received value with its secret to validate whether the packet has followed the correct route. To ensure the established order, *Ordered Proof of Transit* (OPoT) is proposed in[4], which extends the PoT scheme by adding symmetric masking between nodes.

In [4], apart from the OPoT conceptual definition some security aspects are analyzed. RFC 9197 defines an IOAM field to include PoT parameters to support path verification, but not the ordered version. [5] proposes to employ quantum technologies for key generation to support symmetric masking between nodes. While [6] defines an alternative approach to perform Proof of Transit in P4 by using PolKA [7], it does not explicitly define a method for performing operations in a specific order (OPoT) and uses Mersenne numbers to address the lack of modulo operation in the P4 language.

In this paper, the authors propose a P4-based solution that can increase the efficiency and versatility implementing OPoT required functionalities.

The rest of the paper is organized as follows: Section II discusses a pair of examples use cases where OPoT can be used. The prototype implementation details are elaborated in Section III. Section IV is dedicated to the validation of our work, including testing scenarios and presents the results. Finally, Section V concludes the paper and provides insights into potential future work.

## II. Use Cases

This section describes two possible use cases where the developed framework of Ordered Proof of Transit can be applied.

### A. OPoT-Based VPN

This use case aims to address the problem currently affecting most Virtual Private Network (VPN) services offered by MPLS providers, both at Layer 2 in a VPLS and at Layer 3. When a client connects to a VPN, they must fully trust the VPN MPLS provider, as the traffic is routed through the VPN to its destination, and the client has no means to verify

if the traffic has been diverted, or altered the order, during this process.

There might be instances where the provider incorrectly provisions MPLS labels, potentially exposing sites of different clients to other Provider Edge (PE) routers. Also, an error in VPN route configuration could direct traffic through the same PE but to different Customer Edge (CE) routers, also exposing the traffic to other undesired clients. Moreover, to ensure security or maintain compliance requirements, certain organizations utilizing MPLS VPNs may necessitate a strict ordering in packet delivery between their dispersed branches.

By implementing OPoT, clients can trust that their traffic has passed through a specific set of nodes to reach its intended destination and has not exited the VPN without their knowledge. This requires activating PoT for any PE in the operator's network.

This solution can be extended to Software-Defined Wide Area Network (SD-WAN) and Secure Access Service Edge (SASE) technologies, leveraging either MPLS or an overlay connectivity. Integrating OPoT verification into customer premises equipment (CPEs) and providing associated metrics in the SD-WAN client console will enhance the trustworthiness of these solutions.

This use case holds many possibilities. For instance, if the VPN has a security and monitoring system in place, PoT can be used to verify that client traffic goes through this system, enhancing the security of users within the VPN.

### B. *Multi-domain infrastructure verification and OPoT*

Future vertical services supported by 6G will depend on solutions that implement a certain degree of automation in the network management plane to ensure smooth network slice connectivity over different domains interconnected such as public and non-public Radio Access Network (RAN), transport, Edge, or cloud. Part of the management decision process will involve an evaluation of the security and trust levels of these network domains and the end-to-end connectivity for these vertical services. Visibility and verification of how network paths are established will become an essential feature for critical services.

This use case proposes the use of OPoT to generate metrics that assist in this management decision-making process in 6G. An example could be a smart city application deployed across several neighboring metropolitan areas, utilizing a 6G multi-domain network slice neutral-host infrastructure. The deployment involves sensitive data, such as surveillance device data. The public administration deploying the application requires guarantees that the applications and data running over multiple domains such as edge are those specified by contract, for example domain providers with higher security and trust levels for sensitive components and incorporating verifiable privacy principles.

## III. Implementation

### A. *Architecture*

In this section, the architecture of the entire system is explained. As shown in Figure 1, the system comprises four elements: the P4 nodes, responsible for all the OPoT-related logic (adding/removing headers, calculating parameters, and sending metrics); the controller, in charge of configuring the nodes with a P4 program containing the necessary OPoT logic, as well as sending parameters for calculations performed in the nodes; two hosts that generate and receive traffic, which is verified by OPoT; and a collector whose task is to process the metrics received by the nodes and send them to a database for subsequent analysis. These elements are detailed in section III-B.

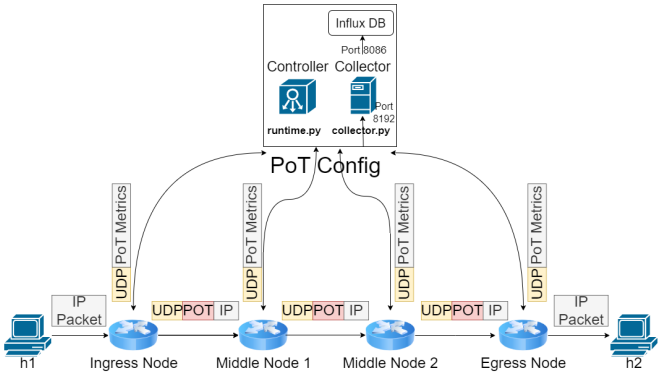



Fig. 1: OPoT system architecture

The general workflow of the system is summarized as follows:

1) The controller generates OPoT configuration and parameters to set up the nodes through gRPC messages while initiating the OPoT service.
2) The controller starts the collector to receive metrics from the nodes.
3) Once nodes are configured, the hosts begin sending traffic. This traffic passes through the established path of nodes, receiving specific treatment depending on the packet position in the path.
4) Packets arrive at the ingress node defined by the controller. The ingress node adds three headers to the packet containing the calculated OPoT parameters. Before sending it, the packet is encrypted with a mask provided by the controller.
5) Then, middle nodes process the packets: extracting the OPoT content by decrypting it with a mask, calculating new parameters, and sending them again encrypted to the next node.
6) Lastly, the egress node extracts the content, performs OPoT calculations, and verifies their correctness.
7) If incorrect calculations are detected, the packet is discarded. If correct, OPoT headers are removed, and the packet is sent to the final host.
8) During this process, nodes send metrics every five processed packets to a collector. These metrics include data like packet sequence number, RND and CML values, or timestamps.
9) The collector processes the received packets and sends them to a database.

From a data plane perspective, each switch operates following the *v1model* architecture [8]. This means that every packet

reaching a switch configured with the P4 program is parsed to obtain header data, verifies the checksum value present in its headers, and enters a pipeline containing multiple match-action tables computed based on the obtained values. After making the necessary modifications to the processed packets, a new checksum value calculation is performed on the headers requiring it, and the packet is deparsed to send it to the next node. Figure 2 summarizes this process.

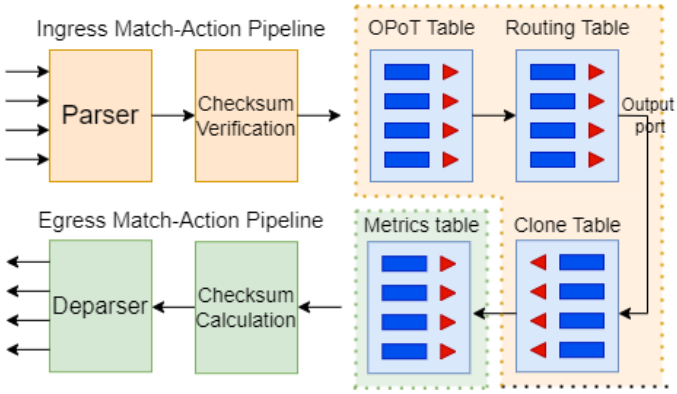



Fig. 2: Flow that packet follows when reaching a P4 node

## B. Components

*1) Controller :* The controller serves as the central entity that manages and orchestrates network traffic, responsible for the overall functioning of the OPoT system. It handles the configuration of P4 nodes by generating specific OPoT parameters based on the network topology. These are necessary for the OPoT nodes to perform the calculation described in [4]:

$$CML = ((f_1(x_i)+f_2(x_i)+RND)*LPC_i)+CML_{i-1}(modp)$$

The list of parameters generated by the controller includes:

- **Prime number *p*** needed for performing all operations within the arithmetic field defined by the prime number.
- **OPoT identifier**: a unique identifier for a specific node route where OPoT is configured. This parameter will be sent to the ingress node.
- **Service Index**: defined by a number, it establishes the position of an OPoT node in the route.
- **Secret Polynomial** $f_1(x)$ of degree $n$-1, where $n$ is the number of nodes in the scenario. $n$ points are generated from this polynomial and sent to each respective node. Additionally, the constant term of this polynomial is sent to the egress node. This term serves as a verification key to confirm that the traffic has passed through the designated OPoT nodes in the configuration.
- **Lagrange Polynomial Constants (LPC)** derived from the $n$ calculated points. They are sent to each node in the same manner as the points of the secret polynomial.
- **Public Polynomial** $f_2(x)$ of degree $n$-1, where $n$ is the number of nodes in the scenario. $n$ points are generated from this polynomial and sent to each respective node.
- **XOR masks** for implementing OPoT.
- **Magic numbers**: Since the P4 language has not implemented modulo or division operations, necessary for OPoT calculations, magic numbers [9] are used to perform these operations. Therefore, the controller sends magic numbers to all nodes to perform the operations. The calculation of these magic numbers varies depending on the generated prime number.

The controller, once the parameters are generated, compiles the P4 program, establishes a secure gRPC channel using TLS for parameter transmission, and configures the nodes through the P4Runtime API to act as PoT nodes.

*2) P4 Nodes :* The P4 nodes are switches are in charge of forwarding packets. These nodes are connected to a controller via gRPC using the *P4Runtime* Control API. In this way, the P4 nodes only need to run a *P4Runtime* server that listens to the controller gRPC communications.

At this stage, there are three different types of P4 nodes, depending on the rules installed in the switches:

**Ingress node**

It is the node that each packet encounters upon entering the OPoT-enabled path. There are only two in the entire system, one for each boundary. Depending on the traffic direction, a P4 node acts as either an ingress or egress node. The ingress node is responsible for adding headers to the packet so that all other nodes recognize it as an OPoT packet and apply the necessary calculations. Specifically, this node performs an encapsulation of the original IP packet. For this implementation, Network Service Header (NSH) [10] has been selected. Here, a size of 16 bytes is specified for the Context Header, which carries the OPoT parameters:

- **RND** (32 bits): it is a random value acting as the independent term of the public polynomial. A different value is calculated each time a packet enters the OPoT. It is used by each node to compute the accumulated value and compare it with the secret key in the final node.
- **CML** (64 bits): it is the accumulated value resulting from OPoT operations. Each node receives this value from the previous node, except the ingress node, which performs calculations without a previous accumulated value.
- **Sequence number** (32 bits): this number identifies each packet in the OPoT. It increments each time a packet passes through the ingress node.

Figure 3 demonstrates how these headers are added and how the packet is encapsulated.

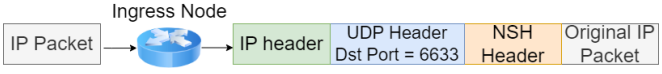



Fig. 3: Packet encapsulation after ingress node processing

Regarding the OPoT, the ingress node needs to perform several operations to construct the NSH header:

- Calculate the RND value randomly between one and the prime number received from the controller.
- Compute the CML value using the parameters received from the controller.
- Increment the sequence number by one to assign it to the current packet. To achieve this, the P4 program uses a register that initializes when the switch starts and updates each time a packet enters the OPoT.

Once the Context Header is constructed with the three parameters, it is encrypted using an upstream mask to perform the OPoT. This same mask is only available to the adjacent node. Consequently, only the next node in the OPoT sequence can correctly decipher the packet. Finally, the output port is set to send the packet to the next P4 node.

**Middle Node**

This node is responsible for forwarding packets to the subsequent node. It neither adds nor removes any headers. There can be as many middle nodes as required. Upon receiving a packet, the values within the Context Header are deciphered using the downstream mask provided by the controller. Using these values, the node performs two distinct operations

- Compute the new CML with the values provided by the controller and the RND and CML values from the preceding node.
- Update the Service Index value.

Once these operations are executed, the metadata from the Context Header is encrypted using the upstream mask, and the packet is forwarded to the next node.

**Egress node**

The egress node performs the opposite of the ingress node. Similar to the ingress node, there are only two egress nodes in the system, one on each border. Firstly, the node extracts the values from the context header and decrypts them using the downstream mask. Subsequently, it executes two operations:

- Similar to the middle node, it computes the new CML using the values provided by the controller and the RND and CML values from the previous node.
- It compares the calculated CML with the sum of the RND value and the validation key (independent term of the secret polynomial). If the values match, it means that the packet has traversed the entire OPoT path, and the packet is accepted. If they do not match, the packet is instantly discarded.

If the packet is accepted, the egress node undoes the encapsulation by removing the headers added previously at the ingress node (Figure 4).

*3) Collector :* The collector is the component responsible for gathering information about the network state. The OPoT node sends periodic metrics about processed packets, which are collected by this component. For this purpose, a collector is deployed within the same subnet as the controller.

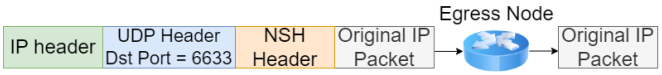



Fig. 4: Packet decapsulation after egress node treatment

Firstly, the OPoT nodes generate and send packets with metrics through the interface connected to the collector. Metrics are generated using the P4 *clone* function, which duplicates processed OPoT packets to modify and process them as metric packets. This process occurs when the egress node discards a packet or periodically if the packets are accepted, to prevent network congestion.

Secondly, the collector extracts information from these packets for processing, and finally, the resulting metrics are stored in an Influx database [11] connected to the collector.

## IV. Validation

This section details the different OPoT implementation scenarios, along with the tests conducted to validate its correct operation and measure its performance.

### A. Scenarios

Various scenarios have been defined based on the nature of the tests performed. For basic functional tests, a basic linear topology scenario with four nodes has been used, as shown in section III-A. Additionally, linear scenarios with a greater number of nodes have been defined for performance tests. Finally, tests have been conducted in a malicious scenario that includes an additional malicious node that does not perform the OPoT. This node acts as an external agent that captures traffic between two OPoT nodes. Specifically, the external agent impersonates an intermediate node by bypassing the traffic, as shown in figure 5.

This scenario serves as a test to demonstrate how the system behaves when the traffic does not follow the expected path, caused by a malicious agent or a configuration error.

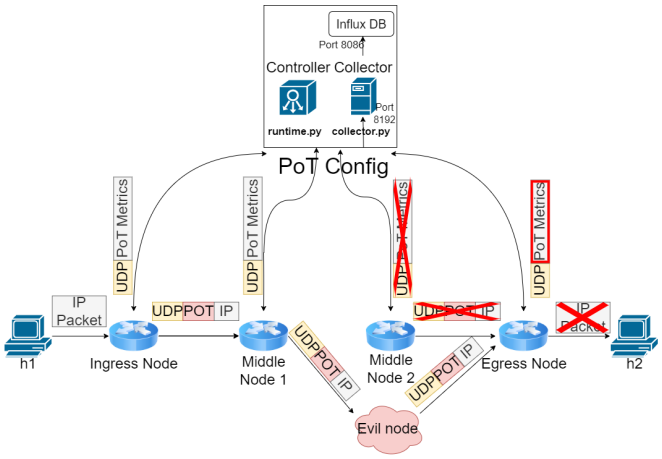



Fig. 5: Malicious scenario

### B. Results

Initially, a scenario is created with four nodes configured to perform OPoT. Subsequently, to verify the system proper

functioning, several packets are transmitted from host 1 to host 2. Accessing the Influx database allows retrieval of metrics related to the transmission, as described in section III-B3.

After the initial test of the four-node scenario, a similar test is conducted in the malicious scenario. However, in this case, packets do not reach their destination (Table I).

| time | CML | Dropped | RND | Sequence Number | Service Index | Switch IP |
|---|---|---|---|---|---|---|
| 0 | 11 | 0 | 58 | 0 | 3 | 10.0.0.11 |
| 3 | 57 | 0 | 58 | 0 | 2 | 10.0.0.12 |
| 6 | 13 | 1 | 58 | 0 | 0 | 10.0.0.14 |
| 1004 | 7 | 1 | 27 | 1 | 0 | 10.0.0.14 |
| 2004 | 37 | 1 | 48 | 2 | 0 | 10.0.0.14 |
| 3003 | 21 | 1 | 10 | 3 | 0 | 10.0.0.14 |
| 4004 | 51 | 1 | 31 | 4 | 0 | 10.0.0.14 |

TABLE I: Malicious scenario stored metrics

Each packet received by the output node is marked as "dropped". The cause of this behaviour lies in the presence of a malicious node acting as a middle node, with IP 10.0.0.13, but not correctly performing the OPoT calculations. Therefore, when the packets reach the egress node, it discards them, and host 2 never receives them. The other nodes in the network continue generating metrics every five packets, except for the middle node, which never receives any packets.

**Performance tests**

Several tests have been conducted to measure the system performance. Initially, the performance of the scenario is compared to an equivalent scenario but with P4 nodes configured for basic packet forwarding without OPoT functionality. Then, the frequency of metric generation and transmission to the collector is changed. Finally, the performance is measured by modifying the number of nodes operating with OPoT.

*1) Performance compared to the scenario without OPoT:* It is crucial to determine the impact on performance caused by OPoT compared to normal traffic behavior. To assess this, latency and throughput are measured in a scenario with four nodes, where each node is configured with a P4 program to operate based on basic forwarding behavior (Figure 6).

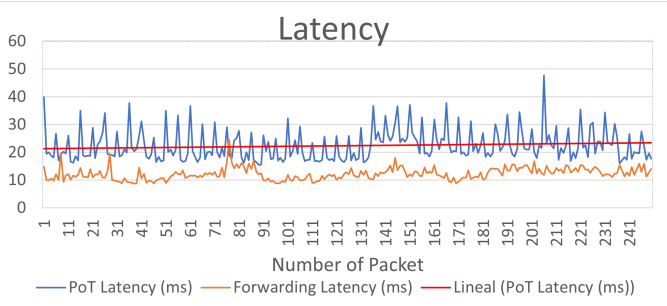



Fig. 6: Latency comparison

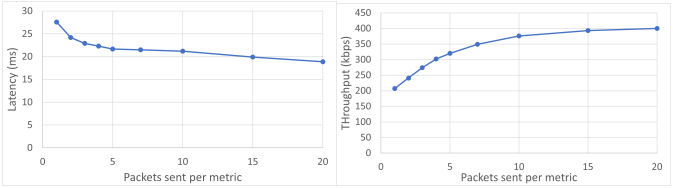
Latency introduced in the forwarding scenario is 11.2 milliseconds, whereas in the OPoT scenario, it reaches 21.7 ms. This represents a 93% latency increase, nearly doubling the initial value. Moreover, the latency variance is higher in the OPoT scenario than in the forwarding scenario. This is due to metric generation, occurring every five packets, requiring more processing time than the usual OPoT processing. This metric generation performance behavior will be further analyzed when changing the frequency of this event. Regarding throughput, the use of OPoT results in a 58% reduction in throughput. In the case of UDP, the impact is more noticeable, with throughput reduced by 82%.

| *Throughput* (kbps) | Forwarding | OPoT | Decrease |
|---|---|---|---|
| TCP | 585 | 243 | 58% |
| UDP | 2970 | 522,5 | 82% |

TABLE II: Throughput stadistics

The observed performance loss indicates that this technology cannot be universally applied to every packet but rather by sampling or selecting those flows or applications where the use of OPoT is critical. This loss is considered to be due to the mathematical calculations performed in the OPoT table depicted in figure 2. Additionally, these tests have been conducted in virtual environments without any hardware acceleration. It is expected that implementing this on Tofino [12] switches will improve performance.

*2) Performance based on metric generation frequency:* For this test, the frequency at which nodes generate and send metrics to the controller is altered to verify its impact on system performance. It is important to note that the output node is not discarding any packets in this test. If this node were to discard any packets, the measurements would be altered, as when the output node discards a packet, additional metrics are generated.



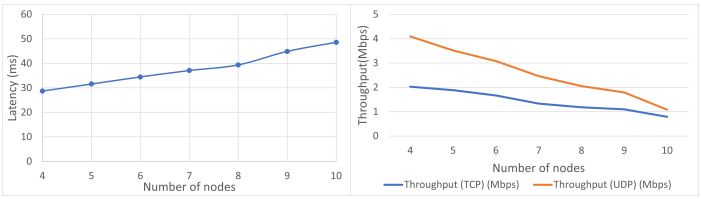

(a) Latency (b) Throughput TCP

Fig. 7: Performance based on metric generation frequency

Both graphs (Figure 7) exhibit a logarithmic trend, where the system performance significantly diminishes at higher frequencies but stabilizes at lower frequencies. Based on these results, the delay introduced by the metric generation process can be estimated. Consequently, it is recommended to use values between five and ten metrics, where there is a better balance between performance and metric collection.

*3) Performance based on the number of nodes in the scenario:* Lastly, these measurements are conducted to evaluate the impact of the number of nodes in the OPoT process. Figure 8 displays the results ranging from four to ten nodes.

It is noticeable that all graphs follow a linear trend, emphasizing UDP throughput values, where the decrease in performance is more pronounced than with TCP. The necessary analysis of the number of required nodes is facilitated by this linear behavior.




(a) Latency (b) Throughput

Fig. 8: Performance based on number of scenario nodes

## V. Conclusions

In this paper, a system for dynamically verifying the routes taken by traffic in the network has been developed. This system has been validated by constructing various scenarios in which different measurements have been conducted. OPoT is a very interesting solution for solving problems of this nature, and with the P4 language, we have been able to implement, with varying levels of difficulty, all the operations required by OPoT, confirming the viability of the solution.

Although the development primary focus was always on the data plane, the project objectives expanded to program the control plane and use a standard for sending OPoT parameters through a secure channel. Additionally, the inclusion of the metrics system posed a challenge in development, but once resolved, it enhances the solution functionality through system state monitoring.

Finally, various validation tests have been conducted to verify the system proper functioning and measure its performance. The results allow us to conclude that implementing this solution in an environment where might be needed, is viable. Nevertheless, it is important to consider that using OPoT has a significant impact on network traffic performance and comes with some limitations. Therefore, this prototype is useful for networks where flows do not require high bandwidth and where there is not a large number of hops.

### A. Future Lines

Regarding future research directions, the top two priorities are to migrate this solution to the Tofino architecture [12] implemented by Intel's intelligent switches and to integrate it to an existing open-source controller such as ETSI TeraFlow SDN [13]. These developments represent an improvement in the performance achieved in packet processing within OPoT, enhancing the system's capabilities and outcomes.

Moreover, OPoT technology is very well positioned as a source of trustworthiness of networks infrastructures. The authors plan to study OPoT measurements integration with trustworthiness and reputation engines to generate trust evaluation for the providers of networks segments and nodes.

Furthermore, another improvement to be implemented in the future is to establish a secure channel for metrics transmission. Several solutions have been proposed to address this, such as using masks to encrypt information, which is resource-intensive as it requires continuous modification to prevent attackers from deciphering the content. There is also consideration for using transport-level security protocols like DTLS [14] or network-level security like IPsec [15].

Lastly, as the collected metric information is stored in a database, the goal is to integrate this information into data models that allow for a more comprehensive and detailed view of the system, facilitating monitoring, analysis, optimization, and informed decision-making. This leads to improved performance, efficiency and scalability of the system.

## Acknowledgment

The research is supported by the PRIVATEER EU project, Grant agreement N 101096110 and by the "Ministerio de Asuntos Económicos y Transformación Digital" and the European Union-NextGenerationEU in the frameworks of the "Plan de Recuperación, Transformación y Resiliencia" and of the "Mecanismo de Recuperación y Resiliencia" under reference 6GMICROSDN_SMARTNICS (TSI-063000-2021-19).

## References

[1] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, "Software-Defined Network (SDN) Data Plane Security: Issues, Solutions, and Future Directions," in *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*, B. B. Gupta, G. M. Perez, D. P. Agrawal, and D. Gupta, Eds. Cham: Springer International Publishing, 2020, pp. 341–387. [Online]. Available: https://doi.org/10.1007/978-3-030-22277-2_14

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[3] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[4] F. Brockners, S. Bhandari, T. Mizrahi, S. Dara, and S. Youell, "Proof of Transit," Internet Engineering Task Force, Internet-Draft draft-ietf-sfc-proof-of-transit-08, Nov. 2020. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-sfc-proof-of-transit/08/

[5] A. Aguado, D. R. Lopez, V. Lopez, F. de la Iglesia, A. Pastor, M. Peev, W. Amaya, F. Martin, C. Abellan, and V. Martin, "Quantum technologies in support for 5g services: Ordered proof-of-transit," in *45th European Conference on Optical Communication (ECOC 2019)*, 2019, pp. 1–3.

[6] E. S. Borges, V. B. Bonella, A. J. D. Santos, G. T. Menegueti, C. K. Dominicini, and M. Martinello, "In-situ proof-of-transit for path-aware programmable networks," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 170–177.

[7] C. Dominicini, D. Mafioletti, A. C. Locateli, R. Villaca, M. Martinello, M. Ribeiro, and A. Gorodnik, "Polka: Polynomial key-based architecture for source routing in network fabrics," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 326–334.

[8] P4 Community, "V1model source code," available at https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4.

[9] H. S. Warren, *Hacker's delight*. Pearson Education, 2013.

[10] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)," RFC 8300, Jan. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8300

[11] "Influxdb," available at https://www.influxdata.com/.

[12] Intel Corporation, "Programabilidad p4 de intel® tofino™ 2 con más ancho de banda," *Available at: https://www.intel.la/content/www/xl/es/products/details/network-io/intelligent-fabric-processors/tofino-2.html*, 2023.

[13] "TeraFlowSDN," available at https://tfs.etsi.org/.

[14] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," RFC 6347, Jan. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6347

[15] S. Frankel and S. Krishnan, "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," RFC 6071, Feb. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6071