In-network Computing for Secure Distributed AI

Invited paper

Chiara Grasselli[1], Lorenzo Rinieri[1], Pol González[2], Luis Velasco[2], Davide Careglio[2], and Franco Callegati[1]

[1]*Department of Computer Science and Engineering, University of Bologna, Italy*

[2]*Department of Computer Architecture, Universitat Politècnica de Catalunya, Spain*

Corresponding author e-mail: davide.careglio@upc.edu

2024 24th International Conference on Transparent Optical Networks (ICTON) | 979-8-3503-7732-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICTON62926.2024.10647267

ABSTRACT

We argue in this work that exploiting in-network computing using a programmable data plane offers a promising approach to secure communications between distributed control agents in forthcoming 6G networks. We propose a solution that uses P4-programmable switches to perform in-network encryption and provide some experimental results that demonstrate the effectiveness of this approach.

Keywords: In-network computing, 6G, Programmable Data Plane, P4, NFV-MANO, SDN

1 INTRODUCTION

It is expected that Artificial Intelligence (AI) will play a key role in enabling autonomous management operations, such as service QoS control, in future-generation networks. Multiple software modules running AI or Machine Learning (ML) control algorithms, called agents, can be deployed in different network sections as Virtual Network Functions (VNFs) and orchestrated according to the Network Function Virtualisation (NFV) principles [1, 2]. The agents will be required to communicate and share knowledge to establish distributed collaborative control pipelines by exchanging telemetry data and other types of information over a public infrastructure.

In such a scenario, the information exchanged between agents will be critical to determining the overall network behavior and, therefore, must be protected from malicious attacks. Nonetheless, securing the agent communications may become non-trivial due to the number of connections and the distributed nature of the interactions. The original proposal of this manuscript is to exploit a programmable data plane to offload the task of securing communications. P4-programmable switches will be able to identify the agent-related traffic and cipher it properly while leaving the legacy traffic untouched.

The manuscript describes the proposed architecture and provides some experimental results that demonstrate the effectiveness of the approach. The main contributions are: **1.** The implementation of a custom P4 Parsing Pipeline and an *extern* function in P4 switches to encrypt/decrypt the payload of the traffic sent between the agents; **2.** An example of integration of this process into an orchestration framework, implemented according to the ETSI NFV-MANO standard, that will enable the control pipeline deployment and the securing of the traffic; **3.** An experimental validation of the proposed approach, showing the effectiveness of in-network encryption.

2 SYSTEM DESCRIPTION

Figure 1 depicts the proposed system architecture. A Service Management and Orchestration (SMO) platform is in charge of the deployment of the control agents according to the NFV paradigm principles. The agents of the same control pipeline can be distributed in different locations across the network depending on the network services and nodes they need to monitor. Local Virtualized Infrastructure Managers (VIMs) control the resources in each computing cluster. As specified by NFV standards, high-level descriptors define the deployment characteristics of each control pipeline, including information such as the agent placement, the resources needed to run them, and the connectivity to be created. Alongside the orchestrator, a network controller is on top of the packet network and manages the connectivity between the agents. In particular, the network controller manages P4 programmable switches placed at the edge of the computing clusters to secure the communications between the agents. An external entity such as the ML Function Orchestrator (MLFO) proposed in [3] can coordinate the whole deployment process and manage the agent configuration, gathering and sending information to the other architectural entities via their Northbound Interfaces (NBIs). When the deployment of a new control pipeline is triggered, the orchestrator translates the high-level descriptors into lower-level implementation directives and sends them to the VIMs to run the agents in containers in the underlying infrastructure. Once the distributed agents are running, the network controller installs rules in the programmable switches to match the traffic flows exchanged between the entities of each pipeline and encrypt the data.

979-8-3503-7732-3/24/$31.00 ©2024 IEEE

Authorized licensed use limited to: UNIVERSITAT POLITECNICA DE CATALUNYA. Downloaded on September 03,2024 at 07:07:53 UTC from IEEE Xplore. Restrictions apply.

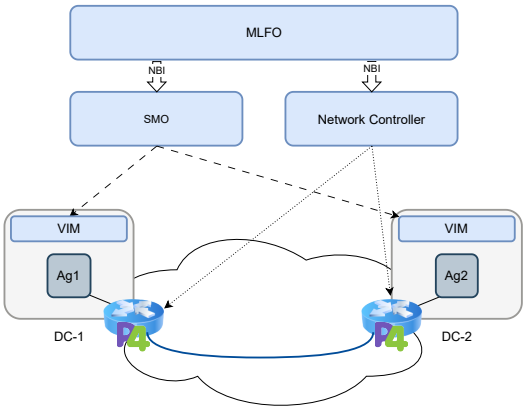

Figure 1: Schematic representation of the system architecture and its main entities. Agents deployed in different computing clusters that belong to the same control pipeline exchange data through a secure channel created with in-network encryption.

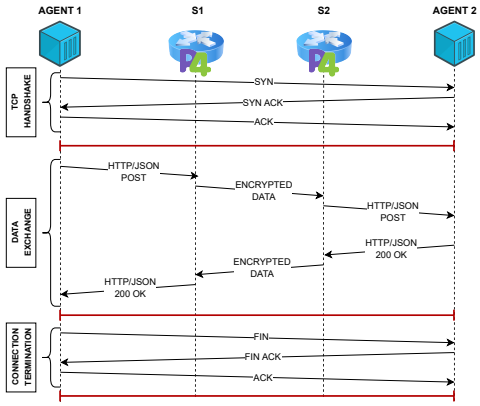

Figure 2: Example of a connection flow between two agents exchanging data via REST API in JSON format. The two P4-programmable switches encrypt and decrypt the TCP packet payload containing the HTTP and JSON data.

3 TESTBED IMPLEMENTATION

We instantiated the operational principle outlined previously within a virtualized environment utilizing the Kathará framework [4], wherein each constituent component is realized as a Docker container. In this framework, we developed specialized Docker container images to implement the functionalities of the agents. In turn, the agents are connected by P4 switches implemented by means of two other containers. They are equipped with a simple level 2 forwarding pipeline and configured to only forward traffic to the end hosts: the P4 code is compiled for the reference virtual target bmv2[1]. Furthermore, we developed a customized implementation of the control plane, responsible for making high-level decisions including switch pipeline reconfiguration [5], among other tasks. The controller, which executes the control plane functionalities, communicates with the switches through the *P4Runtime* specification[2], allowing P4 pipeline reconfiguration at runtime. P4Runtime is a well-established Application Programming Interface (API) for controlling the data plane elements of a device whose behavior is specified by a P4 program.

Taking into account that the communication between the agents occurs through HTTP REST API in JSON format, we developed a specific P4 Parsing Pipeline to handle the variable length of the HTTP message. The steps performed are:

1. The parser extracts the Ethernet and the IP headers from each packet;
2. If the *Protocol* field of the IPv4 header is equal to 6 (i.e. the TCP protocol), the header is extracted;
3. Knowing the field *Total Length* of the IPv4 header field and the field *Data Offset* of the TCP header, the P4 parser calculates the actual length of the TCP header, including the TCP options, and the total payload size;
4. Finally the HTTP message is extracted depending on the value of the TCP payload size.

This procedure facilitates subsequent actions by providing the necessary groundwork. It enables the switch to identify both the source and destination endpoints, which is paramount in determining the application of the encryption and decryption actions.

To secure the in-network channel, 128-bit key AES is the chosen algorithm. In the proposed use case, symmetric AES keys can be pre-installed in the P4 switches at configuration time or can be exchanged between the controllers via Diffie-Hellman (DH) key agreement [6] at runtime and then installed on the switches. From an operational perspective, the establishment of the secure channel between the two bmv2 switches can be accomplished in two ways. It can either be configured permanently at system start-up, routing all communication requiring encryption through this channel. Alternatively, it can be established

[1] `https://github.com/p4lang/behavioral-model`
[2] `https://p4.org/p4-spec/p4runtime/v1.3.0/P4Runtime-Spec.html`

Authorized licensed use limited to: UNIVERSITAT POLITECNICA DE CATALUNYA. Downloaded on September 03,2024 at 07:07:53 UTC from IEEE Xplore. Restrictions apply.

dynamically for specific traffic flows, with each flow having its dedicated encrypted channel. The decision regarding which strategy to employ is a matter for the control plane.

To develop the AES encryption and decryption functions, drawing from our previous work concerning Hash functions [7], we exploited the concept of P4 *extern*. An *extern* is an API that uses an external dependency, which can be queried by the target. In this case, the implemented extern leverages standard C++ implementations of the AES 128-bit encryption and decryption functions. Each switch performs both encryption and decryption, depending on the flow direction, as shown in Figure 2. Subsequently, each pipeline encrypts and decrypts the payload in the ingress queue control by calling the extern and emitting the encrypted/decrypted payload in the Deparser pipeline, as summarized in Alg. 1 and Alg. 2.

Algorithm 1: Encrypt action in the bmv2 Ingress Pipeline

Input : The HTTP message m of the input packet.
Output: The encrypted data *enc* correspondent to the HTTP message m.

1 $enc \leftarrow Encrypt(m)$: call P4 extern AES Encrypt function;
2 Update the value of the IPv4 header field *Total Length*;
3 $m.setInvalid()$: do not emit the original HTTP message in the Deparser;
4 $enc.setValid()$: emit the encrypted message in the Deparser.

Algorithm 2: Decrypt action in the bmv2 Ingress Pipeline

Input : The encrypted HTTP data *enc* of the input packet.
Output: The decrypted HTTP message m.

1 $m \leftarrow Decrypt(enc)$: call P4 extern AES Decrypt function;
2 Update the value of the IPv4 header field *Total Length*;
3 $enc.setInvalid()$: do not emit the encrypted message in the Deparser;
4 $p.setValid()$: emit the decrypted HTTP message in the Deparser.

4 RESULTS

We report in this section the results obtained with the testbed described above. The bar chart in Figure 3 shows the average round-trip time (RTT) calculated by exchanging 100000 HTTP messages between the agents. The error bars in red represent the related standard deviation. For comparison, as a communication time, we considered the time interval between sending an HTTP request and receiving the subsequent response when the in-network encryption or a traditional IPsec tunnel is used. For the IPsec case, we considered different configurations by setting up a secure channel between the agents or the switches using strongSwan[3]. In tunnel mode, the entire IP packet is encrypted whereas only its payload is ciphered in transport mode. The performance exhibited by our proposed solution, utilizing in-network encryption, is comparable in order of magnitude to that of the various IPsec configurations being evaluated. Thus, if we map our approach on a hardware P4 pipeline we should be able to outperform the state of the art in terms of latency and throughput.

Figure 4, instead, shows the average time needed to set up the secure channel and start the connection in three different scenarios. In the first two cases, the P4-programmable switches use pre-shared keys or the Diffie–Hellman (DH) protocol to negotiate a shared secret key over the insecure channel, respectively. In the third scenario, as a comparison, we used the orchestrator to establish an IPsec Host-to-Host tunnel between the agents without in-network encryption. To test the latter scenario, we used Open Source MANO[4] as an orchestration platform to deploy the agents running strongSwan in two computing clusters with OpenStack[5] as VIM and to configure the IPsec tunnel at runtime with Day 2 operations defined in the descriptor packages. We exploited the Juju python framework to build proxy charms[6] as a set of scripts to configure the IPsec tunnel on both agents with a shared secret key, start the secure connection, and retrieve the entire process timestamps. Most notably, the average secure connection establishment time with the DH exchange is far less than the time needed to set up and establish an IPsec connection with the orchestrator. In the end, it is important to highlight that if the symmetric keys are pre-installed in the P4 switches, then the encrypted channel establishment time is reduced to the connection time.

[3] `https://www.strongswan.org/`
[4] `https://osm.etsi.org/`
[5] `https://www.openstack.org/`
[6] `https://github.com/charmed-osm/charms.osm`

Authorized licensed use limited to: UNIVERSITAT POLITECNICA DE CATALUNYA. Downloaded on September 03,2024 at 07:07:53 UTC from IEEE Xplore. Restrictions apply.

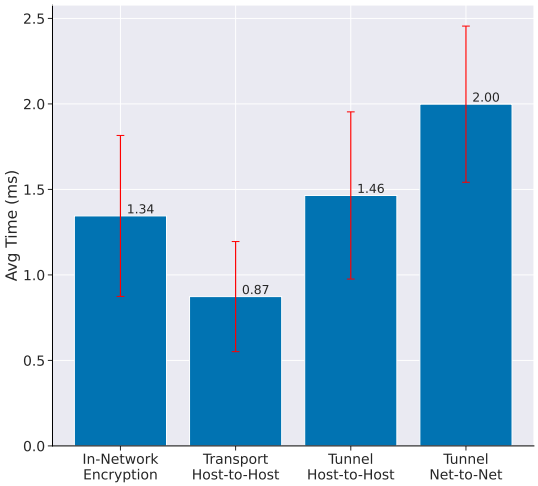

Figure 3: Average communication time between agents, comparing in-network encryption with traditional IPsec. The IPsec is configured between the agents (Host-to-Host) in transport and tunnel mode or between the switches (Net-to-Net).

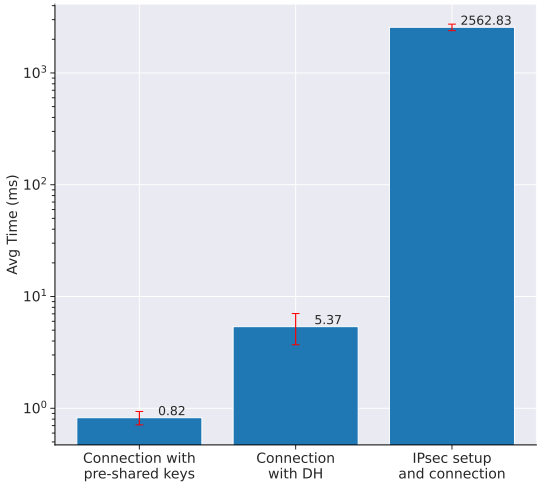

Figure 4: Secure communication average set-up time. The first two bars refer to agents communication via the in-network secure channel. The third bar refers to a traditional IPsec configuration using the orchestrator.

5 CONCLUSION

In this manuscript, we presented a solution to perform in-network encryption to secure communication between distributed control agents in future 6G network scenarios. We discussed the integration of the proposed approach exploiting P4 programmable switches with a custom Parsing Pipeline and extern function into a general orchestration framework to deploy and configure distributed control pipelines across the network. The achieved results prove how in-network encryption through P4 programmable switches is comparable in performance to state-of-the-art solutions such as IPsec while providing more flexibility in the setup phase.

ACKNOWLEDGEMENTS

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU and from the HORIZON SNS JU SEASON (G.A. 101096120).

References

[1] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6G Networks: Use Cases and Technologies," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, 2020.

[2] D. Borsatti, G. Davoli, W. Cerroni, C. Contoli, and F. Callegati, "Performance of Service Function Chaining on the OpenStack Cloud Platform," in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 432–437, 2018.

[3] P. González, A. Zahir, C. Grasselli, A. Muñiz, M. Groshev, S. Barzegar, F. Callegati, D. Careglio, M. Ruiz, and L. Velasco, "Deployment of secure machine learning pipelines for near-real-time control of 6g network services," in *2024 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–3, 2024.

[4] G. Bonofiglio, V. Iovinella, G. Lospoto, and G. Di Battista, "Kathará: A container-based framework for implementing network function virtualization and software defined networks," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, 2018.

[5] A. Al Sadi, M. Savi, A. Melis, M. Prandini, and F. Callegati, "Unleashing Dynamic Pipeline Reconfiguration of P4 Switches for Efficient Network Monitoring," *IEEE Transactions on Network and Service Management*, 2024.

[6] D. K. Gillmor, "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)." RFC 7919, Aug. 2016.

[7] G. F. Pittalà, L. Rinieri, A. Al Sadi, G. Davoli, A. Melis, M. Prandini, and W. Cerroni, "Leveraging Data Plane Programmability to enhance service orchestration at the edge: A focus on industrial security," *Computer Networks*, p. 110397, 2024.

Authorized licensed use limited to: UNIVERSITAT POLITECNICA DE CATALUNYA. Downloaded on September 03,2024 at 07:07:53 UTC from IEEE Xplore. Restrictions apply.