Computer Standards & Interfaces 83 (2023) 103665

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Analysis and practical validation of a standard SDN-based framework for IPsec management

Gabriel López-Millán [a,*], Rafael Marín-López [a], Fernando Pereñíguez-García [b], Oscar Canovas [c], José Antonio Parra Espín [a]

[a] Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain
[b] Department of Engineering and Applied Technologies, University Defense Center - Spanish Air Force Academy, 30720 Murcia, Spain
[c] Department of Computer Engineering, University of Murcia, 30100 Murcia, Spain

ARTICLE INFO

Keywords:
IPSec
IKE
Management
SDN
Performance

ABSTRACT

The Internet Engineering Task Force (IETF), the international standardization organism for the Internet, has recently approved a standard, RFC 9061, which defines an interface and framework with which to manage IPsec SAs autonomously by using the Software Defined Networking (SDN) paradigm. In this framework, a centralized entity, the controller, sends configuration information to IPsec-enabled nodes in the network in order to create IPsec SAs. Two cases are presented: *IKE-case*, in which the nodes ship an IKE implementation that is configured by the controller or *IKE-less*, in which the controller sends the IPsec SAs directly to the nodes, among other relevant security information.

This paper analyzes both cases in depth, provides a design for the controller's operation based on Mealy state machines and obtains experimental results from a virtualized testbed so as to compare these cases, which are missing parts in the standard.

1. Introduction

Networks are becoming increasingly more complex. For instance, cloud-based datacenters are composed of hundreds or thousands of virtual devices whose communications require protection in order to form a mesh of secure connections [1]. Software-defined Wide Area Networks (SD-WAN) [2] are yet another challenging scenario that oblige enterprises to securely and intelligently direct traffic involving communication gateways across the WAN, thus forming a star of secure connections. These are only two of the many examples that have posed new challenges in the field of network administration, but they remain significant when working to extract the security requirements present in any possible scenario.

Communication between the peers involved, must as has been outlined in the aforementioned examples, be protected using standard protocols such as Internet Protocol Security (IPsec) or Transport Layer Security (TLS). However, the management of the configuration parameters associated with those protocols is an error-prone and non-scalable task when performed manually, especially when considering complex networks with many components. The standardization organism, the *Internet Engineering Task Force* (IETF), through the *Interface to Network Security Functions Working Group* (I2NSF WG), has, therefore, been working to define standard interfaces and a framework with which to manage network devices (from now on *nodes*) that perform security operations (e.g. encrypt communications or block unwanted network activity), called *Network Security Functions* (NSFs), with an envisioned architecture aligned with the *Software Defined Networking* (SDN) paradigm. The contributions presented in this paper are based on our previous work [3] to establish IPsec Security Associations (IPsec SAs), an effort that has undergone a standardization process within the I2NSF WG and has recently been published as Proposed Standard RFC (RFC 9061) by the IETF [4].

The SDN paradigm [5] provides new opportunities to implement complex and secure networks by giving the responsibility of making decisions to a centralized entity named the SDN controller, which operates on a control plane in order to exchange configuration information with network nodes interoperating on a separate data plane. The network administrator defines high-level network security requirements on the application plane, on which general security policies are defined (e.g., protect data traffic between nodes A and B). The SDN Controller (from now on *the controller*) translates these general policies into IPsec specific configurations and sends them to the nodes implementing the functionality on the data plane.

* Correspondence to: Facultad de Informatica, Campus de Espinardo S/N, Murcia, 30100, Spain.
E-mail addresses: gabilm@um.es (G. López-Millán), rafa@um.es (R. Marín-López), fernando.pereniguez@cud.upct.es (F. Pereñíguez-García), ocanovas@um.es (O. Canovas), joseantonio.parrae@um.es (J.A. Parra Espín).

https://doi.org/10.1016/j.csi.2022.103665
Received 28 July 2021; Received in revised form 4 May 2022; Accepted 30 June 2022
Available online 2 July 2022

There are two different operation cases in our proposal [4]. In the *IKE case*, the node implements IKEv2 in order to manage the IPsec SAs and the controller merely provides the necessary configuration that allows IKEv2 to create the IPsec SAs. In the *IKE-less case*, the node is simplified by sending only the IPsec logic, delegating the whole key management operations to the SDN controller. Both cases are suitable for different application scenarios, as will be explained later.

The main goal of this paper is to study and validate the applicability of the standardized model within the IETF regarding the implementation and effective management of IPsec in diverse circumstances. Moreover, this study seeks to complete the standard in two ways: 1) by detailing the formalization of the controller's behavior by means of the definition of state machines for the IKE and IKE-less cases, and 2) by evaluating the proposal with a proof-of-concept implementation in order to extract exhaustive performance results for different use cases and varying network topologies, which are representative of real scenarios. To the best of the authors' knowledge, there is no existing contribution that either describes the controller's complete method of functioning to support an automated operation in order to manage IPsec or performs an exhaustive performance evaluation.

The remainder of this paper is organized as follows. Section 2 contains a description of some of the main background technologies included in the proposal, while Section 3 presents the SDN-based IPsec framework designed for IPsec flow protection. Section 4 describes the implementation of the proof of concept and testbed, and discusses the results of the experiments performed, and Section 5 analyzes related work found in the literature researched. Finally, Section 6 concludes with some key remarks, along with future lines of work in this field.

2. Background

2.1. SDN and IPsec

As stated previously, the SDN paradigm decouples network services into two clearly differentiated functions: the decision-making process (*control plane*) and the data operations (*data plane*). For instance, each router in the networking L3 routing service is traditionally configured to deal with both actions: deciding the output interface to which an input IP packet has to be forwarded, after which the routing protocol (control plane) makes this decision and sends the input packet (data plane), moving it to the output interface. When using the SDN paradigm for the routing service [6], the decision-making process is assigned to the SDN controller on the control plane, which has a complete view of the network, while the router becomes a L2 switching service, moving packets from input to output interfaces according to the controller's decision. Openflow [7] is a standardized protocol that is principally used to send switching information from the controller to the nodes (that is, as a *southbound* protocol). Other alternatives are NETCONF [8] or SNMP [9].

The architecture of the IPsec protocol [10] clearly decouples the functionality of providing the security services to IP packets (data confidentiality, authentication, integrity) from that responsible for key management (key agreement, peer authentication, cipher suite agreement, etc.). IPsec can be used to protect IP traffic between any two IPsec-enabled nodes, which can be two routers/gateways (*gateway-to-gateway*); two end devices/hosts (*host-to-host*) or between a host and a gateway (*host-to-gateway*, also known as road-warrior).

In short, when IPsec is activated on a system, for each outgoing IP packet the system has to detect whether the packet has to be protected. This is done at kernel level, by checking the IPsec policies stored in the Security Policy Database (SPD). If the packet matches an entry (i.e., an IPsec policy) in the SPD and this entry gives instructions to protect the packet, then the system has to apply the security services (encryption, integrity, etc.) that the policy mandates. The information about cryptographic algorithms and keys constitute the IPsec Security Association (IPsec SA), which is stored in another database in the kernel, the Security Association Database (SAD). Something similar occurs for incoming IP packets. It is worth noting that at least two IPsec SAs in the SAD are required for each data flow: one for the outgoing packets and another for the incoming ones. That is, the IPsec SAs are unidirectional. All this process is automatic if the SPD and SAD in the system are ready. Moreover, from time to time, the cryptographic keys have to be renewed for security reasons (*rekeying*).

An important aspect is how the SPD and the SAD are filled in a IPsec-based node. Here, two options are considered: they can be filled manually by an administrator, or they can be filled automatically by the IKEv2 service. The IKEv2 service requires a mutual peer authentication between the IPsec endpoints based on, for example, digital signatures or shared secrets. This information is stored in another IPsec database known as Peer Authorization Database (PAD). IKEv2 additionally defines an IKE Security Association (IKE SA) that is used to perform the mutual peer authentication and then to negotiate the IPsec SAs, which requires additional configuration information (e.g. cryptographic algorithms, credentials for the authentication, etc.). In general, the information needed for the IKEv2 service is also manually configured.

This architecture fits perfectly into the SDN concept. An IPsec protected network is usually composed of one of these two types of nodes: those without an IKEv2 service, signifying that they are manually configured by network administrators; or nodes shipping the IKEv2 service. In the first case, key management is a hand-made and tedious process because cryptographic keys in the SADs have to be manually configured and renewed. In the second case, the IKEv2 application services must also be configured manually in the nodes, although IPsec SAs are managed automatically by IKEv2. A description of how the SDN paradigm can be used to automatize the management of IPsec SAs, regardless of whether or not the IKEv2 service is supported by the network node, is provided in the following section.

2.2. NETCONF

NETCONF [8] is a protocol for the remote configuration of network devices. In the NETCONF architecture, the NETCONF client is the management entity, while the device to be configured plays the server role. Configuration and state data are exchanged in *Remote Procedure Call* (RPC) operations in the form of XML data by following a YANG model [11].

NETCONF defines the usage of RPC operations over datastores. For instance, when a NETCONF client wishes to apply a particular configuration in the server, it sends the RPC $\langle edit-config \rangle$ operation. This operation includes the configuration in YIN format, which is an XML representation of the configuration data that is compliant with a YANG language [12]. Making use of the $\langle edit-config \rangle$ allows the server to directly apply the given configuration to the system and, if the configuration is successfully applied, it replies with an $\langle ok \rangle$ message. NETCONF also provides *notifications*, which are sent directly from the NETCONF server to the client after a specific event occurs. The information contained in these notifications is also defined in the YANG model.

A security transport layer, such as SSH (by default) or TLS, must protect all the configuration and state data exchanged between the NETCONF client and server.

2.3. The YANG data model for IPsec management

Myriads of YANG data models are currently defined by the IETF [13]. In the context of this work, we use the YANG data model for IPsec that is available in RFC 9061 as a result of our standardization activity in the IETF.

RFC 9061 defines the usage framework and YANG data models for the configuration and state data of IPsec nodes in a SDN network. It includes two main models, one for each of the cases described above. In the *IKE case*, the *ietf-i2nsf-ike* model defines the configuration for the

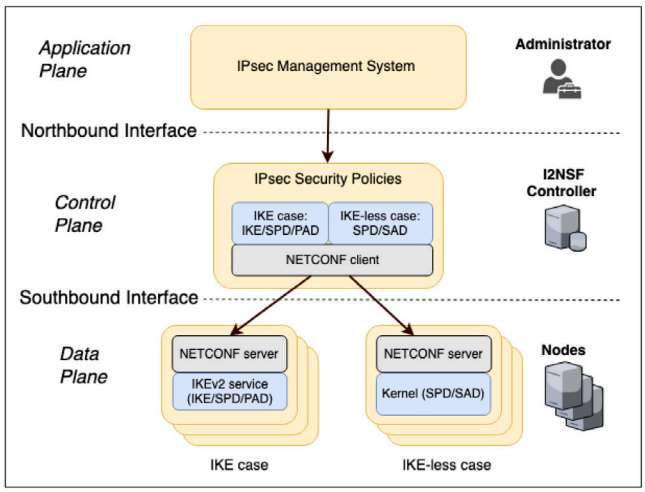


Fig. 1. General architecture.

IKEv2 service, the SPD and PAD at a given IPsec node that ships IKE. In the *IKE-less case*, the *ietf-i2nsf-ikeless* model provides the SPD and SAD configuration for a node without the IKEv2 service. It also includes the required *notifications* for kernel events, such as *sadb_acquire*, when an IPsec SA in the SAD is missing for a given data flow, or *sadb_expire*, indicating the imminent expiration of either an inbound or outbound IPsec SA.

3. The standard framework for SDN-based IPsec flow protection

This section analyzes the most important aspects of this proposal: firstly, the architecture and its components, which are mapped onto the SDN paradigm, and secondly, the operation in detail, by specifying the different exchanges between the entities in the architecture (the controller and the nodes), along with the state machines that define the controller's operation.

3.1. Architecture

Fig. 1 shows the architecture for the SDN-based IPsec management that we describe in RFC 9061. It was conceived in order to support gateway-to-gateway and host-to-host scenarios, leaving host-to-gateway scenarios as future work.

Following the SDN paradigm, the proposed framework moves the automated key management process so as to establish IPsec SAs at a central point (control plane), while IP traffic flow protection (e.g. encryption, integrity and authentication operations) and forwarding logic are placed in the nodes (data plane). Nodes on the data plane communicate by means of the data network in which all nodes are connected. The controller can be connected to this network, or to a dedicated management network. NETCONF is used as a *southbound* protocol.

As stated above, there are two cases that have been considered during the standardization process in the I2NSF WG. In the first case, the *IKE case*, the controller relays part of the key management process in the IKEv2 implementation deployed at the nodes (*IKEv2 service*). The controller, therefore, sends any configuration that IKEv2 requires (keys to perform authentication, IPsec policies information, expected IPsec SA lifetimes, etc.), thus enabling it to establish the IPsec SAs as specified by the controller. In the second case, the *IKE-less case*, the controller performs the key management process by itself since the nodes do not have any IKEv2 service, only a bare-IPsec implementation in the kernel, as specified in RFC 4301 [10]. As such, the controller sends not only the IPsec policies but also the IPsec SAs to the nodes. Both alternatives have use cases, as described in [14]. In fact, although the IKE case is the most likely to be deployed, the second one is useful in some scenarios. For example, when the IPsec peers are memory-constrained nodes or when the network link is constrained and cannot afford an IKEv2 negotiation.

The high level requirements for IPsec protection of flow traffic are defined by the network administrator on the application plane, where general security policies are defined (e.g., protect data traffic between nodes A and B). These general policies are then translated by the controller (control plane) into specific IPsec-related configuration for the nodes. Depending on the case, the controller must provide the following information to the nodes:

- *IKE case*
  - IKE: Configuration for the IKEv2 service, such as IKEv2 and IPsec SAs lifetimes, supported IKE SA cryptographic suites, IKEv2 key material for authentication (pre-shared keys, certificates, etc.).
  - PAD: Identity and authorization data, such as node's network addresses, authentication methods and credentials, etc.
  - SPD: IPsec policies for IKE, which include traffic selectors, AH or ESP protection, tunnel or transport mode, IPsec SA cryptographic suites, etc. Only one IPsec policy element is installed for the outbound traffic flow (the inbound policy is learnt by the node during IKE negotiation).
- *IKE-less case*:
  - SPD: IPsec security policies, such as traffic selectors, tunnel or transport mode, IPsec SA cryptographic suites, etc. One policy per traffic flow direction.
  - SAD: IPsec SA details, such as traffic selectors, inbound and outbound IPsec SA, tunnel or transport mode, AH or ESP protection, cryptographic key material for the IPsec SA, etc. One SA per traffic flow direction.

In RFC 9061 we also define the communication interface between the controller and the nodes (i.e., the southbound interface), and the YANG data model that makes it possible to configure the proposed cases in the nodes, Section 2.

3.2. Workflow operation

This section describes the general steps required to complete the establishment of IPsec SAs between two nodes using a centralized entity in the IKE and IKE-less cases. Moreover, two operation modes are considered in the IKE-less case: *proactive* or *reactive*. In the proactive mode, the centralized entity, the controller, sends the IPsec policies and the IPsec SAs to the nodes simultaneously, thus ensuring that they are ready to protect traffic flows between them. In the reactive mode, the controller first sends the IPsec policies to the nodes, and only when it is necessary, as per node's request, does the controller send the IPsec SAs.

This basic operation, that is, the configuration of two nodes, is not a limitation of our framework, but provides flexibility to form different types of scenarios that can be found in distinct uses cases (see Section 4). The assumption as regards the controller's operation is that it can send the configuration information to both nodes in parallel before receiving an answer from one of them, in order to reduce the configuration time. Additionally, it is worth noting that each NETCONF message includes only the information required to establish the IPsec SAs for a pair of nodes.

This section also describes the controller's operation in detail. In fact, during the standardization of the interface between the controller and the nodes, one of the main discussions was the difficulty involved in expressing the controller operation in these cases, especially for the IKE-less case, taking into account potential errors or unexpected situations. In fact, with the two cases in question, the controller varies

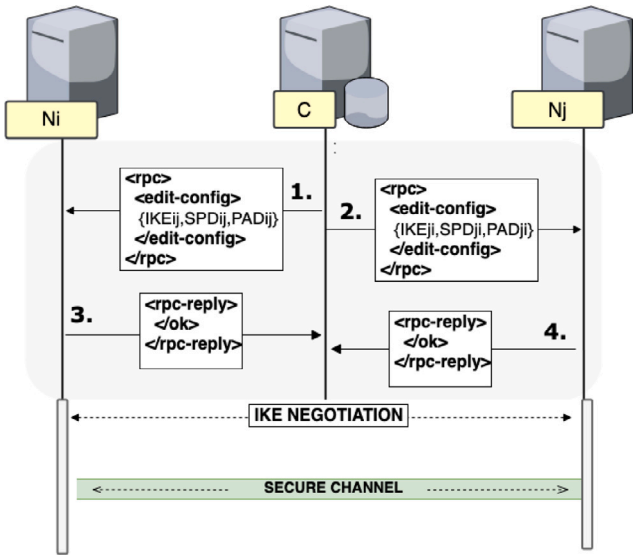


Fig. 2. IKE case.

in complexity. It is worth noting that, owing to its complexity, all matters related to IPsec SAs rekeying process are beyond the scope of this paper and are consequently left as future work.

In the IKE case, the controller needs to send configuration information to the IKEv2 service application in the node, thus allowing IKEv2 to establish the IPsec SAs. As such, the IPsec SAs establishment itself is delegated to the IKE service by the controller. This implies a simpler design of the controller operation. Furthermore, the controller's design is more complex for the IKE-less case, since the IPsec SAs establishment process must be carried out entirely by the controller without relying on any part in the node. In order to analyze these differences, we have formalized the behavior of the controller by means of the definition of state machines.

Each state machine describes how the configuration of two nodes is performed by the controller, and is useful to detail its Fbehavior. The state machines are defined with states and actions in the transitions (Mealy state machine). That is, the state machine evolves from one state i to another state j after a particular event e. During the transition and before reaching state j, action $a()$ is performed.

3.2.1. IKE-case

Fig. 2 shows the different exchanges that a controller C performs in order to configure IKE for two nodes Ni and Nj. The controller sends a NETCONF $\langle edit-config \rangle$ operation with the XML configuration to each node in parallel (*steps 1 and 2*), which includes the information about IKE, the SPD (IPsec policies) and the PAD (i.e., *IKEij, SPDij, PADij*). If a node successfully applies the configuration, it confirms this with a $\langle rpc - reply \rangle$ with $\langle /ok \rangle$. Once one of the nodes has been configured, and according to the IKE configuration sent from the controller, the node applies this information in the IKEv2 service application (e.g. Strongswan, Libreswan, etc.) and can automatically start the IKE negotiation with the other peer in order to generate the corresponding IPsec SAs.

Fig. 3 describes the specific state machine that makes it possible to visualize the controller's operation for the IKE case in more detail. It is a simple state machine with 4 states and 7 transitions with the corresponding actions. When the event $START$ is received (the controller is instructed to configure Ni and Nj), the state machine performs the action (**transition 1**) $install()$. This action creates and sends an $\langle edit-config \rangle$ message with the XML files based on the YANG data model for the IKE case to configure Ni and Nj simultaneously. If the controller receives an $\langle rpc - reply \rangle$ with $\langle /ok \rangle$ (event OK), the state machine moves to state $WAIT\_OK2$ in which the action $log\_ok()$ is performed to merely register that the configuration for that node has been successfully applied (e.g. Ni). The $WAIT\_OK2$ state is defined in order to wait for the next OK coming from the other node (e.g. Nj). These steps and **transitions 1, 2 and 3**, therefore, correspond to the operation described in Fig. 2, which is the situation in which no errors occurred during the process.

However, some errors may occur during the configuration. For example, an $\langle rpc - reply \rangle$ with $\langle /error \rangle$ is sent by the node to the controller; a timeout without receiving any answer from the node or a node has simply crashed (event ERR). **Transitions 4, 5, 6 and 7** deal with these situations. More specifically, **transition 4** occurs when the controller firstly detects that it was not able to apply the configuration to a node (e.g. Ni), registering this fact ($log\_err()$). If the controller receives an OK from the other node (e.g. Nj), this means that the configuration has been applied there (**transition 5**). In this case, the controller must perform a $rollback()$ operation of the configuration sent to Nj (it is no longer useful because the configuration in Ni failed). A similar situation may occur when the OK is received first (**transition 2**) by a node and ERR then occurs when configuring the other (**transition 6**). Again, a $rollback()$ action is required to remove the configuration installed in the node that sent the OK.

Finally, **transition 4** and **transition 7** deal with the case in which two errors have occurred during the configuration process. This implies that no configuration was applied to either node and the controller can only report the error ($log\_err()$).

3.2.2. IKE-less case proactive mode

In this case, the nodes do not deploy an IKEv2 service. Fig. 4 shows how, once the nodes are registered, the controller C must send the IPsec policies and the IPsec SAs for both nodes Ni and Nj in parallel (*steps 1 and 2*).

When the nodes receive the IPsec policies (i.e., *SPDij, SPDji*) and the information about the IPsec SAs (i.e., *SADij, SADji*), the NETCONF server translates them into specific IPsec kernel instructions for the installation, and the nodes stay ready to protect the incoming data flows. If the information received was correct, the nodes answer with an $\langle rpc - reply \rangle$ with $\langle /ok \rangle$ (*steps 3 and 4*).

In essence, the controller's behavior and, therefore, the shape of the state machine, do not differ from that defined for the IKE case (see Fig. 3). The differences concern the operations carried out by the actions. On the one hand, $install()$ prepares the XML files with the IPsec policies for the nodes' SPD and the IPsec SAs for the nodes' SAD. On the other, in the case of ERR, $rollback()$ removes the IPsec policies from the SPD, along with the IPsec SAs installed in the SAD of the node that was successfully configured.

3.2.3. IKE-less case reactive mode

Fig. 5 shows the workflow for this mode, in which the controller C first sends the IPsec policies to the SPD in the nodes (*steps 1, 2, 3 and 4*) but waits for a node to send an $acquire$ notification (*step 5*), indicating that an outgoing data flow needs to be protected and that the IPsec SAs in the SAD need to be installed (*steps 6 to 9*). This may occur in scenarios in which data flows are sporadic and there is no need to deploy IPsec SAs immediately, thus saving some resources in the nodes.

Fig. 6 shows the state machine that defines the behavior for the reactive mode, which is far more complex. The reason for this is related to the fact that the reactive mode is divided into two parts: first, the configuration of the IPsec policies, and second, the configuration of the IPsec SAs only when the nodes require them, that is, after the controller receives an $acquire$ notification from a node (event ACQ). In a general case, **transitions 1 to 3** imply a correct installation of the IPsec policies in the nodes' SPD ($install\_spd()$) in both nodes ($commit\_spd()$).

The controller will then wait for the reception of an ACQ from any two nodes. If an ACQ occurs, then **transition 4** starts the installation of

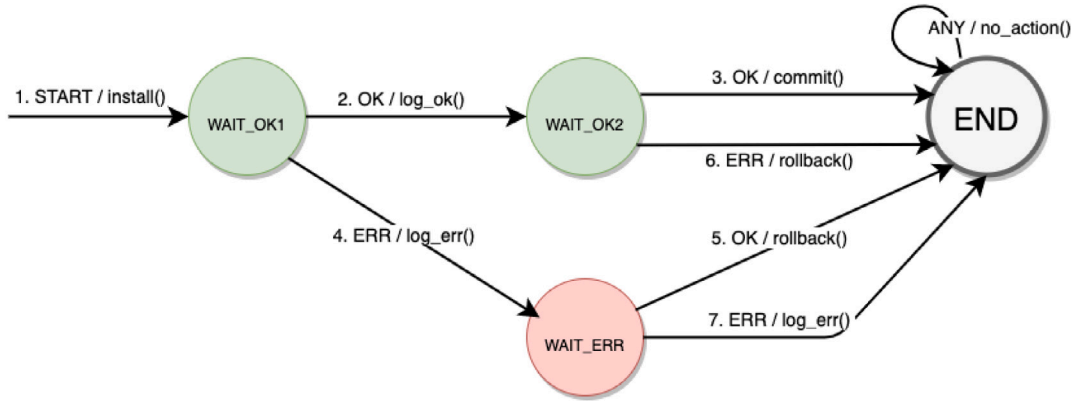


Fig. 3. Controller's state machine for IKE case and IKE-less case proactive mode.

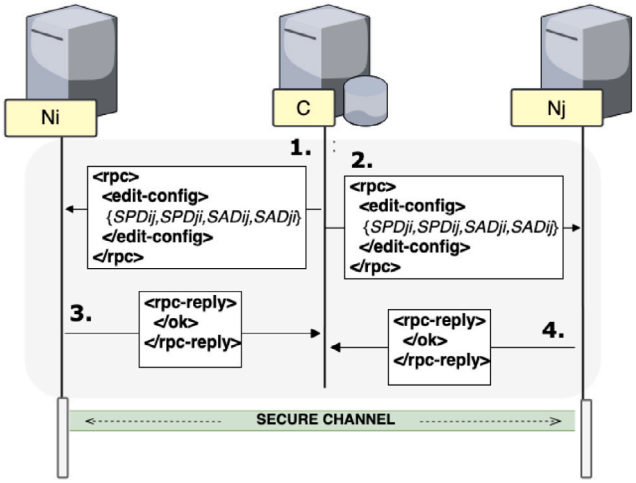


Fig. 4. IKE-less case proactive mode.

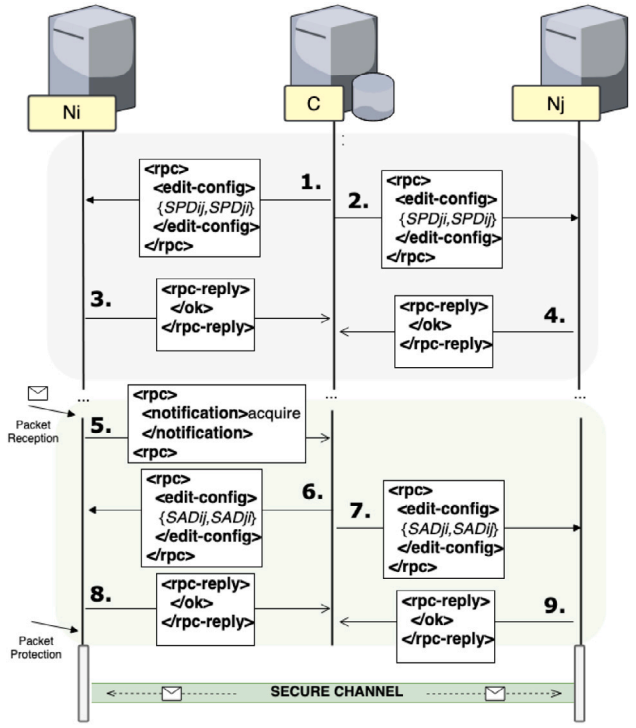


Fig. 5. IKE-less case reactive mode.

the IPsec SAs required in both nodes ($install\_sad()$). **Transitions 5 and 6** imply that IPsec SAs were correctly installed in the SAD of both nodes and that the controller can create a state ($commit\_sad()$) regarding the IPsec SAs installed. These transitions correspond to the case in Fig. 5.

Nevertheless, the controller needs to deal with the following situation: once a node has applied the IPsec policies in the SPD, it may send an *acquire* notification as soon as IP packets matching the configured IPsec policies require IPsec protection and there is no IPsec SA in the SAD to process them. This means that the ACQ may well arrive from any of the nodes (or both) even before receiving an OK from the nodes or even before one of the nodes has been configured with the IPsec policies. The controller state machine has two states with which to deal with this situation: $WAIT\_OK1\_ACQ$ and $WAIT\_OK2\_ACQ$. If an ACQ was received and the SPD configuration finished successfully (**transitions 7, 8 and 10** or **transitions 9 and 10**), the controller stores information for the IPsec policies installed and starts the configuration of the IPsec SAs in the SAD of the nodes immediately ($commit\_spd()$ and $install\_sad()$). At this point, the state machine enters the part dedicated to installing SAD entries.

Finally, the controller needs to deal with errors. If errors occur during the installation of IPsec policies in the SPD, there is no need to proceed with installing IPsec SAs in the nodes (*Transitions 11 to 16*). In these cases, simply reporting the error ($log\_err()$) is sufficient, since either the controller is still waiting for the answer from the other node (*transitions 11 or 12*) or because the controller cannot do anything else as both nodes have failed (*transition 13*). However, if the controller, for example, received OK from Nj and an ERR from Ni or viceversa, a $rollback\_spd()$ is required to remove any state created with the SPD in the node that reported the OK (**transitions 14, 15 or 16**).

If the IPsec policies installation was a success, the IPsec SAs installation process may also find errors, which are dealt with by **Transitions 17, 18, 19 and 20**. Similarly to the previous cases, if the controller received two ERR events (transitions 17 and 18), this means that no IPsec SA was successfully installed in the nodes and the controller needs only to remove the policies from the SPD ($rollbak\_spd()$). If some of the nodes reports an OK (i.e., *transition 17 and 19* or *transitions 5 and 20*), the controller removes not only the IPsec policies installed in both nodes but also the IPsec SAs installed in the node that reported OK.

4. Validation and experimental results

In order to test the validity of this proposal, we have implemented a proof-of-concept testbed with which to extract performance results for the IKE and IKE-less cases in representative network scenarios.

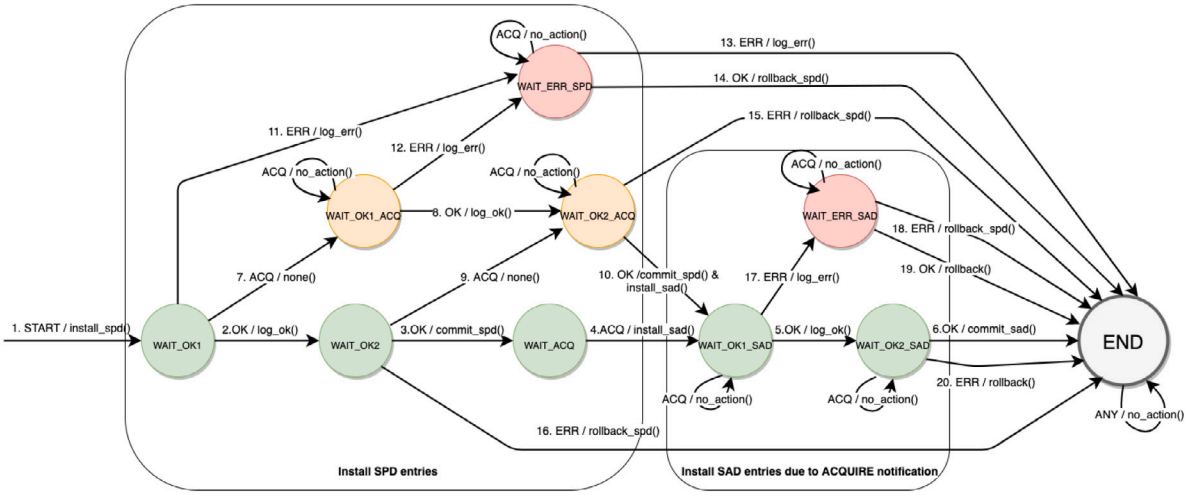


Fig. 6. Controller's state machine for IKE-less case reactive mode.

4.1. Deployed testbed

Fig. 7 shows the general testbed, consisting of an SDN network composed of *N* nodes and a single controller, that we have developed in order to evaluate our framework in two representative scenarios: a mesh and a star topology of IPsec secure channels. The former is typical in datacenters [1], while the latter usually appears in SD-WAN deployments [2]. This virtualized SDN network has been deployed with *Kubernetes v1.20.2*,[1] with one Kubernetes node acting as master and four additional Kubernetes nodes acting as workers and connected to the same cluster. The different microservices are deployed as Kubernetes *pods* following a balanced approach in order to distribute uniformly among the nodes of the cluster. For the sake of simplicity, there is only one network with which to communicate controllers and nodes (data and management networks are the same one). The way in which the existence of different networks might affect the validation results will be analyzed in future work.

The server in which this virtualized testbed has been deployed ships an AMD EPYC 7302P 16-core processor and 256 GB of DDR4 memory. Each virtual machine (one for the master and four for the workers) has 8 GB of memory and 4 cores (kvm64). The virtualization is managed using proxmox v6.3–3 and the virtual machines are interconnected using OpenVirtualSwitch v2.12.0.

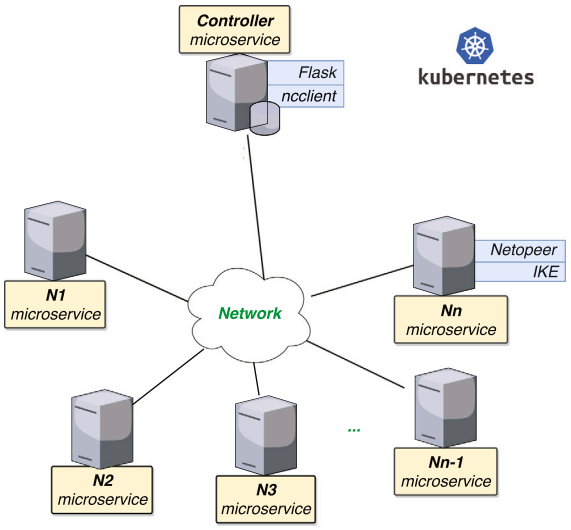


Fig. 7. Overview of the virtual SDN scenario for testing purposes.

4.2. Implementation aspects

With regard to the different elements comprising the testbed, details of the main implementation characteristics are shown as follows.

4.2.1. Controller

The controller is a Ubuntu 18.04-server Docker container deployed as a Kubernetes pod. The main process is a Python program implementing the functionality of the controller (the IKE and IKE-less cases), and it is implemented by following a parallel approach, with different threads running the state machines needed to configure the information required to establish the IPsec SAs between the nodes in the scenario.

The NETCONF client is implemented with *ncclient*.[2] In order to automatize the experiments, we have also implemented a simple registration process for the nodes to notify that they are ready to receive the configuration at a specific network address. This is a REST service implemented with *Flask*.[3] The southbound communication with the nodes is executed in parallel (with a pool of as many threads as nodes).

4.2.2. Nodes

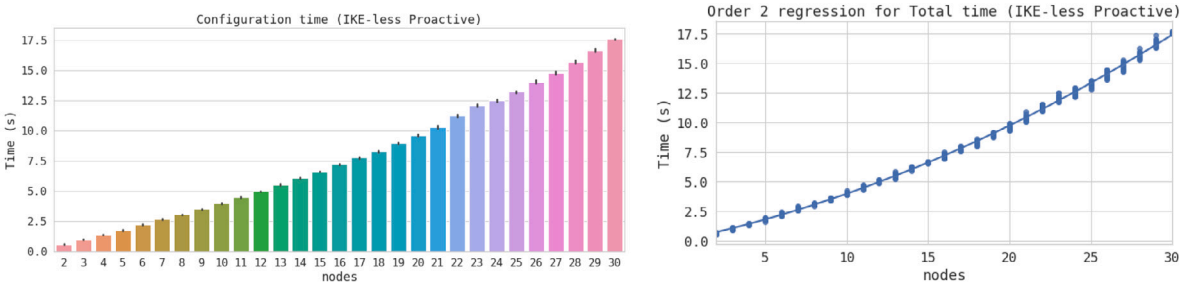
Each N*i* is a Ubuntu 18.04-server container that implements a node (in the SDN terminology, not in the Kubernetes one). It is deployed across the Kubernetes cluster, with a variable number of replicas, as will be described later.

The main service of a node is a NETCONF server based on *Netopeer*,[4] and a NETCONF server application based on *libnetconf2*,[5] and *Sysrepo*[6] which makes it possible to load our proposed YANG data models and schedule callbacks on the arrival of new configurations. In the IKE case, the IKE service application has been implemented using Strongswan v5.5,[7] and experiments have been carried out by making use of IKEv2 node authentication on the basis of a pre-shared key.

4.3. Performance evaluation

In order to illustrate the suitability of the proposed framework for the aforementioned representative SDN application scenarios, we

[1] https://www.kubernetes.io/
[2] https://pypi.org/project/ncclient/
[3] https://flask.palletsprojects.com
[4] https://github.com/CESNET/netopeer
[5] https://github.com/CESNET/libnetconf2
[6] https://github.com/sysrepo/sysrepo
[7] https://www.strongswan.org



(a) Total configuration time (b) Order 2 regression

Fig. 8. Configuration time for a mesh of n nodes using IKE-less case proactive mode.

have defined two *tasks* to be performed by the controller in order to configure the IPsec SAs between nodes.

Each task is representative of the aforementioned SDN scenarios, a mesh and a star topology of IPsec SAs, and they will be used to evaluate the different operational modes, that is, the IKE and IKE-less cases (proactive and reactive modes) with different workloads and conditions.

For each task, we have tested different workloads (number of nodes to be configured). As will be shown, we incrementally vary the number of nodes involved, simulating from lighter to more stressful situations for the controller. The ultimate goal of these experiments is to measure the time that the controller takes to perform the task, that is, to configure the nodes (*configuration time*). All the results for each task are presented and discussed below.

4.3.1. Task 1: Mesh of n nodes

In this task, the controller is committed to configure a mesh of n nodes, which will be fully protected using IPsec.

The controller is able to build a mesh of n nodes using the basic operation, described in Section 3, in order to establish two IPsec SAs between two nodes. In particular, the controller takes a node Ni and establishes IPsec protected communications with each node j, where $1 <= j <= i - 1$. This operation is performed in parallel for each Ni in the mesh, where $1 < i < n$.

As such, each NETCONF message includes only the information required to establish the IPsec connection between a pair of nodes. The framework would also allow the inclusion of multiple configuration elements in each NETCONF message, although this would not be valid if the total number of nodes was unknown or changing. Moreover, this decision is useful because it provides some homogeneity to the performance evaluations comparing different approaches, and also makes it possible to evaluate the worst scenario requiring the highest number of configuration messages.

In this scenario, we have measured the *configuration time* in a mesh topology of different numbers of nodes n. Particularly, we have run 15 executions for each n from $n = 2$ to $n = 30$ in order to observe the behavior and scalability of the system as the number of nodes increases.

For the **IKE-less case proactive mode**, Fig. 8(a) shows the mean time (and confidence interval) required by the controller to complete the configuration (Y-axis) of a mesh of n nodes (X-axis). As explained in Section 3, in order to configure IPsec between a pair of nodes the controller must provide information about IPsec policies (for the SPD) and IPsec SAs (for the SAD). To form a complete mesh of n nodes, the controller must accordingly configure a total of $n * (n - 1)$ unidirectional (inbound or outbound) IPsec SAs ($O(n^2)$ complexity). That is the reason why the configuration time has a quadratic tendency, as shown in Fig. 8(b). For example, according to the data, the controller can configure a mesh network of 30 nodes, that is, a total of 30*29= 870 IPsec SAs, in approximately 17,5 s.

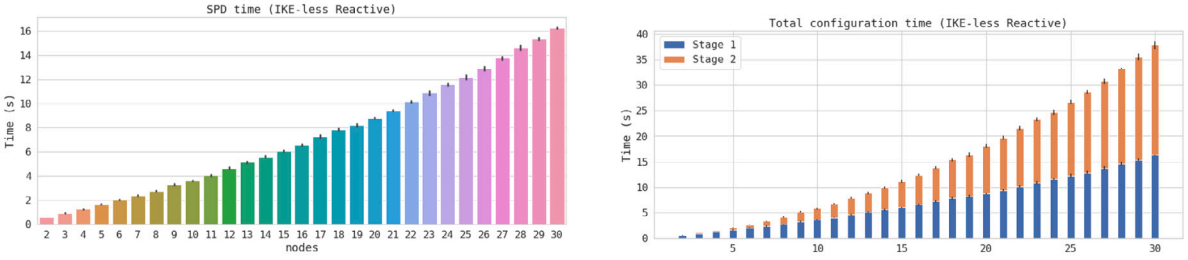
Following the operation of the **IKE-less case reactive mode**, we have divided the experiment into two different stages. *Stage 1* performs the distribution of the IPsec policies, assuming no IP packet will require IPsec protection at that stage. In *stage 2*, in order to represent a case with the highest workload for the controller, we have prepared the experiment in such a manner that each node of the mesh needs to exchange traffic (ping packets) with the remaining nodes at the same time. This involves a massive arrival of *acquire* notifications at the controller and the resulting configuration messages already depicted in Fig. 5.

Fig. 9(a) shows the mean time (and confidence interval) required by the controller to complete the configuration of the IPsec policies (stage 1) in the SPD (Y-axis) of a mesh of n nodes (X-axis) following the IKE-less case reactive approach. For example, the controller can configure this information for a mesh network of 30 nodes in approximately 16 s, a lower value than that of the proactive approach since the information to be exchanged contains less data (only the IPsec policies) and, therefore, requires less processing time. Again, the configuration time has a quadratic tendency, for the same aforementioned reason.

Fig. 9(b) also adds the time required to exchange the configuration of the IPsec SAs (stage 2) once the traffic flow has to be protected simultaneously for all the nodes. As is shown, the total tendency is also quadratic and the time related to the IPsec SAs configuration tends to be higher than the IPsec policy configuration time as the number of nodes in the mesh network increases. This is caused by two different factors: first, the IPsec SA configuration stage is initiated by the reception of *acquire* notifications, as shown in Section 3, signifying that it is a phase in which the controller must process three NETCONF messages (*acquire, edit-config, ok*) rather than 2 messages (*edit-config, ok*); second, for each pair of nodes, Ni and Nj, it is possible to receive two different *acquire* notifications that must be processed (for example when both nodes realize at the same time, or almost, that they need an IPsec SA to protect the IP traffic between them). The controller will configure the necessary IPsec SAs only once, as a result of the first received *acquire* notification, but it will have to process the second one anyway in order to discard it (see Fig. 6 for the IKE-less case reactive mode state machine).

In the **IKE case**, the controller has to provide the configuration for IKE, SPD and PAD. Once this information has been established between a pair of nodes, the IKEv2 negotiation takes place and the IPsec SAs are established eventually. As occurred with the IKE-less reactive approach, the distribution of the configurations does not overlap with the IKEv2 negotiation since no packet yet requires protection. Note that the time devoted to the negotiation is not part of the configuration time from the controller's perspective, since the controller delegates the IPsec SA establishment to the IKEv2 in the node.

Fig. 10 shows the mean time (and confidence intervals) required by the controller to complete the configuration of IKE, SPD and PAD (Y-axis) of a mesh of n nodes (X-axis) following the IKE case. For example, the controller can provide the configuration information of a mesh network of 30 nodes in approximately 13 s, a slightly lower value than that of the previous approaches since the information that IKEv2 needs in order to establish the IPsec SA is less than in the IKE-less case. Whatever the case may be, the configuration time has a quadratic tendency, for the same aforementioned reason.



(a) Stage 1 configuration time (IPsec policies)

(b) Total configuration time (Stages 1 + 2)

Fig. 9. Configuration times for a mesh of n nodes using IKE-less case reactive mode.

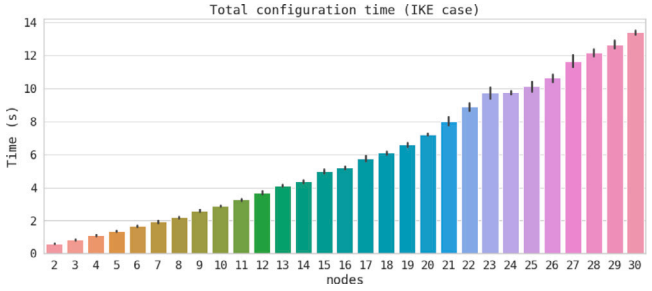


Fig. 10. Total Configuration time for a mesh topology of n nodes using the IKE case.

For the sake of completeness, we have calculated the mean time required to perform an IKEv2 negotiation between all the nodes comprising the mesh network. The resulting time may be considered negligible since the negotiation requires between 2 and 4 ms in most cases.

Having analyzed the results of each approach separately, a comparison of the performance of the different options is now provided in Fig. 11. As will be noted, the total configuration time for the IKE-less reactive mode is much higher, but it is necessary to take into account that it shows the worst scenario, in which all nodes in the mesh request the establishment of IPsec SAs at the same time. In those situations, in which the administrator knows that the nodes will need to establish a full mesh of IPsec SAs immediately, it would be a better option to make use of the IKE-less proactive approach. In a typical scenario in which the IPsec SAs are established sparsely and in a wider range of time, the reactive strategy is more suitable since the configuration time for the IPsec policies (stage 1) is lower when compared to the proactive strategy, and it will perform better when the workload of the controller is not so intensive. Moreover, it avoids populating the nodes' kernel with information about IPsec SAs that may not be used.

In relation to the other approaches, note that the IKE case provides better results, although those times do not include the IKE negotiation time, they refer only to the IKE configuration time. Despite the fact that the IKE-less proactive cases do not perform as well as in the IKE case, it is necessary to take into account that the former provides the node with all the information required to establish the IPsec SAs and there is no need to perform any later negotiation between the nodes.

4.3.2. Task 2: Star topology of n nodes

We have also evaluated the performance of the proposed framework with which to configure IPsec in a SDN network of n nodes deployed following a star topology. In other words, $n-1$ nodes establish individual IPsec connections with a central node, which is the typical situation in certain SD-WAN scenarios [15]. The controller configures each IPsec connection using two messages: one message for the central node and another for the peer node.

As occurred before, we ran 15 executions, from $n = 2$ to $n = 30$, for each network size in order to observe the behavior of the system as the topology size increases.

Fig. 12(a) shows the mean time (and confidence intervals) required by the controller to complete the configuration (Y-axis) of a star topology of n nodes (X-axis) following the **IKE-less case proactive mode**. In this case, connecting $n-1$ nodes with the central node requires the establishment of $2 * (n-1)$ unidirectional IPsec SAs ($O(n)$ complexity). For example, the controller can configure a star topology of 30 nodes in approximately 7 s, which implies configuring $2 * (30-1) = 58$ IPsec policies and 58 IPsec SAs. As expected, the configuration time has a linear tendency ($O(n)$), as shown in Fig. 12(b). This linear relationship also explains the reduction in the required configuration time in relation to the mesh network, which had an $O(n^2)$ complexity.

For the **IKE-less case reactive mode**, we analyze the results in the same way as that employed for the mesh network. We have divided the establishment of the star topology into two different stages: *stage 1* for the distribution of the configurations related to the IPsec policies (SPD), and *stage 2* for the configuration of the IPsec SAs (SAD) after receiving the *acquire* notification from the nodes. We have also prepared the experiment in such a way that all the nodes need to exchange traffic with the central node at the same time.

Fig. 13(a) shows the mean time (and confidence intervals) required by the controller to complete the configuration of the IPsec policies (Y-axis) in a mesh network of n nodes (X-axis) following the IKE-less reactive approach, that is, before any IPsec SA is configured, since there is, as yet, no traffic to be protected. For example, the controller can configure the IPsec policies of a network of 30 nodes in approximately 7 s (configuration of 29*2 = 58 IPsec policies), which is a similar value to that obtained for the IKE-less case proactive mode. The configuration time has a linear tendency, for the same aforementioned reason. Fig. 13(b) also includes the time required to configure the IPsec SAs in the nodes' SAD once the traffic has to be protected simultaneously for all the nodes. There is also a linear tendency for the total time. In this scenario, there is a significantly lower workload for the controller in order to configure the IPsec SAs in the nodes in comparison with the mesh topology, and this time, therefore, shows a better performance of the controller.

In the **IKE case**, we shall follow the same reasoning as that employed in the previous scenario. There are two different stages: IKE configuration and then IKE negotiation. As occurred previously, the distribution of the IKE configuration including the IPsec policies for the nodes' SPD is the task to be performed by the controller, which does not overlap with the IKE negotiation since no packet requires protection yet. For the second stage, we have prepared the experiment in such a way that all the nodes need to exchange IP traffic that needs to be protected with the central node, and the IKE negotiation starts. This has been implemented only for verification purposes, since it is not part of the configuration time from the controller's perspective.

Fig. 14 shows the mean time (and confidence intervals) required by the controller to complete the configuration of IKE, SPD and PAD (Y-axis) for a star topology with n nodes (X-axis) following the IKE case. For example, the controller can configure the IKE information of 30 nodes in approximately 7 s, as occurred with the other methods shown.

Having analyzed the results of each approach separately, a comparison of the performance of the different options is now provided in

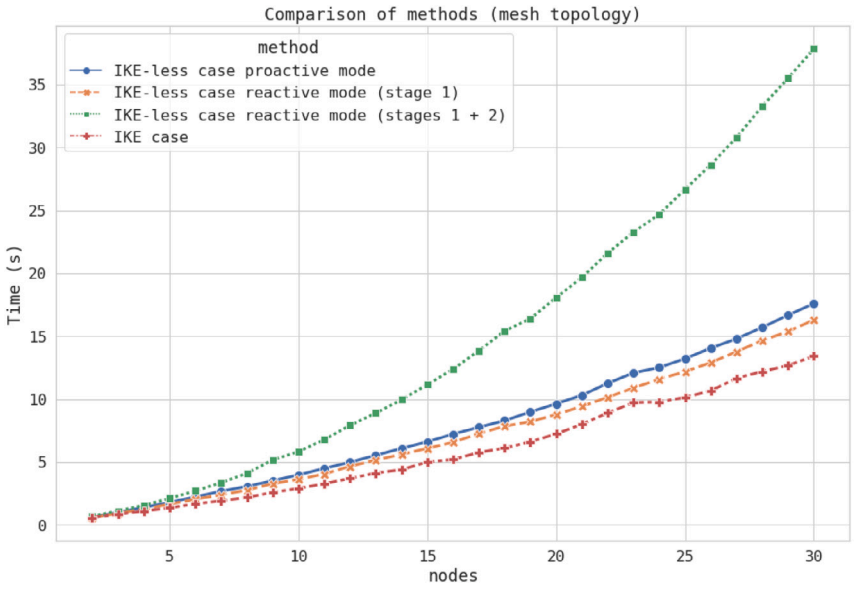
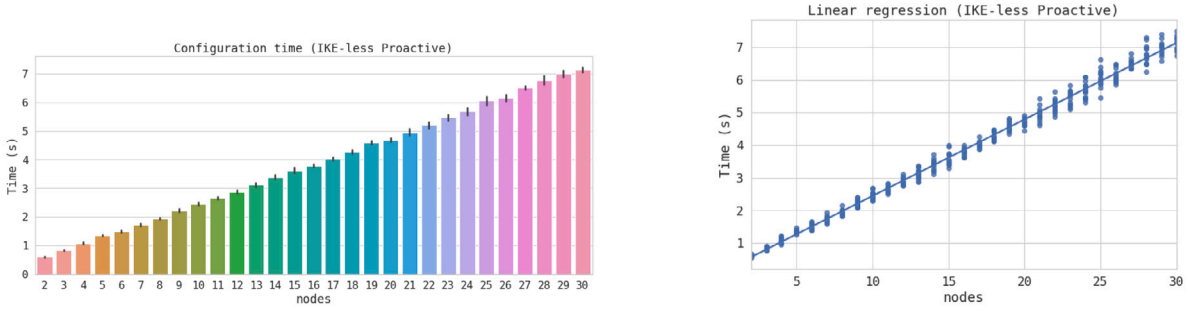
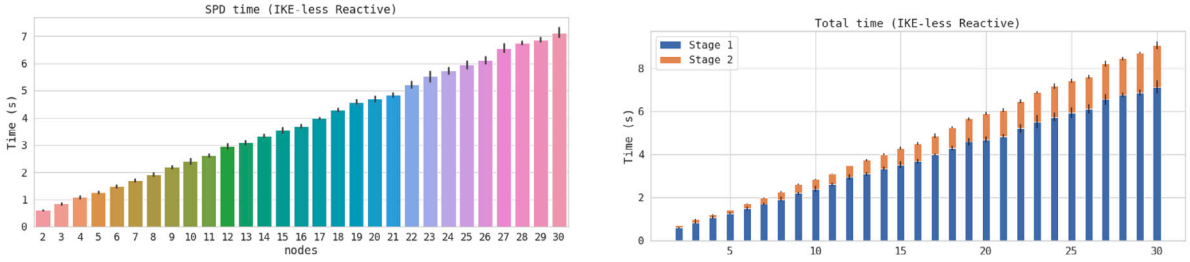


Fig. 11. Configuration times of the different approaches for the mesh scenario.



(a) Configuration time (b) Linear regression

Fig. 12. Configuration time for a star topology of n nodes using IKE-less case proactive mode.



(a) Stage 1 configuration time (IPsec policies) (b) Total configuration time (Stages 1 + 2)

Fig. 13. Configuration times for a star topology of n nodes using IKE-less case reactive mode.

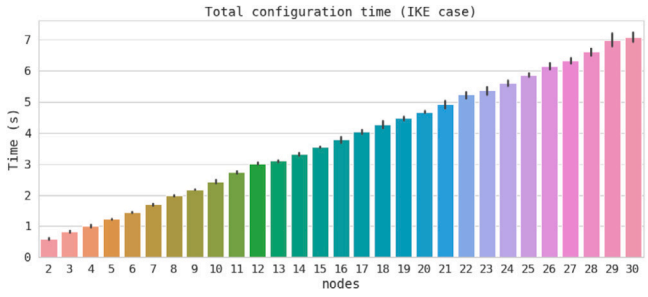
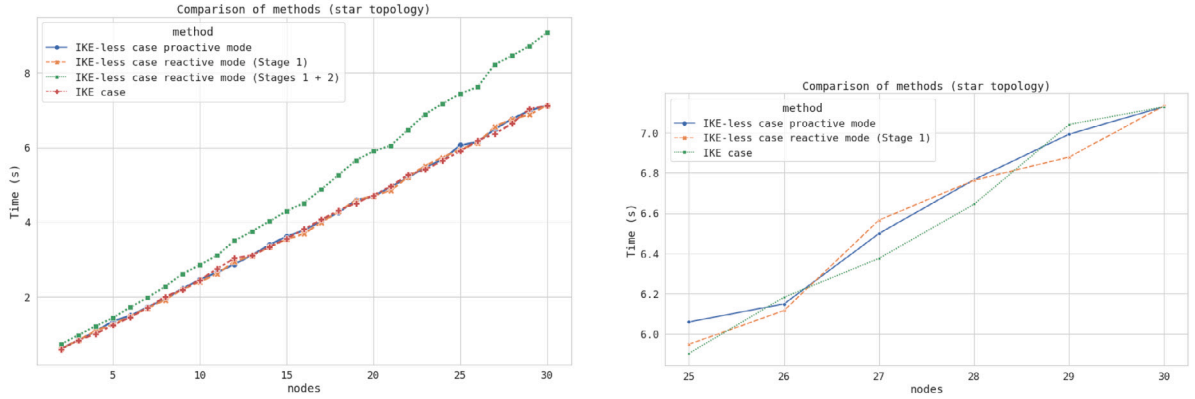


Fig. 14. Total configuration time for a star topology of n nodes using the IKE case.

Fig. 15(a). As observed in the star scenario, the IKE-less reactive case provides the worst performance owing to the unsuitable condition for a reactive approach, since all the nodes request the establishment of the IPsec SAs with the central node at the same time. In relation to the other approaches, note that they all provide very similar results for each n. Fig. 15(b) shows a Fig. 15(a) enlargement, in order to appreciate the slight differences between the scenarios, with only some time changes owing to variations between experiments.

4.4. Testbed scalability

One of the main objectives of our experiments was to analyze the tendency as regards the controller's response, its behavior, when an



(a) Configuration times of IKE, IKE-less proactive, IKE-less reactive (stage 1) and IKE-less (stages 1 + 2) cases

(b) Enlargement from 25 to 30 nodes

Fig. 15. Configuration times of the different approaches for the star topology.

increasing number of nodes must be configured at the same time, which is considered as a worst case for the scenario. Specifically, as expected, in a scenario with n nodes, there is a tendency of $O(n^2)$ in the mesh topology (as depicted in Fig. 8(b)), and $O(n)$ in the star topology (Fig. 12(b)).

Moreover, while in real scenarios the number of nodes may be higher, this does not mean that all of them must be configured at the same time. In order to saturate the controller, our experiments have, therefore, been carried out with cases in which all nodes request the establishment of IPsec SAs at the same time. In this respect, we consider that the number of nodes in the experiments is sufficiently large to show this tendency.

Nevertheless, scalability issues may appear in real scenarios, such as data centers, in which the number of nodes increase further. Several techniques with which to alleviate this problem are currently employed, such as using multiple controllers as the basis on which to decrease the time consumption as the overall number of connections are distributed among them. However, controllers must be coordinated in order to maintain the same context and behavior for all nodes. High availability and load balancing techniques can be used to accomplish this, such as memory sharing, load balancing with proxy servers or failover strategies [16].

5. Related work

In the last decade, numerous research works have evaluated the application of SDN technology in order to develop advanced security services such as security policy enforcement [17] or packet inspection [18]. In fact, the urgency is such that the industry has started to develop commercial and proprietary solutions for VPNs [19], clusters [20–22] or service mesh networks [23–25].

The standardization bodies have reacted to this need and there are several initiatives addressing this problem, with the IETF being one of the most active organizations. One of the earliest works [26] proposed the configuration of VPNs using NETCONF/YANG, although the authors provided only a high-level example and did not model configuration parameters. The authors of [27] defined a preliminary YANG model with which to remotely configure IKE/IPsec nodes, although the proposed interface was not conceived to operate using the SDN paradigm. In other words, this proposal was not able to achieve an automated management of IPsec SAs from a centralized controller since, for example, it did not allow nodes to inform of relevant events such as IPsec SA expiration.

In [28], the authors propose the usage of SDN controllers as trusted entities with which to distribute Diffie–Hellman (DH) public keys, thus allowing nodes to establish IPsec SAs without requiring the execution of the IKEv2 protocol. However, these authors focus only on the technical details of their solution and do not provide YANG models with which to implement it. This solution has been integrated into Ethernet VPNs (EVPN) [29] so as to design a mechanism with which to simplify the establishment of IPsec SAs between EVPN nodes. However, this proposal does not define the way that SPD and SAD are populated with the information required to identify, for example, the type of traffic protected or the cryptographic algorithms used.

Despite the fact that none of these works become a standard specification, the IETF community has persisted in its efforts to develop SDN-inspired solutions that will enable the agile management of secure communications. In fact, the I2SNF WG has recently released a standard specification describing a YANG data model for IPsec management [4]. As mentioned earlier, the authors are the main contributors of the standard and this paper has analyzed and validated both cases, the IKE case and the IKE-less case, proposed in the document for the central management of IPsec SAs. Furthermore, this line of work within the IETF is reaching other well-known network security protocols such as SSH [30] or TLS [31].

In parallel to standardization efforts, academic research has also evaluated the application of SDN technology to achieve a flexible and dynamic management of network security. Despite the fact that is possible to find some other works focused on the TLS protocol [15, 32,33], most of the contributions in this field consider the protection of communications at the network layer using IPsec. The proposals in this area can be classified into three groups, depending on the entity responsible for executing the IKE protocol: IKE executed on the control plane by the controller, IKE executed on the data plane by the nodes, or IKE not used.

Regarding the first group, the authors of [34] propose a method with which to integrate IPsec into SDN networks using OpenFlow as a southbound protocol. Despite demonstrating the applicability of their solution in a realistic use case, this work neither details the configuration parameters nor validates the proposal. The same objective is addressed in [35] but, in this case, for the establishment of IPsec tunnels between gateways. However, the authors do not provide any details about the configuration parameters and simply evaluate their proposal during the establishment of a single IPsec tunnel. The same problems appear in [36], whose authors focus on the usage of OpenFlow to configure IPsec VPNs in a SD-WAN scenario. Unlike the aforementioned contributions, which propose extensions to OpenFlow, the work presented in [37] configures IPsec tunnels using a new non-standardized southbound protocol, a design decision that obstructs its possible adoption. Moreover, the authors neither detail how IPsec policies are configured nor provide support to essential IPsec management tasks such as the expiration of SAs, and perform a simple validation of their proposal to establish a VPN between two routers.

With respect to the second group (IKE in the controller), there are two relevant contributions. On the one hand, Protego [38] proposes a solution for site-to-site VPN protection which takes advantage of SDN technology. Its authors specifically conceive IPsec gateways that exclusively perform encryption/decryption of IPsec traffic while the controller handles IKE traffic and distributes keys to the IPsec gateways. However, communication between control and data planes is not based on a standard southbound protocol and does not define the set of configuration parameters that are necessary to transfer the parameters negotiated by IKE to the IPsec gateways. The same approach is followed in [39] to propose a design that separates IKE function and IPsec processing. Despite the fact that this work specifies only some basic parameters with which to transfer the negotiated IPsec SA, these are insufficient to support the complete IPsec protocol operation (e.g. rekeying). Furthermore, this work lacks a clear definition of the different APIs used and makes use of a non-standardized JSON (JavaScript Object Notation) description for the configuration required by the IPsec peers. Another general problem of these type of contributions is the exportation of the security context negotiated by the controller (using IKE) to data plane nodes, since this operation is not supported by the IKE protocol itself.

Finally, with regard to the third group (IKE-less solutions), there are several studies that take advantage of directly providing data plane nodes with IPsec SA configuration parameters in order to avoid the execution of IKEv2. This means of managing IPsec SAs has been recognized to be an appropriate solution by which to manage secure communication channels among hosts in a data center [14]. Despite finding usages in non-SDN enabled scenarios like IoT networks [40], here we analyze these proposals for SDN networks.

For example, in [41] there is a key management solution in which the controller is responsible for distributing keys to data plane nodes that need to establish encrypted connections. However, this proposal formulates extensions to OpenFlow protocol in order to support a new key distribution protocol and opts for the use of the recently proposed WireGuard protocol rather than IPsec, which hampers the adoption of this solution.

Similarly, the authors of [42] propose a scheme called *Software Defined Security Associations*, which is based on the IKE-less case introduced in the work that led to the RFC 9061 standard. In this respect, the schema directly configures IPsec SAs between nodes communicating in a Service Function Chain (SFC). This work uses the obsolete YANG configuration model mentioned earlier [27], which is not able to support an automated management of IPsec SAs.

Finally, there is another interesting work in [43], which presents an implementation of our IKE-less operation mode for P4-enabled switches. The implementation is limited since it supports only the IKE-less proactive case for road-warrior and gateway to gateway scenarios, and the YANG data model is partially implemented (e.g. traffic selectors are only based on IP addresses).

Most of the problems detected in the academic works analyzed above are solved in our former work [3], together with its evolution as a standard framework with which to manage IPsec SAs [4]. The proposed solution follows a software defined strategy, supports an automated operation guided by a controller and uses the standard NETCONF protocol to implement the southbound interface. In addition, the proposed mechanism is able to deal with all the IPsec protocol operations (e.g. IPsec SA expiration) and allows the configuration of data plane nodes regardless of whether or not they support IKE.

In conclusion, to the best of our knowledge, the SDN-based IPsec management framework analyzed in this paper is a relevant solution, since it is an standard mechanism to be considered by future developments originating both industry and academia that require a dynamic and automated control of IPsec. Additionally, after analyzing existing works in this field, we have been unable to find any contributions that either describe the complete operation of the controller or perform an exhaustive performance evaluation.

6. Conclusions and future work

This work presents an experimental validation of an SDN-based framework for the management of IPsec configurations in networking scenarios. It is based on two main previous works by the authors: firstly, the IETF Proposed Standard RFC 9061 [4], in which the YANG data models for the interface between SDN controllers and IPsec nodes are described; and secondly, but of no less importance, the description of the proposed framework found in [3], in which cases, detailed operation and implementation description of the nodes, discussion and challenges are described.

In this work, we present a general description of the proposed framework required to introduce the workflow operations of the IKE and IKE-less (proactive and reactive) cases proposed. These operations lead the behavior of the controller, which has been developed in order to support the different cases. The state machines describing this behavior is an important result of this work, since it formalizes what is expected from the controller, including how to deal with error management in a SDN scenario.

The validation of this work is based on the deployment of two network scenarios, mesh and star network topologies, representing examples of network designs for current uses cases in which IPsec is being employed: SD-WAN, cloud providers, datacenters, etc. For each network topology, and making use of virtualization technologies such as Docker and Kubernetes, we have run the different cases proposed in the framework: IKE case and IKE-less case in proactive and reactive modes. The experiments are based on increasing the number of nodes requiring to connect to the mesh network, for the first topology, or to connect with the central node, for the star topology. The objective of these experiments has been to measure the time that the controller takes to perform the task of deploying the whole network, that is, to configure the nodes with the IPsec parameters required.

The experiments have shown the feasibility of the proposed framework, which is able to deploy a mesh or star network topology of around 30 nodes in approximately 16 and 7 s, respectively, for the IKE and IKE-less proactive case in our testbed. They have also shown the increasing time for the IKE-less reactive case, in which the configuration process is split into two stages. The results obtained from the experiments are also coherent with the complexity order for each topology ($O(n^2)$ and $O(n)$ respectively). A detailed discussion of the feasibility of the experiments can be found in Section 4.

However, this work is not complete, and the analysis and validation of another important component of the IPsec management is missing: the renewal of the IPsec SA's keys (rekey) for the IKE and IKE-less cases. Rekey implies creating new IPsec SAs with which to replace the old ones and is a cornerstone of IPsec and, therefore, of any SDN-based solution for the management of security protocols. It is for good reason that the RFC 9061 already describes this part. However, different alternatives may be evaluated in order to analyze the best rekey algorithm implemented in the controller so as to avoid, for example, packet loss or to reduce the rekey times. This part has been deliberately omitted by authors owing to the complexity of the rekey process in order to focus mainly on the initial configuration stages. The work concerning rekey will be published as a future work.

Another interesting future research work is to analyze other key provisioning alternative, in order to avoid the situation of the controller having to deal with the generation of the IKE and IPsec symmetric key material. The idea is to study potential key distribution schemes that will allow the controller to distribute symmetric keys to nodes without knowing the real value of the keys. Possible alternatives may be based on, for example, Diffie–Hellman and may require updates to existing YANG data models, new workflows, security requirements, etc.

Finally, the extension of the proposed SDN framework in order to support additional security protocols, such as TLS or SSH, is another interesting research line. For example, the new TLS 1.3 provides security properties, such as 0-RTT, that can be exploited to establish TLS channels using the SDN framework proposed in this paper.

CRediT authorship contribution statement

Gabriel López-Millán: Conceptualization, Methodology, Software, Writing – review & editing. **Rafael Marín-López:** Conceptualization, Methodology, Software, Writing – review & editing. **Fernando Pereñíguez-García:** Conceptualization, Methodology, Writing – review & editing. **Oscar Canovas:** Software, Resources, Validation, Writing – review & editing. **José Antonio Parra Espín:** Software, Validation, Writing – review & editing.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.csi.2022.103665.

Acknowledgments

This work has been partly funded by University of Murcia's project 33713-"Gestión automática de canales de comunicación seguros mediante el paradigma de redes definidas por software".

References

[1] L. Helali, M.N. Omri, A survey of data center consolidation in cloud computing systems, Comp. Sci. Rev. 39 (2021) 100366, http://dx.doi.org/10.1016/j.cosrev.2021.100366, URL https://www.sciencedirect.com/science/article/pii/S157401372100006X.

[2] O. Michel, E. Keller, SDN In wide-area networks: A survey, in: 2017 Fourth International Conference on Software Defined Systems, SDS, 2017, pp. 37–42, http://dx.doi.org/10.1109/SDS.2017.7939138.

[3] G. Lopez-Millan, R. Marin-Lopez, F. Pereniguez-Garcia, Towards a standard SDN-based IPsec management framework, Comput. Stand. Interfaces 66 (2019) 103357, http://dx.doi.org/10.1016/j.csi.2019.103357.

[4] R. Marin-Lopez, G. Lopez-Millan, F. Pereniguez-Garcia, A YANG Data Model for IPsec Flow Protection Based on Software-Defined Networking, SDN, RFC 9061, RFC Editor, 2021, http://dx.doi.org/10.17487/RFC9061, URL https://rfc-editor.org/rfc/rfc9061.txt.

[5] Open Network Foundation, SDN Architecture 1.1, TR-521, 2016.

[6] M. Caria, A. Jukan, M. Hoffmann, SDN partitioning: a centralized control plane for distributed routing protocols, IEEE Trans. Netw. Serv. Manag. 13 (3) (2016) 381–393, http://dx.doi.org/10.1109/TNSM.2016.2585759.

[7] Open Networking Foundation, OpenFlow switch specification version 1.5.1, 2015.

[8] R. Enns, M. Björklund, A. Bierman, J. Schönwälder, Network Configuration Protocol (NETCONF), RFC 6241, RFC Editor, 2011, http://dx.doi.org/10.17487/RFC6241, URL https://rfc-editor.org/rfc/rfc6241.txt.

[9] M. Fedor, M.L. Schoffstall, J.R. Davin, D.J.D. Case, Simple Network Management Protocol (SNMP), RFC 1157, RFC Editor, 1990, http://dx.doi.org/10.17487/RFC1157, URL https://rfc-editor.org/rfc/rfc1157.txt.

[10] K. Seo, S. Kent, Security Architecture for the Internet Protocol, RFC 4301, RFC Editor, 2005, http://dx.doi.org/10.17487/RFC4301, URL https://rfc-editor.org/rfc/rfc4301.txt.

[11] M. Björklund, YANG - A Data Modeling Language for the Network Configuration Protocol, NETCONF, RFC 6020, RFC Editor, 2010, http://dx.doi.org/10.17487/RFC6020, URL https://rfc-editor.org/rfc/rfc6020.txt.

[12] M. Bjorklund, The YANG 1.1 Data Modeling Language, RFC 7950, RFC Editor, 2016.

[13] YANG Catalog, 2021, https://yangcatalog.org/. (Accessed 15 July 2021).

[14] Y. Nir, A Data Center Profile for Software Defined Networking (SDN)-based IPsec, Internet Engineering Task Force, 2019, draft-nir-i2nsf-ipsec-dc-prof-00, in preparation, URL https://datatracker.ietf.org/doc/html/draft-nir-i2nsf-ipsec-dc-prof-00.

[15] M. Vajaranta, J. Kannisto, J. Harju, Implementation experiences and design challenges for resilient SDN based secure WAN overlays, in: Proceedings - 11th Asia Joint Conference on Information Security, AsiaJCIS 2016, 2016, pp. 17–23, http://dx.doi.org/10.1109/AsiaJCIS.2016.25.

[16] D. Suh, S. Jang, S. Han, S. Pack, M.-S. Kim, T. Kim, C.-G. Lim, Toward highly available and scalable software defined networks for service providers, IEEE Commun. Mag. 55 (4) (2017) 100–107, http://dx.doi.org/10.1109/MCOM.2017.1600170.

[17] S.K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, J.C. Mogul, Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags, in: 11th USENIX Symposium on Networked Systems Design and Implementation, NSDI 14, USENIX Association, Seattle, WA, 2014, pp. 543–546.

[18] A. Bremler-Barr, Y. Harchol, D. Hay, Y. Koral, Deep packet inspection as a service, in: Proceedings of the 2014 Conference on Emerging Networking Experiments and Technologies, CoNEXT 2014, 2014, pp. 271–282, http://dx.doi.org/10.1145/2674005.2674984.

[19] Network services orchestrator VPN solution overview, 2021, https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/solution-overview-c22-734917.html. (Accessed 15 July 2021).

[20] RedHat OpenShift. Encrypting Traffic between nodes with IPsec, 2021, https://docs.openshift.com/container-platform/3.11/admin_guide/ipsec.html. (Accessed 15 July 2021).

[21] IBM CLoud Private. Encrypting cluster data network traffic with IPsec, 2021, https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.0/installing/ipsec_mesh.html. (Accessed 15 July 2021).

[22] Amazon Web Services Cloud. Deploying an opportunistic IPsec mesh on the AWS Cloud, 2021, https://aws.amazon.com/about-aws/whats-new/2019/05/new-quick-start-deploys-opportunistic-ipsec-mesh-on-aws/?nc1=h_ls. (Accessed 15 July 2021).

[23] AWS App Mesh. Transport Layer security (TLS), 2021, https://aws.amazon.com/about-aws/whats-new/2019/05/new-quick-start-deploys-opportunistic-ipsec-mesh-on-aws/?nc1=h_ls. (Accessed 15 July 2021).

[24] RedHat OpenShift Service mesh, 2021, https://docs.openshift.com/container-platform/4.6/service_mesh/v2x/ossm-security.html. (Accessed 15 July 2021).

[25] Google anthos service mesh, 2021, https://cloud.google.com/anthos/service-mesh5. (Accessed 15 July 2021).

[26] P.A. Shafer, An Architecture for Network Management Using NETCONF and YANG, in: Request for Comments, RFC 6244, RFC Editor, 2011, http://dx.doi.org/10.17487/RFC6244, URL https://rfc-editor.org/rfc/rfc6244.txt.

[27] K.N. Tran, H. Wang, V.K. Nagaraj, X. Chen, Yang Data Model for Internet Protocol Security, IPsec, draft-tran-ipsecme-yang-01, Internet Engineering Task Force, 2016, in preparation, URL https://datatracker.ietf.org/doc/html/draft-tran-ipsecme-yang-01.

[28] D. Carrel, B. Weis, IPsec Key Exchange using a Controller, draft-carrel-ipsecme-controller-ike-01, Internet Engineering Task Force, 2019, in preparation, URL https://datatracker.ietf.org/doc/html/draft-carrel-ipsecme-controller-ike-01.

[29] A. Sajassi, A. Banerjee, S. Thoria, D. Carrel, B. Weis, J. Drake, Secure EVPN, draft-sajassi-bess-secure-evpn-05, Internet Engineering Task Force, 2021, in preparation, URL https://datatracker.ietf.org/doc/html/draft-sajassi-bess-secure-evpn-05.

[30] K. Watsen, YANG Groupings for TLS Clients and TLS Servers, draft-ietf-netconf-tls-client-server-28, Internet Engineering Task Force, 2022, in preparation, URL https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tls-client-server-28.

[31] K. Watsen, YANG Groupings for SSH Clients and SSH Servers, draft-ietf-netconf-ssh-client-server-28, Internet Engineering Task Force, 2022, in preparation, URL https://datatracker.ietf.org/doc/html/draft-ietf-netconf-ssh-client-server-28.

[32] A. Ranjbar, M. Komu, P. Salmela, T. Aura, An SDN-based approach to enhance the end-to-end security: SSL/TLS case study, in: 2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016, 2016, pp. 281–288, http://dx.doi.org/10.1109/NOMS.2016.7502823.

[33] E. Sousa, V.A. Cunha, M.B. de Carvalho, D. Corujo, J.P. Barraca, D. Gomes, A.E. Schaeffer-Filho, C.R.P. dos Santos, L.Z. Granville, R.L. Aguiar, Orchestrating an SFC-enabled SSL/TLS traffic processing architecture using MANO, in: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN, 2018, pp. 1–7, http://dx.doi.org/10.1109/NFV-SDN.2018.8725675.

[34] V. Heydari Fami Tafreshi, H. Cruickshank, Z. Sun, Integrating IPsec within OpenFlow architecture for secure group communication, 12 (2014) 41, http://dx.doi.org/10.3939/j.issn.1673-5188.2014.02.007.

[35] A. Coly, M. Mbaye, S-SDS: A Framework for Security Deployment as Service in Software Defined Networks, in: Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, vol. 296, 2019, pp. 92–103, http://dx.doi.org/10.1007/978-3-030-34863-2_9.

[36] Y. Li, J. Mao, SDN-Based access authentication and automatic configuration for IPSec, in: Proceedings of 2015 4th International Conference on Computer Science and Network Technology, ICCSNT 2015, 2016, pp. 996–999, http://dx.doi.org/10.1109/ICCSNT.2015.7490904.

[37] W. Li, F. Lin, G. Sun, SDIG: Toward software-defined IPsec gateway, in: Proceedings - International Conference on Network Protocols, ICNP, Vol. 2016-December, 2016, http://dx.doi.org/10.1109/ICNP.2016.7785316.

[38] J. Son, Y. Xiong, K. Tan, P. Wang, Z. Gan, S. Moon, Protego: cloud-scale multitenant IPsec gateway, in: 2017 USENIX Annual Technical Conference, USENIX ATC 17, USENIX Association, Santa Clara, CA, 2017, pp. 473–485, URL https://www.usenix.org/conference/atc17/technical-sessions/presentation/son.

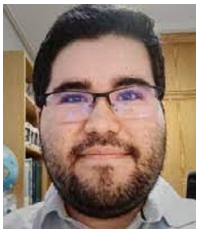
[39] M. Vajaranta, J. Kannisto, J. Harju, IPsec and IKE as Functions in SDN Controlled Network, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 10394 LNCS, 2017, pp. 521–530, http://dx.doi.org/10.1007/978-3-319-64701-2_39.
[40] S. Aragon, M. Tiloca, M. Maass, M. Hollick, S. Raza, ACE Of spades in the IoT security game: a flexible IPsec security profile for access control, in: 2018 IEEE Conference on Communications and Network Security, CNS 2018, 2018, http://dx.doi.org/10.1109/CNS.2018.8433209.
[41] A.S. Braadland, Key Management for Data Plane Encryption in SDN Using WireGuard (Master's thesis), Norwegian University of Science and Technology (NTU), 2017.
[42] H. Gunleifsen, T. Kemmerich, V. Gkioulos, A proof-of-concept demonstration of isolated and encrypted service function chains, Future Internet 11 (9) (2019) http://dx.doi.org/10.3390/fi11090183.
[43] F. Hauser, M. Haberle, M. Schmidt, M. Menth, P4-IPsec: site-to-site and host-to-site VPN with IPsec in P4-based SDN, IEEE Access 8 (2020) 139567–139586, http://dx.doi.org/10.1109/ACCESS.2020.3012738.

Gabriel López-Millán is a full-time assistant professor in the Department of Information and Communications Engineering of the University of Murcia. He has collaborated actively in IETF WG such as ABFAB and I2NSF. He is co-author of RFCs 9061 and 7499. His research interests include network security, SDN/NFV, PKI, identity management, authentication and authorization. He received his Ph.D. in computer science from the University of Murcia.

Rafael Marin-Lopez is a full-time assistant lecturer in the Department Information and Communications Engineering at the University of Murcia (Spain). Additionally, he is collaborating actively in IETF, especially in I2NSF and ACE Working Groups. He is co-author of RFC 9061 and other standards. His main research interests include network access authentication, key distribution and security in different type of networks such as SDN, IoT and mobile networks.

Fernando Pereniguez-Garcia received an M.Sc. in Computer Engineering in 2007, and his Ph.D. in Computer Science in 2011, both from University of Murcia. Currently, he is Assistant Professor in the Department of Science and Informatics at University Center of Defense - Spanish Air Force Academy. His research interests include security, privacy and mobility in wireless networks.

óscar Cánovas was born in Barcelona (Spain), in 1975. He received his Ph.D. degree in Computer Science from the University of Murcia, Spain, in 2003. From 1999 to 2009 he was an Assistant Professor with the University of Murcia. Since 2009 he has been an Associate Professor at the same university. His research interests include information security, access control, payment systems, ubiquitous computing and indoor positioning.



José Antonio Parra received an M.Sc. in Computer Engineering in 2021 from University of Murcia. Currently, he is researcher in the Department of Information and Communications Engineering of the University of Murcia. His research interests include network communications and security.