A Zero-Touch and NFV-Based VPNaaS Solution

Rafael Direito, Daniel Gomes, Diogo Gomes, Rui L. Aguiar

Instituto de Telecomunicações and Universidade de Aveiro, Portugal
Email: *{rdireito, dagomes, dagomes, ruilaa}@av.it.pt;*

***Abstract*—Inter-domain scenarios are immensely desirable in NFV since they allow network services to span across independent domains. However, providing a communication channel between the services' components in different domains is challenging, thus the need for mechanisms to simplify and automate this process. One possible approach to this problem is the employment of VPNaaS mechanisms. This work presents a zero-touch and NFV-based full-mesh VPNaaS solution that can be used to establish a VPN mesh network encompassing different independent domains, thus enabling inter-communication between them. Furthermore, the VPN tunnels' orchestration is fully automated, being achieved without any human intervention.**

***Index Terms*—VPNaaS, full-mesh, NFV, VPN, inter-domain, zero-touch**

I. Introduction

Network Functions Virtualization (NFV) enables abstracting Network Functions (NFs) from the hardware where they are deployed. Moreover, Virtualized Network Functions (VNFs) can easily be adjusted to the client's demands, thus offering additional flexibility, performance, and reliability [1].

The advantages of NFV are further accentuated when considering that VNFs can be coupled into Service Function Chains (SFCs), which provide complex network services. In SFCs, VNFs are placed according to a specific topology in which the traffic is steered between the VNFs based on a pre-determined rule set [2].

During the last few years, inter-domain scenarios have been highly desirable in NFV since they enable the spanning of services across different independent domains, thus increasing the service's coverage area. SFCs can also exist in inter-domain scenarios. However, enabling communication between inter-domain SFC VNFs is a complicated process, given the complex negotiations that occur between the domains right before the deployment of the SFC's VNFs.

A possible solution to simplify this is employing Virtual Private Networks (VPNs) to interconnect all domains in which an SFC is to be deployed. Some authors have already addressed this approach, resulting in the introduction of the Virtual Private Network-as-a-Service (VPNaaS) concept. This concept addresses a paradigm where VPNs are quickly and automatically orchestrated, thus simplifying their configuration and operation.

This paper presents a fully-fledged service for deploying full-mesh VPNs to interconnect independent domains. Our solution is NFV-based, does not require any human intervention, and is fully orchestrated by NetOr [3]. The VPN tunnels are created through Wireguard [1].After the deployment of the tunnels, operators may entirely rely on them to achieve inter-domain connectivity, for instance, in use cases involving distributed SFCs. This paper is organized as follows. Section II addresses some previous work around the researched topic. These works steered the aim of our work, which is presented in Section III. Section IV follows to showcases our work to advance the state-of-the-art, showcasing (i) our work proposal, (ii) the architecture of the developed system, and (iii) its implementation. Several experiments we performed and their results and discussion can be found in Section V. Finally, we present our conclusions and the suggested future work in Sections VI and VII, respectively.

II. Related Work

The first approaches to the challenge previously described relied on Software Defined Networking (SDN)-related technologies. In [4], the authors present their work on orchestrating an inter-domain L3-VPN by leveraging the OpenDaylight controller and several OpenFlow switches located in different domains. This work resulted in a prototype that enables the automatic setup and tear-down of VPNs by their end-users. To enable this, the authors of [4] developed an Agent responsible for processing the VPN creation requests from the end-users and interacting with the SDN controllers to provision the requested VPNs. The results achieved in [4] showcase that a VPN was created in no more than 10 seconds after the users requested it.

Even though the solution presented in [4] showcased positive introductory results of the applicability of SDN to automate the on-demand establishment of VPNs, it also suffers from one crucial drawback: the need for human intervention. This drawback heavily reduces the automation of this solution.

Another approach to the automated establishment of VPNs in cloud-native environments is presented in [5]. There, the authors studied how they could create additional VPN tunnels between independent domains when

[1]https://www.wireguard.com

a substantial increase in network traffic occurrs between them. The authors evaluated two VPN architectures: full-mesh VPNs versus Hub-and-Spoke VPNs, and questioned how they could leverage each architecture to achieve the deployment of new VPN tunnels on demand. They followed the Hub-and-Spoke architecture, in which all private networks communicate through a central Hub Gateway. The chosen technology to implement the VPN tunnels was IPsec.

Even though the authors were able to achieve the on demand deployment of VPN tunnels, their approach [5] raises several questions concerning the resilience of the VPNs. Since all VPN tunnels are connected to a central Hub-Gateway, this entity can be considered a central failure point, which is troublesome.

The on-demand configuration of additional VPN tunnels is also addressed in [6]. In contrast to what is presented in [5], the authors of [6] support a full-mesh architecture for the establishment of inter-domain VPN tunnels. However, their focus resides on the challenge of maintaining inter-connectivity between the different domains when some VPN mesh nodes are offline. To address this, they present a fairness-oriented topology formation algorithm that enables the addition of new nodes to the VPN without the need to rewrite the still-active overlay links. This approach highlights a new layer of complexity to the problem at hand: the need to maintain the network's throughput when some VPN nodes are offline.

Nevertheless, the presented approach also requires that the VPN tunnels are manually configured. Such limitation impacts the adoption of these solutions since they do not provide any automation mechanisms that can enable a more straightforward configuration and management of the VPN tunnels.

In [2], the authors present another approach to solve the problems related to inter-domain VNFs communication. This approach relies on deploying a VPN node VNF in all domains where the SFC spans. The VPN nodes are coordinated to establish a VPN among all of them, thus providing a secure and protected network layer for the communication of the VNFs placed in the different domains. To achieve this, the VPN nodes are an integral part of the SFC's composition, being always deployed alongside all other VNFs. To configure the VPN nodes to provide inter-domain connectivity, the authors employ day-2 operations, which are executed via a REST API, offered by the these nodes. However, once again, this approach involves always deploying the VPN node VNFs alongside all the SFC's VNFs located in an independent domain, which is highly impractical.

Finally, in [7], the authors present a similar approach to the one described in [2]. However, they only aim to automate the establishment of VPN tunnels between 2 independent domains. To create the VPN tunnel, the authors suggest the creation of Wireguard Server VNFs. The NFV Orchestrators (NFVOs) of each domain orchestrate these VNFs. To configure the Wireguard Servers, day-0, day-1, and day-2 operations were employed. A highly addressed topic in [7] is the workflow for the key exchange between the Wireguard Servers to establish the VPN tunnels. This workflow relies on day-2 operations invoked at the Open Source MANO (OSM)'s level. To execute these actions, however, the domains' administrators must first establish a Secure Shell (SSH) connection with the Wireguard VNFs, and manually retrieve their keys. Then, the administrators may invoke the operations that lead to creating a VPN tunnel. Thus, the process of establishing this tunnel is considered a manual process. The authors deployed and configured their vertical service in 266 seconds, taking out of the equation the time needed to gather the Wireguard keys and invoke the operations required to establish the tunnel. During the evaluation of their solution, the authors of [7] compared the performance of their approach, using Wireguard, to the usage of OpenVPN [2] to achieve the same scenario. Their experiments suggest that WireGuard has a 5.3 times higher throughput than OpenVPN and provides a 41% decrease in the network's latency when compared to OpenVPN.

III. Aim of This Work

Based on the issues identified in Section II, we now present the goals of our work.

Our first objective is to provide a system capable of deploying inter-domain SFCs. This system should also provide the needed capabilities to establish connectivity between the SFCs spanning across the independent domains. This was already achieved in [5], [2], and [8], but we still aim to develop a system with the same capabilities.

Our second goal is to provide the tools needed to establish inter-domain connectivity. These tools will work on top of the previously described system by leveraging its capabilities. Furthermore, the inter-domain connectivity mechanism must also be as resilient as possible. Thus, it cannot rely on a central entity during its operation, as it happens in [5]. However, we need a central entity to coordinate the configuration of the VPN tunnels, but this entity will only be required during the configuration phase, not during the operation of the tunnels.

Our third and last objective is for the VPN tunnels to be properly configured without the need for human intervention. This will result in a truly zero-touch VPNaaS, contrarily to the ones presented in [2], [7], [4], and [8], where manual configuration of the VPN tunnels is required.

Our work presents a solution that fulfills the three abovementioned goals. It is true that other works have already achieved some of the objectives we described. However, to the best of our knowledge, no other work fulfilled all these three goals simultaneously, and thus the motivation behind our efforts.

[2]https://openvpn.net

IV. An Inter-Domain Mesh VPN as a Service

To tackle the orchestration of an inter-domain solution, we used NetOr [3]. NetOr is an Operations Support System (OSS)/Bussiness Support System (BSS) system that operates on top of operators' Fifth Generation (5G) infrastructures, providing a platform for orchestrating Vertical Services. Besides this, NetOr can also fully tackle inter-domain scenarios by coordinating the NFVOs that reside in the different domains.

Regarding the VPNaaS solution itself, we chose to orchestrate a full-mesh VPN, since it provides more throughput and resilience than a non-full-mesh VPN and a Hub-and-Spoke VPN. The VPN tunnels rely on the Wireguard VPN, given its high throughput, configuration simplicity, and lightweightness. This decision is further backed up by the results of [7] and by [9], which showcase that Wireguard provides more throughput and less latency than OpenVPN and IPsec.

By leveraging NetOr and Wireguard-enabled full-mesh VPNs, we have already addressed the first two proposed objectives. To automate the key exchange process between the Wireguard VPN nodes, we relied on a central entity available to all VPN nodes. This entity provides Service Discovery (SD), a core principle of microservice architectures. However, NetOr did not offer any SD mechanism, so we extended NetOr to support it. We studied several SD technologies, such as Eureka, Apache ZooKeeper, and HashiCorp Consul, but concluded that a simple Domain Name System (DNS) Server suffices our needs. Besides, SD using DNS is already standardized in Internet Engineering Task Force (IETF)'s RFC 6763 [10], simplifying our VPNaaS mechanism's development.

By employing DNS-Based SD, we can comply with our third and last objective: the automated establishment of the VPN tunnels with no human intervention.

A. Architecture

To offer DNS-Based SD in NetOr, we added a new component to it – the *DNS Server*. This component was implemented using the PowerDNS[3] DNS Server, given that it is an open-source technology and provides a straightforward management REST API.

Regarding our VPNaaS solution, each mesh's VPN node is achieved through a VNF enclosed in a Network Service (NS) deployed in all domains we aim to connect. After the deployment of a VPN node, it will register its (i) location, (ii) public key, and (iii) information on the networks that must be made available through the tunnels in the DNS Server offered by NetOr. After the registering step, the VPN nodes will start to pull information from the DNS Server to find other additional VPN nodes that may come to life, and establish VPN tunnels with those nodes.

[3]https://www.powerdns.com/

B. Implementation

The implementation of our VPNaaS solution was tailored to the NFVOs we aimed to leverage during its orchestration. So far, we intend our solution to be deployed using OSM. Thus, we relied on its VNF Manager (VNFM), Juju, to configure our solution. It is through Juju Charms that we install Wireguard in the VNF nodes and achieve their remaining workflows. However, this does not mean that our VPNaaS solution depends on this NFVO since, if necessary, it can quickly and straightforwardly be adapted to be deployed by another NFVO.

To orchestrate our VPNaaS solution, we rely on day-0, day-1, and day-2 operations. Day-0 configurations set up the correct instantiation resources for the VPN Nodes. It is through day-0 configurations that we select the IP that should be attached to the VPN node VNFs, to be sure this IP can be publicly exposed. Day-1 operations are then used to install and configure Wireguard in the VNFs. These operations rely on instantiation parameters that NetOr added before the instantiation requests were forwarded to the NFVOs that reside in the domains we wish to interconnect. Among these parameters, one may find information on how to access and interact with the DNS Server that NetOr provides and a cipher key that may be used to publish sensitive data in the DNS Server records. It is through day-1 operations that the VPN nodes will publish their information in the DNS Server and collect information on all the other VPN nodes. It is expected that after all day-1 operations take place, all VPN tunnels are established, and inter-domain communication is achieved. Finally, day-2 operations can be used to manually configure the VPN tunnels further.

As previously stated, we upgraded NetOr [3] to be able to orchestrate our VPNaaS solution. Upon a request to instantiate a Vertical Service, NetOr's *Coordinator* component creates a zone for that specific service on the DNS Server. This component will also inject instantiation parameters on the original request, which are required to enable the abovementioned operations. Among the injected parameters is the cipher key, which is individual to the specific Vertical Service, meaning that all service's VNFs will be injected with the same key and can use it to share protected data between themselves. Subsequently, the *Coordinator* will request the *Domain* component to forward the extended instantiation requests to the NFVOs that live in the domains one wants to interconnect. Once the instantiation requests reach the NFVOs, these will deploy the VPN nodes and delegate their configuration to the NFVOs' VNFMs. Which, in this case, are the Juju controllers provided by OSM. After this phase, the aforementioned day-1 operations can take place. These operations are provided via a Juju Charm that (i) installs Wireguard in the VPN nodes, (ii) registers the VPN node's information in NetOr's DNS Server, and (iii) collects all the information required to establish the VPN full-

mesh network between all VPN nodes. To better elucidate the reader on the VPN nodes' deployment workflow, we provide a sequence diagram in Fig. 1.

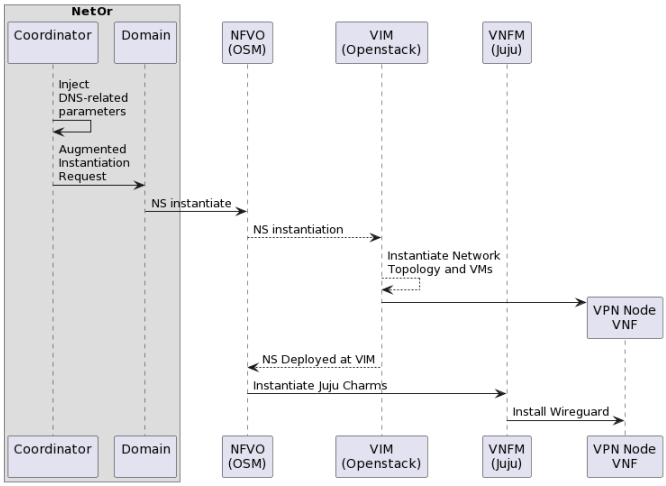

Fig. 1: VPN Nodes' Instantiation Workflow

After Wireguard and all its dependencies are installed in a soon-to-be VPN node, the Wireguard configuration process takes place. During this phase, a private and a public key are created. These will be used by Wireguard to encrypt the network traffic that will be forwarded through the VPN tunnels. Once this is achieved, the Wireguard service is started, but no tunnels between the VPN nodes are established. To achieve this, the VPN nodes have to share their (i) public key, (ii) location, and (iii) information on the networks that should be accessible to the other VPN nodes with all nodes that should be part of the mesh VPN. At this phase, the cipher key injected by NetOr becomes necessary since it enables the VPN nodes to encrypt secret information, such as their public keys. It is worth remembering that the sharing of the nodes' information obeys the IETF RFC 6763 standard.

After a VPN node publishes its information to NetOr's DNS Server, it will have to remain probing this entity to look for other available nodes. Once a new VPN node is found, the process of establishing a VPN tunnel takes place.

This mechanism heavily simplifies the process of adding new VPN nodes to the mesh network, which provides a high degree of automation and dynamism to our VPNaaS solution.

V. Experiments and Results

A. Experiments and Results

To validate our VPNaaS solution, a series of experiments were conducted. During these experiments, we orchestrated a mesh VPN between 3 independent domains: (i) the Telecommunications Institute (TI) in Portugal, (ii) the University of Murcia (UoM) in Spain, and (iii) the University of Patras in Greece (UoP). To orchestrate our solution, we relied on NetOr, which was hosted in TI's domain. NetOr was running in a Virtual Machine (VM) with 4 vCPUs and 8GB of RAM. Regarding the NFVOs used throughout these experiments, TI provided an OSM Rel11 deployed in an Ubuntu 20.04 VM with 8 vCPUs and 16 GBs of RAM. UoM's NFVO was an OSM Rel 10 deployed in an Ubuntu 20.04 VM with 12 vCPUs and 8 GBs of RAM, and UoP provided and OSM Rel 10 deployed in an Ubuntu 20.04 VM with 4 vCPUs and 8 GBs of RAM as their NFVO.

All NFVOs delegated the deployment of the VPN node VNFs to local Openstacks. TI's Openstack was a 3-node cluster with 96 vCPUs and 768 GB of RAM, all equally distributed among the cluster nodes. On the other hand, UoM provided a 2-node Openstack cluster with 160 vCPUs and 512 GB of RAM, all equally distributed. Finally, UoP's Openstack was an 8-node cluster with 450 vCPUs and 1.5 TB of RAM, all equally distributed among the cluster nodes. Regarding networking, all Openstacks were limited to 1Gbps Network Interface Controllers (NICs).

We orchestrated a VPN node VNF in each of the described domains. All these VNFs were deployed as an Ubuntu 20.04 VM with 8 vCPUs and 2 GBs of RAM. To simulate an inter-domain SFC, we also deployed one VM with 2 vCPUs and 2 GBs of RAM, in each domain. These VMs were used to validate the performance of the inter-domain VPN tunnels established by our VPNaaS solution. The scenario in which we conducted our experiments is showcased in Fig. 2.

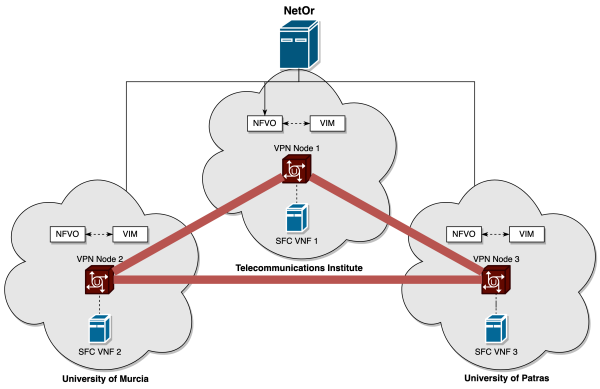

Fig. 2: Experimentation Scenario

We first started by analyzing the instantiation time of our VPNaaS solution. To measure this, we orchestrated it 10 times and collected the time deltas between the initial orchestration requests and the moment when all desired VPN tunnels were established.

Regarding the instantiation process, we considered that it encompassed two phases: (i) the deployment of the VPN node VNFs and (ii) the configuration of these nodes and establishment of the VPN tunnels. The VPN nodes' deployment time encompasses all operations since the initial instantiation request was submitted until the the VPN

node VNF is deployed and its Execution Environment is operational, i.e., all the Juju-related artifacts were created and are operational. On the other hand, the time needed to establish the VPN tunnels, considers all the configuration of the VPN nodes and further VPN tunnel establishment. Table I presents these measurements.

Domain	Δt for VPN node's instantiation	Δt for VPN node's configuration and VPN tunnels establishment
TI	321.12 ± 9.86 s	43.61 ± 10.52 s
UoM	324.05 ± 9.92 s	38.20 ± 9.97 s
UoP	285.61 ± 7.31 s	83.12 ± 11.50 s

TABLE I: VPN Nodes Instantiation and Configuration Time

From Table I, the reader may observe that the different VPN nodes in the different domains take different times to establish all VPN tunnels. However, we only consider that our VPNaaS solution is fully operational when all the VPN nodes have established all desired VPN tunnels. This is related to the fact that a bi-directional VPN tunnel is required to enable connectivity between two domains. Considering this, the overall instantiation time of our solution is 374.51 ± 10.49 seconds.

Then we performed a collection of performance tests. These were conducted from an end-to-end perspective, meaning they evaluated the connectivity between the SFC member VNFs. To evaluate the network's performance, the following aspects were considered: (i) the maximum available bandwidth, (ii) the network jitter, (iii) the percentage of lost packets, and (iv) the packets' Round Trip Time (RTT). The first two tests relied on the *iPerf3* tool, whereas the last two relied on the *ping* tool.

During the performance testing phase, we considered 2 scenarios: (i) all SFC member VNFs communicate through the VPN tunnels established by our VPNaaS solution, and (ii) all SFC's VNFs were publicly exposed, thus communicating through their public interface. The last scenario was introduced since several organizations use that approach to enable communication between SFC member VNFs that reside in different domains, since it is an approach that can easily be automated. However, this scenario poses severe security risks. Thus, it should be avoided at all costs. Still, during our experimentation phase, we compared both approaches' network performance to study the impact of providing an inter-domain communication layer using VPN tunnels. All these experiments were executed in batches of 10 sequential test executions that evaluated a specific network capability for 5 minutes.

Table II presents the measured throughputs during our experimentation, while Tables III and IV present the jitter measurements and the RTT, respectively. Regarding the percentage of lost packets, all measurements revealed a 0% loss in network packets.

Throughput - Public Interface			
Domains	**TI**	**UoM**	**UoP**
TI		122.79 ± 29.22	51.29 ± 2.40
UoM	49.92 ± 10.76		86.87 ± 14.93
UoP	42.26 ± 10.98	96.46 ± 16.73	
Throughput - Private Interface			
From \ To	**TI**	**UoM**	**UoP**
TI		99.49 ± 3.63	44.22 ± 6.53
UoM	67.61 ± 13.32		107.10 ± 15.25
UoP	52.4 ± 11.22	104.02 ± 29.39	

TABLE II: Measured Network Throughtput (Mbits/s)

Jitter - Public Interface			
Domains	**TI**	**UoM**	**UoP**
TI		0.40 ± 0.17	0.42 ± 0.12
UoM	0.89 ± 0.94		0.19 ± 0.03
UoP	0.50 ± 0.98	0.15 ± 0.03	
Jitter - Private Interface			
From \ To	**TI**	**UoM**	**UoP**
TI		0.54 ± 0.30	0.58 ± 0.9
UoM	0.46 ± 0.45		0.18 ± 0.13
UoP	0.35 ± 0.19	0.09 ± 0.07	

TABLE III: Measured Network Jitter (ms)

RTT - Public Interface			
Domains	**TI**	**UoM**	**UoP**
TI		28.70 ± 0.56	77.95 ± 0.43
UoM	28.70 ± 0.56		129.17 ± 0.16
UoP	77.95 ± 0.43	129.17 ± 0.16	
RTT - Private Interface			
Domains	**TI**	**UoM**	**UoP**
TI		25.10 ± 0.45	72.44 ± 0.96
UoM	25.10 ± 0.45		77.91 ± 0.84
UoP	72.44 ± 0.96	77.91 ± 0.84	

TABLE IV: Measured RTT (ms)

B. Results Analysis

By analyzing Table I, it is perceptible that the initial instantiation of the VPN node VNFs takes considerable time compared to the one needed to configure Wireguard and establish the VPN Tunnels. In fact, the instantiation of the VNFs comprises around 82% of the overall instantiation time of our solution. However, given that our solution is NFV-based, it is always limited by the NFVOs and the underlying infrastructure conditions where the VPN nodes are deployed, which is an aspect out of our control. This limitation is widely known in the NFV community and has been addressed in works like [7].

When only considering the time needed to configure the VPN node VNFs and establish the VPN tunnels, contrastingly, results are positive, showcasing that it takes, on average, 55 seconds for a VPN node to establish all desirable VPN tunnels. Considering our VPNaaS solution's overall instantiation time, we achieved a full-mesh inter-domain VPN in less than 7 minutes. We can compare our results with the ones of [7], where the authors achieved

the configuration of their VPN tunnel in 266 seconds, taking out of the equation the time needed to gather the Wireguard keys. Considering that only one VPN tunnel was configured in [7] and that our results encompass the time needed to gather the Wireguard keys, we can affirm that our solution's deployment and configuration time is somehow similar to the one of the solution presented in [7]. We must reinforce that our solution is also fully automated, contrastingly to the solution presented in [7].

Regarding the network performance provided by our solution, the obtained results differed from the expected ones. When considering the network's throughput, the experiments showed that the VPN tunnels increased the bandwidth available to the SFC VNFs. Only the tunnels directed to TI's domain provided less bandwidth than the one achieved when the SFC VNFs communicated through their public interfaces. Still, the available bandwidth only decreased by $\sim 17\%$. Similar behavior was also detected when analyzing the network jitter results. Once again, there was an overall improvement in network jitter inside the tunnels than when the packets flowed through the VNFs public interfaces. When considering RTT, our solution provided better performance than when the VNFs communicated through their public interfaces. This analysis reveals that our solution provided an overall network performance increase, which was not expected since VPNs are predicted to reduce the network's performance. However, our results can be justified by the domains' firewall configurations. All domains are protected by robust firewalls that perform packet inspection and throttling. This leads to a loss in network performance.When using Wireguard, the network packets are encrypted when they pass through the firewall and can only be decrypted once they reach the VPN tunnel's end. Hence, a firewall cannot thoroughly inspect these packets, only forwarding them to their target. This enables the saving of precious milliseconds during the packets' transmission, which explains the obtained results.

Ideally, we would also compare our solution's network performance with the one of the other State of the Art solutions. However, doing so would require us to mimic the exact network conditions where the other solutions were tested, which is troublesome since our tests were performed in network domains owned by universities, and thus cannot be easily tailored to the intended scenarios. As such, we could then compare the performance of the VPN service we relied upon against other VPN tools. However, this is already addressed in [7] and [9]. These works state that Wireguard has better performance than OpenVPN and IPSec, thus we choosing Wireguard to implement our solution.

VI. Conclusions

This work presents a zero-touch NFV-based full-mesh VPNaaS solution. This solution enables the easier deployment and configuration of inter-domain SFCs by providing end-to-end and secure communication channels to the SFC VNFs. Furthermore, it automatically orchestrates a full-mesh VPN in under 7 minutes and without human intervention.

Network performance-wise, our solution has proven it does not heavily injure the communication performance between SFC VNFs deployed in different domains.

VII. Future Work

Even though we achieved positive results with our VPNaaS solution, our work is not completed. We aim to tackle the operational issues that may arise during the employment of this solution. Such an example is the case when one of the VPN tunnels becomes offline, and one ceases having a full-mesh scenario. To deal with this, we will provide monitoring capabilities to the VPN nodes, such as it is possible to detect when a tunnel is offline and take the required measures to re-establish it. Furthermore, we also aim to implement routing protocols to our solution, so as to improve the tunnels' network performance.

Acknowledgment

This work was supported by the European Horizon 2020 Programme for research, technological development and demonstration under Grant 101016448 (5GASP).

References

[1] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of Network Function Virtualization," *Computer Networks*, vol. 133, pp. 212–262, 2018.

[2] A. Huff, G. Venâncio, V. F. Garcia, and E. P. Duarte, "Building Multi-domain Service Function Chains Based on Multiple NFV Orchestrators," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2020, pp. 19–24.

[3] D. G. D. C. D. G. João Alegria, Rafael Direito, "NetOr: An Inter-domain Vertical Service Orchestrator for 5G Networks," in *IEEE NFV-SDN Proceedings*, 2022.

[4] R. van der Pol, B. Gijsen, P. Zuraniewski, D. F. C. Romão, and M. Kaat, "Assessment of sdn technology for an easy-to-use vpn service," *Future Generation Computer Systems*, vol. 56, pp. 295–302, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X15002903

[5] K. Ishimura, T. Tamura, S. Mizuno, H. Sato, and T. Motono, "Dynamic ip-vpn architecture with secure ipsec tunnels," in *8th Asia-Pacific Symposium on Information and Telecommunication Technologies*, 2010, pp. 1–5.

[6] A. Detti, A. Caricato, and G. Bianchi, "Fairness-oriented overlay vpn topology construction," in *2010 17th International Conference on Telecommunications*, 2010, pp. 658–665.

[7] S. Haga, A. Esmaeily, K. Kralevska, and D. Gligoroski, "5G Network Slice Isolation with WireGuard and Open Source MANO: A VPNaaS Proof-of-Concept," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2020, pp. 181–187.

[8] A. Francescon, G. Baggio, R. Fedrizzi, E. Orsini, and R. Riggio, "X-mano: An open-source platform for cross–domain management and orchestration," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1–6.

[9] J. Donenfeld, "Wireguard: Next generation kernel network tunnel," Donenfeld, Jason, Tech. Rep., June 2020.

[10] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," RFC 6763, Feb. 2013. [Online]. Available: https://www.rfc-editor.org/info/rfc6763