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]

[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*

Abstract

The Internet Engineering Task Force (IETF), international standardization organism for Internet, has recently approved a standard, RFC 9061, which defines an interface and framework to manage autonomously IPsec SAs 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 to create IPsec SAs. Two cases are presented: *IKE-case*, where the nodes ship an IKE implementation that is configured by the controller or *IKE-less* where the controller sends directly the IPsec SAs 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 to compare these cases, which are missing parts in the standard.

Keywords: IPSec, IKE, Management, SDN, performance

1. Introduction

Networks are becoming more and more complex. For example, cloud-based datacenters are composed by hundreds or thousands of virtual devices needing to protect communications to form a mesh of secure connections [1]. Software-defined Wide Area Networks (SD-WAN) [2] are another challenging scenario requiring enterprises to securely and intelligently direct traffic involving communication gateways across the WAN, forming a star of secure connections. These are only two of many examples that have posed new challenges for the network administration, especially considering the security requirements derived from these scenarios.

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

The SDN paradigm [5] brings new opportunities to implement complex and secure networks by giving the responsibility of taking decisions to a centralized entity named SDN controller, which operates in a control

*Corresponding author
Email addresses: gabilm@um.es (Gabriel López-Millán), rafa@um.es (Rafael Marín-López), fernando.pereniguez@cud.upct.es (Fernando Pereñíguez-García), ocanovas@um.es (Oscar Canovas)

plane to exchange configuration information with network nodes interoperating in a separated data plane. The high level network security requirements are defined by the network administrator at the application plane, where general security policies are defined (e.g., protect data traffic between nodes A and B). These general policies are translated by the SDN Controller (from now on *the controller*) into IPsec specific configurations that are sent to the nodes implementing the functionality in the data plane.

In our proposal, there are two different operation cases [4]. In the *IKE case* the node implements IKEv2 to manage the IPsec SAs and the controller just provides the necessary configuration to IKEv2 to create the IPsec SAs. In the *IKE-less case*, the node is simplified by only shipping the IPsec logic, delegating the whole key management operations to the SDN controller. Both cases are suitable for different application scenarios, as we will explain later.

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

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

2. Background

2.1. SDN and IPsec

As previously described, the SDN paradigm decouples a network service in two clearly differentiated functions: the process of decision making (*control plane*) and the data operations (*data plane*). For example, traditionally, each router for the networking L3 routing service is traditionally configured to make both actions: to take the decision about the output interface where an input IP packet has to be forwarded and, once this decision has been taken by the routing protocol (control plane), to move the input packet to the output interface in order to be sent by the router (data plane). When using the SDN paradigm for the routing service [6], the decision making process is moved to the SDN controller in the control plane, which has the complete view of the network while the router becomes a L2 switching service just moving packets from input to output interfaces, according to the decision taken by the controller. Openflow [7] is one of the main standard protocols in these cases to send switching information from the controller to the nodes, i.e. used as *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 the one responsible for key management (key agreement, peer's authentication, cipher suite agreement, etc.). IPsec can be used to protect IP traffic between two any 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 if 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 instructs to protect the packet, then the system has to apply the security services (encryption, integrity, etc.) mandated by the policy. 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 happens 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

on the system are ready. Beside, 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). Additionally, IKEv2 defines a IKE Security Association (IKE SA) used to perform the mutual peer's 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 perfectly fits in the SDN concept. An IPsec protected network is usually composed by one of these two types of nodes: those without an IKEv2 service, so 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 be also configured manually in the nodes though IPsec SAs are managed automatically by IKEv2. Next section describes how the SDN paradigm can be used to automatize the management of IPsec SAs regardless of whether the IKEv2 service is supported by the network node or not.

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 and the device to be configured plays the server role. Configuration and state data are exchanged over RPC (Remote Procedural Call) operations in the form of XML data following a YANG model [11].

NETCONF defines the usage of RPC operations over datastores. For example, when a NETCONF client wants to apply some configuration in the server, it sends the RPC $< edit-config >$ operation. This operation includes the configuration in YIN format, which is a XML representation of the configuration data compliant with a YANG language [12]. Making use of the $< edit-config >$ means the server directly applies the given configuration into the system and, if the configuration is successfully applied, the server returns an $< /ok >$ response to the client. NETCONF also provides *notifications*. Notifications are messages directly sent from the NETCONF server to the client after some determined event. The information contained in these notifications are also defined in the YANG model.

The configuration and state data exchanged between the NETCONF client and server is protected by a security transport layer, such as SSH (by default) or TLS.

2.3. The YANG data model for IPsec management

There is a myriad of YANG data models currently defined by the IETF [13]. In the context of this work, we make use of the YANG data model for IPsec 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 case described before. For the *IKE case*, the *ietf-i2nsf-ike* model defines the configuration for the IKEv2 service, the SPD and PAD at a given IPsec node shipping IKE. For 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 to the SDN paradigm; secondly, the operation in detail, by specifying the different exchanges between the entities in the architecture (the controller and the nodes) as well as the state machines that define the controller's operation.

3.1. Architecture

Figure 1 shows the architecture for the SDN-based IPsec management that we describe in RFC 9061. It is conceived 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 to establish IPsec SAs to a central point (control plane) while IP traffic flow protection (e.g. encryption, integrity and authentication operations) and forwarding logic are placed at the nodes (data plane). Communication between nodes in the data plane is done through the data network where all nodes are connected. The controller can be connected to this network, or to a dedicated management network. NETCONF is used as *southbound* protocol.

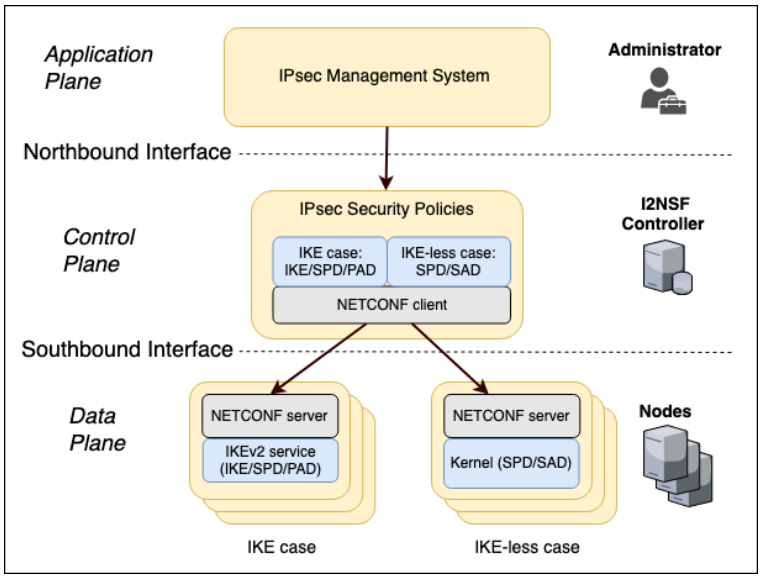


Figure 1: General architecture

As described before, 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*). Therefore, the controller sends any configuration that IKEv2 requires (keys to perform authentication, IPsec policies information, expected IPsec SA lifetimes, etc.) so that it can 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 at 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 includes traffic selectors, AH or ESP protection, tunnel or transport mode, IPsec SA cryptographic suites, etc. Only one IPsec policy element for the outbound traffic flow in installed (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 allows 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 simultaneously the IPsec policies and the IPsec SAs to the nodes, so 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, the controller sends 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 we can find in distinct uses cases (see section 4). The assumption for 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 information required to establish the IPsec SAs for a pair of nodes.

This section also describes the controller's operation in detail. Indeed, during the standardization of the interface between the controller and the nodes, one of the main discussions was the difficulty to express the controller operation in these cases, specially for the IKE-less case, taking into account potential errors or unexpected situations. In fact, with the two cases at hand, the controller varies in complexity. It is worth noting that due to its complexity, all matters related with IPsec SAs rekeying process are beyond the scope of this paper and, consequently, left as future work.

For the IKE case, the controller needs to send configuration information to the IKEv2 service application in the node, so IKEv2 can 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. On the other hand, 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 relaying in any part in the node. In order to analyze these differences, we have formalized the behaviour 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 it is useful to detail its behaviour. 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 some event e. During the transition and before reaching state j, action $a()$ is performed.

3.2.1. IKE-case

Figure 2 shows the different exchanges that a controller C performs to configure IKE for two nodes Ni and Nj. The controller sends a NETCONF $< edit-config >$ 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 with a $< rpc-reply >$ with $< /ok >$. Once one of the nodes is configured, and according to the IKE configuration sent from the controller, the node applies this information in the IKEv2 service application (e.g. Strongswan,

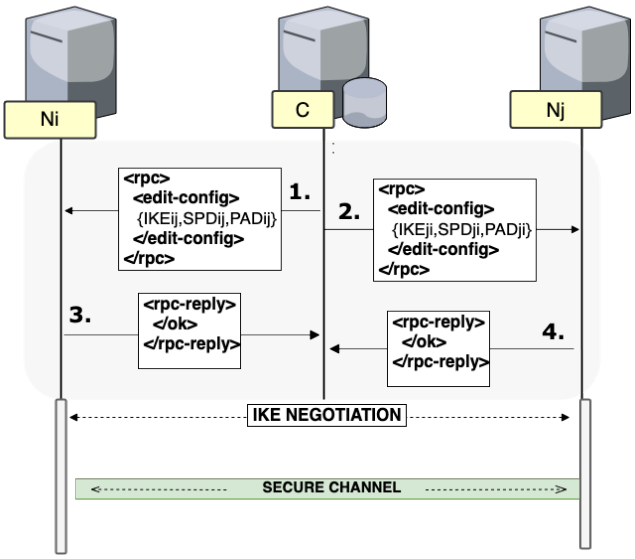


Figure 2: IKE case

Libreswan, etc.) and can automatically start the IKE negotiation with the other peer in order to generate the corresponding IPsec SAs.

Figure 3 describes the specific state machine that allows 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 $< edit-config >$ message with the XML files based on the YANG data model for the IKE case to configure Ni and Nj at the same time. If the controller receives an $< rpc-reply >$ with $< /ok >$ (event OK) the state machine moves to state $WAIT\_OK2$ where 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 to wait for the next OK coming from the other node (e.g. Nj). Thus, these steps and **transitions 1, 2 and 3** correspond to the operation described in Fig. 2, which is the situation where no errors happened during the process.

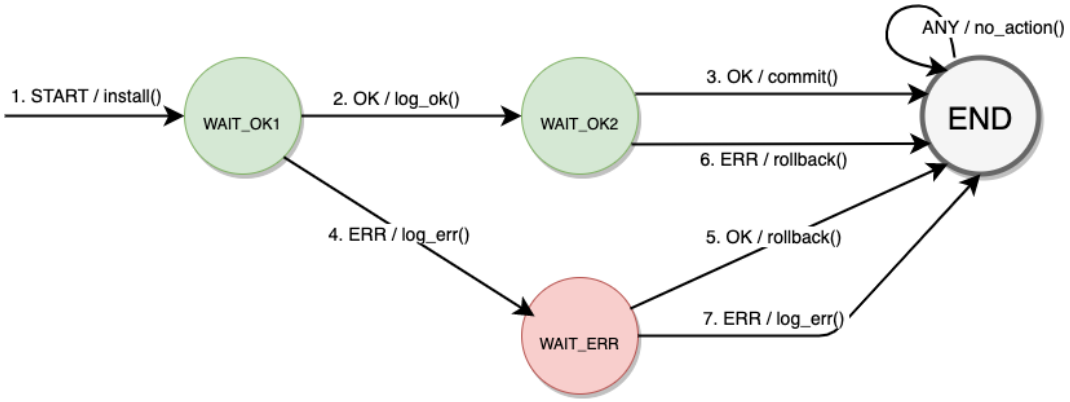


Figure 3: Controller's state machine for IKE case and IKE-less case proactive mode

However, some errors may happen during the configuration. For example, an $< rpc-reply >$ with $< /error >$ is sent by the node to the controller; a timeout without receiving any answer from the node or a node simply crashed (event ERR). **Transitions 4, 5 ,6 and 7** deal with these situations. More specifically, **transition 4** happens when the controller firstly detects that was not able to apply the configuration to

a node (e.g. Ni), registering this fact ($log\_err()$). Then, if the controller receives an OK from the other node (e.g. Nj) it means the configuration was applied there (**transition 5**). In this case, the controller must perform a $rollback()$ operation of the configuration sent to Nj (it is not useful anymore because the configuration in Ni failed). A similar situation may happen when the OK was received firstly (**transition 2**) by a node and then a posterior ERR happens 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 where two errors have happened 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. Figure 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*).

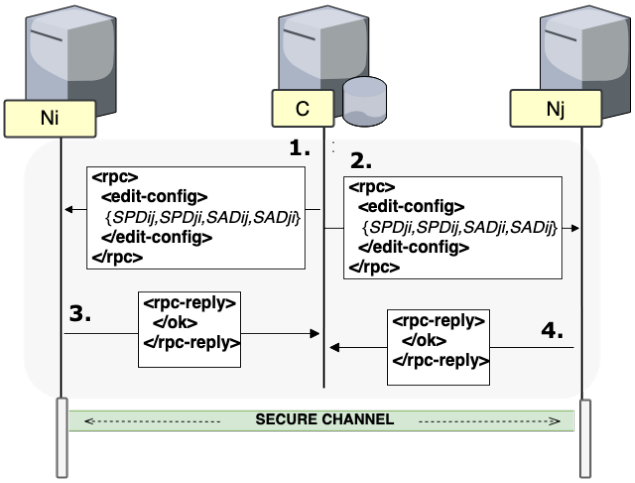


Figure 4: IKE-less case proactive mode

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 $< rpc - reply >$ with $< /ok >$ (*steps 3 and 4*).

In essence, the controller's behaviour and, therefore, the shape of the state machine do not differ from the one defined for the IKE case (see Fig. 3). The differences are in 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 hand, in case of ERR, $rollback()$ removes the IPsec policies from the SPD as well as the IPsec SAs installed in the SAD of the node that was successfully configured.

3.2.3. IKE-less case reactive mode

Figure 5 shows the workflow for this mode, where 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*), pointing out that an outgoing data flow needs to be protected and the IPsec SAs in the SAD need to be installed (*steps 6 to 9*). This situation may happen in scenarios where data flows are sporadic and there is no need to deploy IPsec SAs immediately, saving some resources in the nodes.

Figure 6 shows the state machine that defines the behaviour for the reactive mode, which is far more complex. The reason is related with the fact that the reactive mode is divided in two parts: first, configuration of the IPsec policies; second, 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**

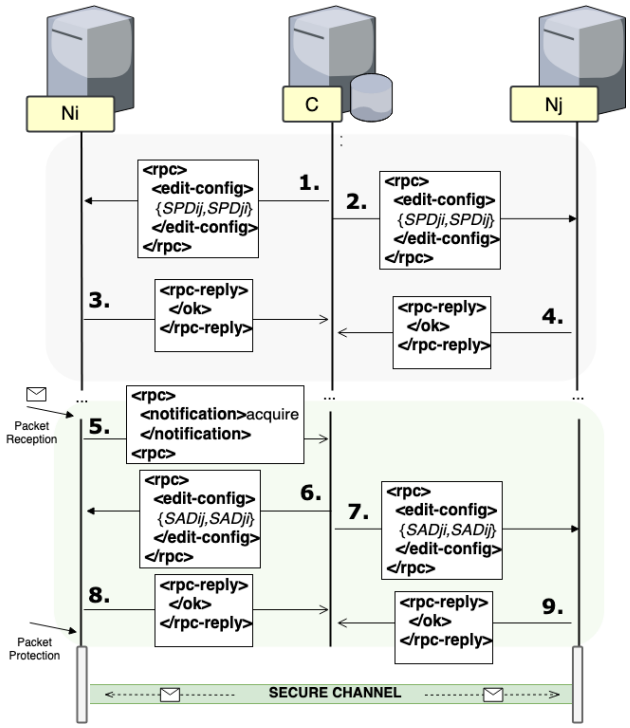


Figure 5: IKE-less case reactive mode

1 to 3 imply a correct installation of the IPsec policies in the nodes' SPD (*install_spd*()) in both nodes (*commit_spd*()).

After that, the controller will wait for the reception of an *ACQ* from any of two nodes. If an *ACQ* happens then **transition 4** starts the installation of the required IPsec SAs in both nodes (*install_sad*()). **Transitions 5 and 6** imply that IPsec SAs were correctly installed in both nodes' SAD and that controller can create some state (*commit_sad*()) about the installed IPsec SAs. 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 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 nodes' SAD right away (*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 happens 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 just reporting the error (*log_err*()) is enough 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 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. **Transitions 17, 18, 19 and 20** deal with them. Similarly to the previous cases, if the controller received

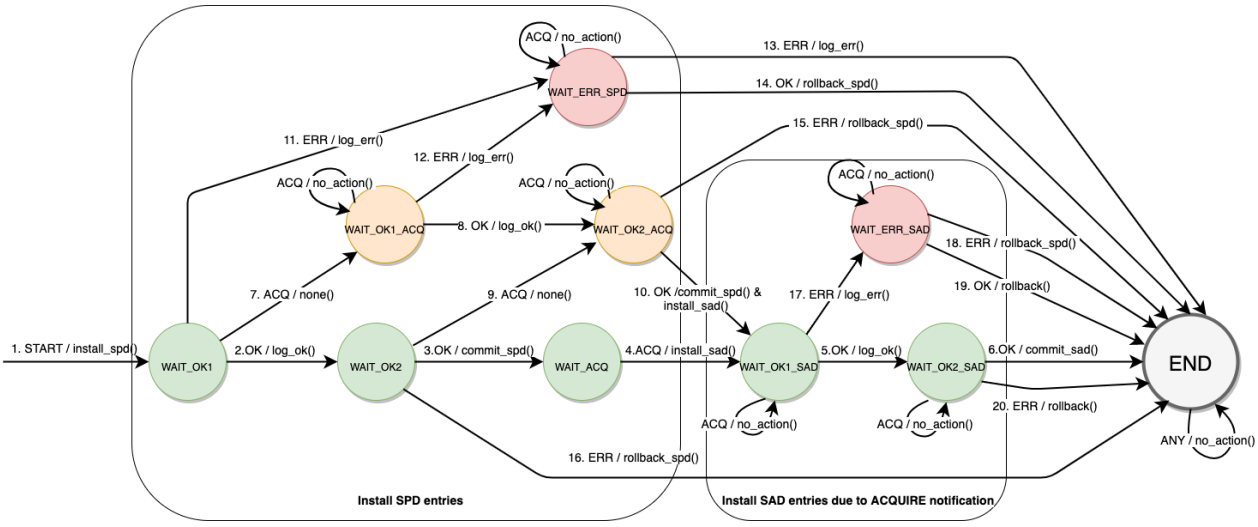


Figure 6: Controller's state machine for IKE-less case reactive mode.

two *ERR* events (transitions 17 and 18), it means no IPsec SA was successfully installed in the nodes and the controller only needs to remove the policies in 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 does not only remove 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 to extract performance results for the IKE and IKE-less cases in representative network scenarios.

4.1. Deployed testbed

Figure 7 shows the general testbed, consisting of SDN network composed of N nodes and a single controller, that we have developed to evaluate our framework in two representative scenarios: a mesh and a star topology of IPsec secure channels. The former is a typical one in datacenters [1], while the latter usuarlly 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 to distribute uniformly among the nodes of the cluster. For the sake of simplicity, there is only one network to communicate controllers and nodes (data and management networks are the same one). It is left for future works to analyze how the existence of different networks might affect the validation results.

The server where 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.

4.2. Implementation aspects

In relation to the different elements composing the testbed, we detail next the main implementation characteristics.

[1]https://www.kubernetes.io/

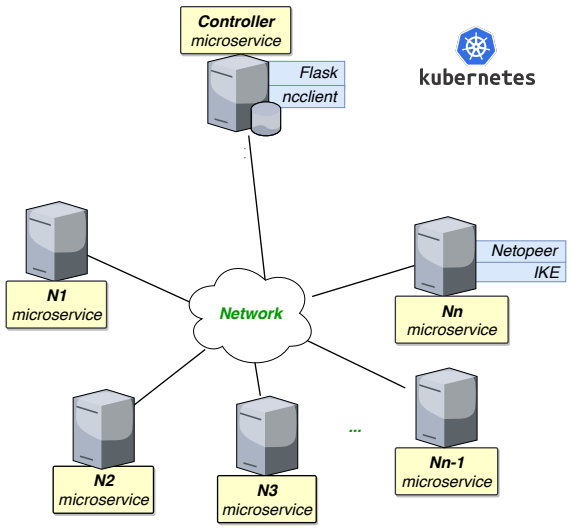


Figure 7: Overview of the virtual SDN scenario for testing purposes

4.2.1. Controller

It 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 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]. To automatize the experiments, we have also implemented a simple registration process for the nodes to notify 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

Every N*i* is a Ubuntu 18.04-server container implementing 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 we will describe 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 allows loading our proposed YANG data models and scheduling callbacks on the arrival of new configurations. For the IKE case, the IKE service application has been implemented using Strongswan v5.5[7], and experiments have been done making use of IKEv2 nodes authentication based on 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 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.

[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

For each task, we have tested different workloads (number of nodes to be configured). As we will see, 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*). In the following, we present and discuss all the results for each task.

4.3.1. Task 1: Mesh of n nodes

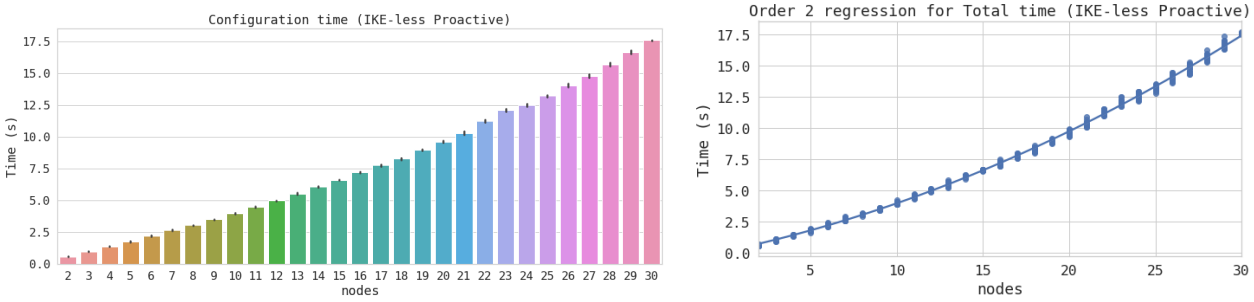
In this task, the controller is committed to configure a mesh of n nodes, which will be fully protected using IPsec.

The controller is able to build a mesh of n nodes using the basic operation, described in section 3, for establishing 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 information required to establish the IPsec connection between a pair of nodes. The framework would also allow including multiple configuration elements in each NETCONF message, however this would not be valid when the total number of nodes is unknown or changing. Moreover, this decision is useful because provides some homogeneity to the performance evaluations comparing different approaches, and also allows 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 number of nodes n. Particularly, we run 15 executions for each n from $n = 2$ to $n = 30$, to observe the behavior and scalability of the system as the number of nodes increases.

For the **IKE-less case proactive mode**, Figure 8a 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, 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). Accordingly, to form a complete mesh of n nodes, the controller must 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 it is shown in Figure 8b. 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 seconds.

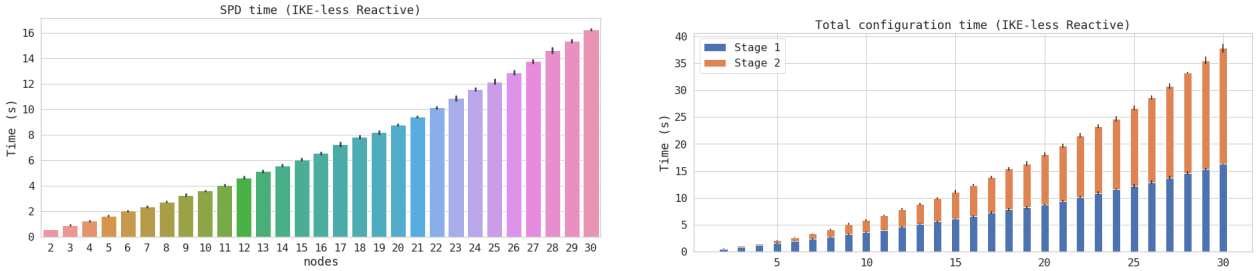


(a) Total configuration time

(b) Order 2 regression

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

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 that there will not be any IP packet requiring 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 rest of nodes at the same time. This involves a massive arrival of *acquire* notifications to the controller and the resulting configuration messages that we already depicted in Figure 5.



(a) Stage 1 configuration time (IPsec policies)

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

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

Figure 9a 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 seconds, a lower value than for 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.

Figure 9b 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 it 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 we shown in section 3, so it is a phase where the controller must process three NETCONF messages (*acquire, edit-config, ok*) instead of 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 only once the necessary IPsec SAs, as a result of the first received *acquire* notification, but it will have to process the second one anyway to discard it (see Figure 6 for the IKE-less case reactive mode state machine).

For the **IKE case**, the controller has to provide the configuration for IKE, SPD and PAD. Once this information is established between a pair of nodes, the IKEv2 negotiation takes place and the IPsec SAs are established eventually. As we did for the IKE-less reactive approach, the distribution of the configurations is not overlapped with the IKEv2 negotiation since there will not be any packet requiring protection yet. 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.

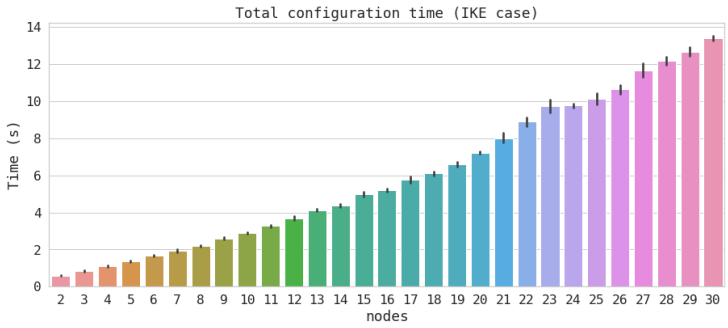


Figure 10: Total Configuration time for a mesh topology of n nodes using the IKE case

Figure 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 seconds, a slightly lower value than for the previous approaches since the information that IKEv2

needs to establish the IPsec SA is less than IKE-less case. In any case, the configuration time has a quadratic tendency, for the same aforementioned reason.

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

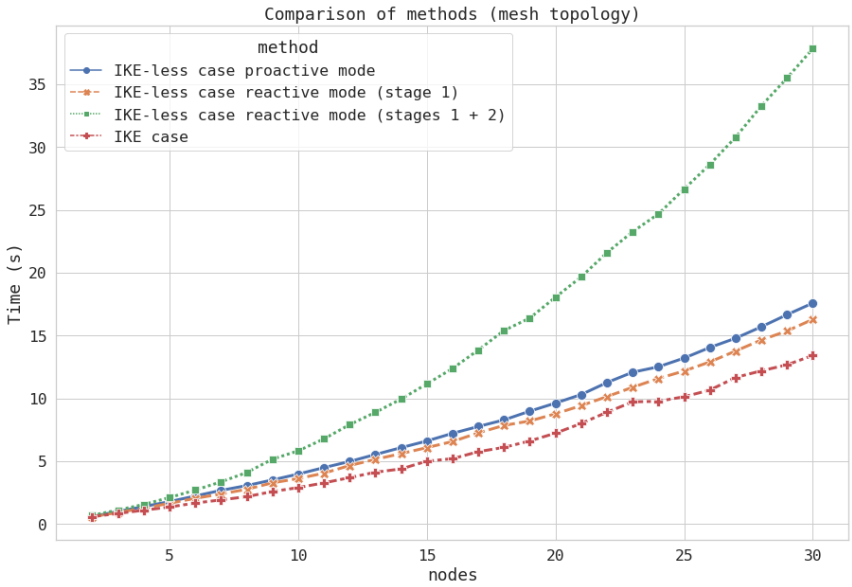


Figure 11: Configuration times of the different approaches for the mesh scenario

Once we have analyzed in a separate manner the results of each approach, we illustrate in Figure 11 a comparison of the performance of the different options. As we can see, the total configuration time for the IKE-less reactive mode is much higher, but we have to take into account that it shows the worst scenario, where all nodes in the mesh request at the same time the establishment of IPsec SAs. In those situations, where the administrator knows that the nodes will need to establish immediately a full mesh of IPsec SAs, it would be a better option to make use of the IKE-less proactive approach. In a typical scenario where the IPsec SAs are established sparsely and along a wider range of time, the reactive strategy is more suitable since the configuration time for the IPsec policies (stage 1) is lower compared to the proactive, and it would exhibit a better performance 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, we observe that the IKE case provides better results, though those times do not include the IKE negotiation time, they are only referring to the IKE configuration time. Despite of the fact that the IKE-less proactive cases exhibits a slightly lower performance than the IKE case, we have 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.

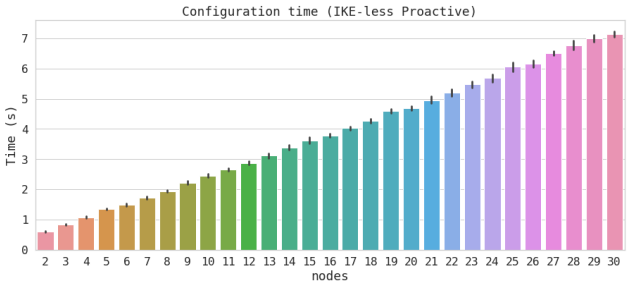
4.3.2. Task 2: Star topology of n nodes

We have also evaluated the performance of the proposed framework 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 some SD-WAN scenarios [15]. The controller configures each IPsec connection using two messages: one message for the central node and another one for the peer node.

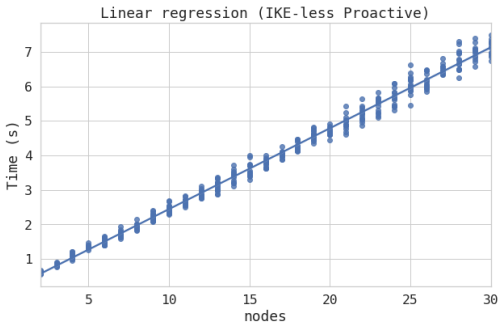
As we did before, for each network size we run 15 executions, from $n = 2$ to $n = 30$, to observe the behavior of the system as the topology size increases.

Figure 12a 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 seconds, 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 it is shown in Figure 12b. This linear relationship also explains the reduction in the required configuration time in relation to the mesh network, which presented an $O(n^2)$ complexity.



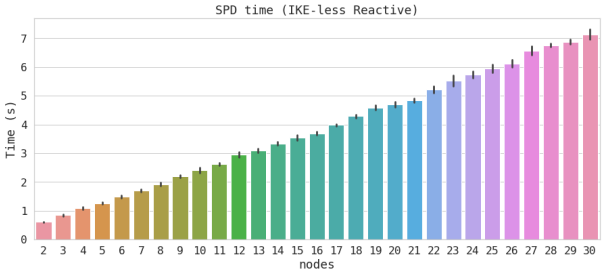
(a) Configuration time



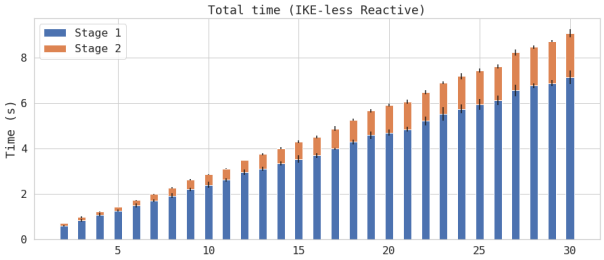
(b) Linear regression

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

For the **IKE-less case reactive mode**, we analyze the results in the same way we did 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.



(a) Stage 1 configuration time (IPsec policies)



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

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

Figure 13a 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 no traffic to be protected yet. For example, the controller can configure the IPsec policies of a network of 30 nodes in approximately 7 seconds (configuration of 29*2 = 58 IPsec policies), a similar value than the one obtained for the IKE-less case proactive mode. The configuration time has a linear tendency, for the same aforementioned reason. Figure 13b includes also 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 mesh topology, and therefore this time exhibits a better performance of the controller.

For the **IKE case** we will follow the same reasoning that we employed in the previous scenario. There are two different stages: IKE configuration and then IKE negotiation. As we did before, the distribution of the IKE configuration including the IPsec policies for the nodes' SPD is the task to be performed by the

controller, which is not overlapped with the IKE negotiation since there will not be any packet requiring protection yet. For the second stage, we have prepared the experiment in such a way all the nodes need to exchange IP traffic that need 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.

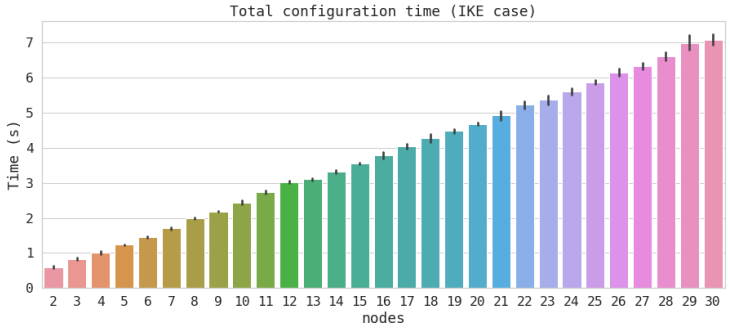


Figure 14: Total configuration time for a star topology of n nodes using the IKE case

Figure 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 seconds, as other methods exhibited.

Once we have analyzed in a separate manner the results of each approach, we illustrate in Figure 15 a comparison of the performance of the different options. As we observed in the mesh scenario, the IKE-less reactive case provides the worst performance due to the unsuitable condition for a reactive approach, since all the nodes request at the same time the establishment of the IPsec SAs with the central node. In relation to the other approaches, we observe they all provide very similar results for each n.

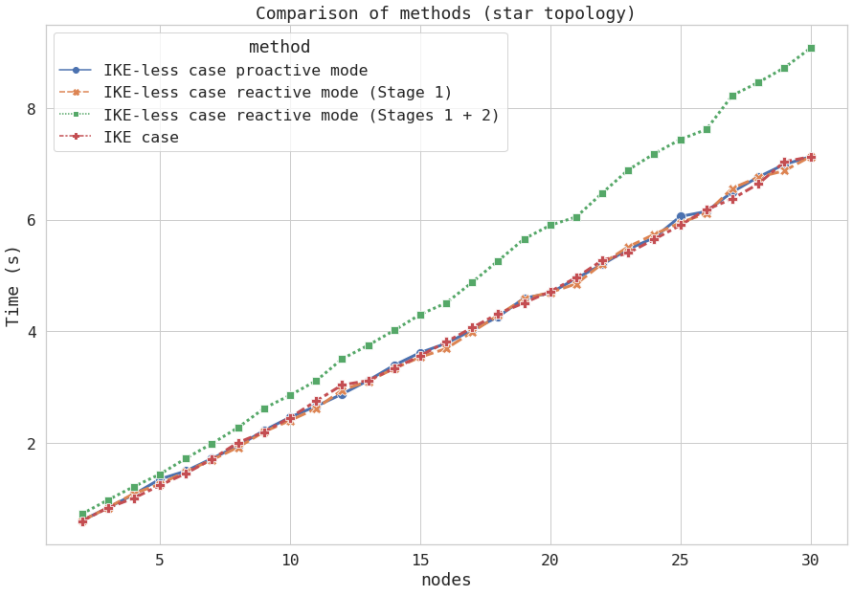


Figure 15: Configuration times of the different approaches for the star topology

5. Related work

In the last decade, numerous research works have evaluated the application of SDN technology to develop advanced security services like security policy enforcement [16] or packet inspection [17]. In fact, the urgency

is such that the industry has started to develop commercial and proprietary solutions for VPNs [18], clusters [19, 20, 21] or service mesh networks [22, 23, 24].

The standardization bodies have reacted to this need and we can find several initiatives addressing this problem, being the IETF one of the most active organizations. One of the earliest works [25] proposed the configuration of VPNs using NETCONF/YANG, although they only provided a high-level example and do not model configuration parameters. Authors in [26] define a preliminary YANG model to remotely configure IKE/IPsec nodes, however the proposed interface was not conceived to operate using the SDN paradigm. In other words, this proposal is not able to achieve an automated management of IPsec SAs from a centralized controller since, for example, it does not allow nodes to inform of relevant events such as IPsec SA expiration.

In [27], authors propose the usage of SDN controllers as trusted entities for distributing Diffie-Hellman (DH) public keys, thus allowing nodes to establish IPsec SAs without requiring the execution of IKEv2 protocol. However, authors only focus on the technical details of their solution and do not provide YANG models to implement it. This solution has been integrated in Ethernet VPNs (EVPN) [28] to design a mechanism to simplify the establishment of IPsec SAs between EVPN nodes. However, this proposal leaves undefined the way SPD and SAD are populated with information necessary to identify, for example, the type of traffic protected or the cryptographic algorithms used.

Despite all these works did not become a standard specification, the IETF community has persisted in its efforts to develop SDN-inspired solutions enabling 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, IKE case and 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 [29] or TLS [30].

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 we can find some other works focused on the TLS protocol [31, 15, 32], most of the contributions in this field consider the protection of communications at network layer using IPsec. Depending on the entity responsible for executing the IKE protocol, the proposals in this area can be classified into three groups: IKE executed in the control plane by the controller, IKE executed in the data plane by the nodes, or IKE is not used.

Regarding the first group, authors in [33] propose a method for integrating IPsec in SDN networks using OpenFlow as southbound protocol. Despite they demonstrate 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 [34] but, in this case, for the establishment of IPsec tunnels between gateways. However, 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 [35], 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 [36] configures IPsec tunnels using a new non-standardized southbound protocol, a design decision obstructing its possible adoption. Moreover, authors neither detail how IPsec policies are configured nor provide support to essential IPsec management tasks like the expiration of SAs, and perform a simple validation of their proposal to establish a VPN between two routers.

With respect the second group (IKE in the controller), we find two relevant contributions. On one hand, Protego [37] proposes a solution for site-to-site VPN protection which takes advantage of SDN technology. Specifically, they conceive IPsec gateways exclusively performing 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 to the IPsec gateways the parameters negotiated by IKE. The same approach is followed in [38] to propose a design that separates IKE function and IPsec processing. Despite this work only specifies some basic parameters to transfer the negotiated IPsec SA, these are insufficient to support the complete IPsec protocol operation (e.g. rekeying). Furthermore, this work lacks of 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 relies on the exportation of the security context negotiated by the controller (using IKE) to data plane nodes, since it is an operation not supported by the IKE protocol itself.

Finally, related to the third group (IKE-less solutions), we find several studies taking advantage of directly providing data plane nodes with IPsec SA configuration parameters to avoid the execution of IKEv2. This way of managing IPsec SAs has been recognized to be an appropriate solution to manage secure communication channels among hosts in a data center[14]. Despite we find usages in non-SDN enabled scenarios like IoT networks [39], here we analyze those proposals for SDN networks.

For example, in [40] we find a key management solution where the controller is responsible for distributing keys to data plane nodes needing to establish encrypted connections. However, this proposal formulates extensions to OpenFlow protocol in order to support a new key distribution protocol and opts by using the recently proposed WireGuard protocol instead of IPsec, which hamper the adoption of this solution.

Similarly, authors in [41] propose a scheme called *Software Defined Security Associations*, which is based on the IKE-less case introduced in the work that lead to the standard RFC 9061. In this sense, 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 [26], which is not able to support an automated management of IPsec SAs.

Finally, we find another interesting work in [42] where it is presented an implementation of our IKE-less operation mode for P4-enabled switches. The implementation is limited since only supports 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 previously analyzed are solved in our former work [3], together with its evolution as standard framework 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 affords the configuration of data plane nodes regardless of whether they support IKE or not.

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, coming from both industry and academia, needing a dynamic and automated control of IPsec. Additionally, after analyzing existing works in this field, we have not found contributions neither describing the complete operation of the controller nor performing an exhaustive performance evaluation.

6. Conclusions and future work

This work presents an experimental validation of a SDN-based framework for the management of IPsec configurations in networking scenarios. It is based on two main previous works from the authors: firstly, the IETF Proposed Standard RFC 9061 [4], where the YANG data models for the interface between SDN controllers and IPsec nodes are described; and secondly, but no less important, the description of the proposed framework found in [3], where 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 behaviour of the controller, which has been developed in order to support the different cases. The state machines describing this behaviour is an important result of this work since it formalizes what it 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 where IPsec is being used: 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 to deploy the whole network, that is, to configure the nodes with the required IPsec parameters.

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 seconds, 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, where the configuration process is split in 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 about 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 for the IPsec management is missing: the IPsec SA's keys renewal (rekey) for the IKE and IKE-less cases. Rekey implies to create new IPsec SAs to replace the old ones and it is a cornerstone in IPsec and, therefore, in any SDN-based solution for the management of security protocols. Not in vain, the RFC 9061 already describes this part. However, different alternatives may be evaluated to analyze the best rekey algorithm implemented in the controller in order to avoid, for example, packet lost or to reduce the rekey times. Due to the complexity of the rekey process, this part has been omitted deliberately by authors to focus mainly on the initial configuration stages. The work about rekey will be published as a future work.

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

Finally, the extension of the proposed SDN framework 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.

Declaration of competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Authors contribution

Gabriel López-Millán: Conceptualization, Methodology, Software, Writing - Review & Editing. Rafael Marín-López: Conceptualization, Methodology, Software, Writing - Review & Editing. Fernando Pereñiguez-García: Conceptualization, Methodology, Writing - Review & Editing . Óscar Cánovas: Software, Resources, Validation, Writing - Review & Editing.

References

[1] L. Helali, M. N. Omri, A survey of data center consolidation in cloud computing systems, Computer Science Review 39 (2021) 100366. doi:https://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. doi:10.1109/SDS.2017.7939138.

[3] G. Lopez-Millan, R. Marin-Lopez, F. Pereniguez-Garcia, Towards a standard SDN-based IPsec management framework, Computer Standards & Interfaces 66 (2019) 103357. doi:https://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 (Jul. 2021). doi:10.17487/RFC9061.
URL `https://rfc-editor.org/rfc/rfc9061.txt`

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

[6] M. Caria, A. Jukan, M. Hoffmann, Sdn partitioning: A centralized control plane for distributed routing protocols, IEEE Transactions on Network and Service Management 13 (3) (2016) 381–393. doi:10.1109/TNSM.2016.2585759.

[7] Open Networking Foundation, OpenFlow Switch Specification Version 1.5.1 (March 2015).

[8] R. Enns, M. Björklund, A. Bierman, J. Schönwälder, Network Configuration Protocol (NETCONF), RFC 6241 (Jun. 2011). doi: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 (May 1990). doi:10.17487/RFC1157.
URL https://rfc-editor.org/rfc/rfc1157.txt

[10] K. Seo, S. Kent, Security Architecture for the Internet Protocol, RFC 4301 (Dec. 2005). doi: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 (Oct. 2010). doi: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 (August 2016).

[13] YANG Catalog., https://yangcatalog.org/ (Accessed 2021-07-15).

[14] Y. Nir, A Data Center Profile for Software Defined Networking (SDN)-based IPsec, Internet-Draft draft-nir-i2nsf-ipsec-dc-prof-00, Internet Engineering Task Force, work in Progress (Jul. 2019).
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, 2016, pp. 17–23. doi:10.1109/AsiaJCIS.2016.25.

[16] 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.

[17] A. Bremler-Barr, Y. Harchol, D. Hay, Y. Koral, Deep Packet Inspection as a service, 2014, pp. 271–282. doi:10.1145/2674005.2674984.

[18] Network Services Orchestrator VPN Solution Overview., https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/solution-overview-c22-734917.html (Accessed 2021-07-15).

[19] RedHat OpenShift. Encrypting traffic between nodes with IPsec, https://docs.openshift.com/container-platform/3.11/admin_guide/ipsec.html (Accessed 2021-07-15).

[20] IBM Cloud Private. Encrypting cluster data network traffic with IPsec, https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.0/installing/ipsec_mesh.html (Accessed 2021-07-15).

[21] Amazon Web Services Cloud. Deploying an opportunistic IPsec mesh on the AWS Cloud., https://aws.amazon.com/about-aws/whats-new/2019/05/new-quick-start-deploys-opportunistic-ipsec-mesh-on-aws/?nc1=h_ls (Accessed 2021-07-15).

[22] AWS App Mesh. Transport layer Security (TLS)., https://aws.amazon.com/about-aws/whats-new/2019/05/new-quick-start-deploys-opportunistic-ipsec-mesh-on-aws/?nc1=h_ls (Accessed 2021-07-15).

[23] RedHat OpenShift Service Mesh, https://docs.openshift.com/container-platform/4.6/service_mesh/v2x/ossm-security.html (Accessed 2021-07-15).

[24] Google Anthos Service Mesh., https://cloud.google.com/anthos/service-mesh5 (Accessed 2021-07-15).

[25] P. A. Shafer, An Architecture for Network Management Using NETCONF and YANG, RFC 6244 (Jun. 2011). doi:10.17487/RFC6244.
URL https://rfc-editor.org/rfc/rfc6244.txt

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

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

[28] A. Sajassi, A. Banerjee, S. Thoria, D. Carrel, B. Weis, J. Drake, Secure EVPN, Internet-Draft draft-sajassi-bess-secure-evpn-04, Internet Engineering Task Force, work in Progress (Feb. 2021).
URL https://datatracker.ietf.org/doc/html/draft-sajassi-bess-secure-evpn-04

[29] K. Watsen, YANG Groupings for TLS Clients and TLS Servers, Internet-Draft draft-ietf-netconf-tls-client-server-24, Internet Engineering Task Force, work in Progress (May 2021).
URL https://datatracker.ietf.org/doc/html/draft-ietf-netconf-tls-client-server-24

[30] K. Watsen, YANG Groupings for SSH Clients and SSH Servers, Internet-Draft draft-ietf-netconf-ssh-client-server-24, Internet Engineering Task Force, work in Progress (May 2021).
URL https://datatracker.ietf.org/doc/html/draft-ietf-netconf-ssh-client-server-24

[31] A. Ranjbar, M. Komu, P. Salmela, T. Aura, An SDN-Based Approach to Enhance the End-to-End security: SSL/TLS Case Study, in: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, 2016, pp. 281–288. doi:10.1109/NOMS.2016.7502823.

[32] 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. doi:10.1109/NFV-SDN.2018.8725675.

[33] V. Heydari Fami Tafreshi, H. Cruickshank, Z. Sun, Integrating IPsec Within OpenFlow Architecture for Secure Group Communication 12 (2014) 41. doi:10.3939/j.issn.1673-5188.2014.02.007.

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

[35] Y. Li, J. Mao, SDN-based Access Authentication and Automatic Configuration for IPSec, 2016, pp. 996–999. doi:10.1109/ICCSNT.2015.7490904.

[36] W. Li, F. Lin, G. Sun, SDIG: Toward Software-Defined IPsec Gateway, Vol. 2016-December, 2016. doi:10.1109/ICNP.2016.7785316.

[37] 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

[38] M. Vajaranta, J. Kannisto, J. Harju, IPsec and IKE as Functions in SDN Controlled Network, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10394 LNCS (2017) 521–530. doi:10.1007/978-3-319-64701-2_39.

[39] 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, 2018. doi:10.1109/CNS.2018.8433209.

[40] A. S. Braadland, Key Management for Data Plane Encryption in SDN Using WireGuard, Master's thesis, Norwegian University of Science and Technology (NTU) (jun 2017).

[41] H. Gunleifsen, T. Kemmerich, V. Gkioulos, A Proof-of-Concept Demonstration of Isolated and Encrypted Service Function Chains, Future Internet 11 (9) (2019). doi:10.3390/fi11090183.

[42] 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. doi:10.1109/ACCESS.2020.3012738.