Computer Standards & Interfaces 66 (2019) 103357

Contents lists available at ScienceDirect

Computer Standards & Interfaces

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

Towards a standard SDN-based IPsec management framework

Gabriel Lopez-Millan*,[a], Rafael Marin-Lopez[a], Fernando Pereniguez-Garcia[b]

[a] Dept. Information and Communications Engineering (DIIC), University of Murcia, 30100, Spain
[b] Department of Sciences and Informatics, University Defense Center - Spanish Air Force Academy, 30720, Spain

ARTICLE INFO

Keywords:
SDN
IPsec
IKEv2
Management
Security
NETCONF
YANG

ABSTRACT

The *Software-defined Network* (SDN) paradigm enables an efficient management of future networks by decoupling the *control plane* from the *data plane*. Specifically, *network resources* (e.g. switches or routers) only perform Internet Protocol (IP) packet forwarding in the data plane based on rules dictated by *SDN controllers* that implement the control plane.

Despite the applicability of SDNs to manage network security is a hot topic, managing security associations using SDNs to protect data plane communications is not well covered in the literature.

In this sense, the IP Security (IPsec) protocol is the standard to protect IP traffic at network level and it is foreseen a key element in the forthcoming 5G networks or Software-Defined WAN (SD-WAN). Traditionally, the IPsec operation is assisted by a key management protocol, such as the *Internet Key Exchange* (IKEv2), responsible for establishing IPsec Security Associations (IPsec SAs). Yet, manual configuration of IKEv2 is still required, which does not scale when the number of IPsec entities is high.

In this paper we propose a solution to manage IPsec SAs using SDNs avoiding manual configuration in the network resources and enabling a reduced involvement of network administrators. We present two different cases, *IKE case* and *IKE-less case*, balancing between the participation of the SDN controller in the IPsec management and the complexity of the network resource. We provide a comprehensive explanation and deep analysis of the solution, which is undergoing standardization at the *Internet Engineering Task Force* (IETF), describing the interfaces, the operation and security aspects. Finally, we include a simple but significative performance analysis of both cases and a proof-of-concept implementation of the proposal.

1. Introduction

The *Software-Defined Network* (SDN) technology [1] will be a basic component in future 5G networks [2], virtualization environments [3] and the so-called Software-Defined Wide Area Networks (SD-WANs) [4]. SDNs allow network administrators the autonomous management of different *network resources* without dealing with the configuration details. This is possible by separating and centralizing, in a entity called *SDN controller*, the component of the network resources that takes decisions over the network traffic (*control plane*) from the component that merely forwards traffic to the destination (*data plane*). This separation enables the network behavior to be programmed with high level languages, thus flexibilizing the network administrator's tasks, and the simplification of network resources.

The successful application of softwarized technology to current networks will depend on the ability of SDNs to provide adequate levels of security. We live in a connected world where the use of public networks like Internet introduces risks jeopardizing the security of communications. To fight against them, traditionally networks include security functions, each one specialized in mitigating a specific type of attack: *Firewalls* to track and control communications, *Intrusion Prevention/Detection Systems* (IPS/IDS) to detect intrusion activities and perform responsive actions, *Authentication, Authorization and Accountability* (AAA) servers to enforce security policies, etc.

This paper focuses on a specific security function: the establishment of security associations to protect communications from unauthorized access (confidentiality) or modification (integrity). In general, this is called *flow-based* security function. Since this aspect has not been well covered in the literature, we propose a solution for SDNs to deal with the management of security associations, an essential process to guarantee security of communications. Although, there exist multiple choices to protect communications at different layers (i.e. Transport Layer Security (TLS) at transport layer, Secure Shell (SSH) protocol at application layer), this paper focuses on IP Security (IPsec) [5] at network layer.

Indeed, IPsec has been widely adopted to protect communications

* Corresponding author.
E-mail addresses: gabilm@um.es (G. Lopez-Millan), rafa@um.es (R. Marin-Lopez), fernando.pereniguez@cud.upct.es (F. Pereniguez-Garcia).

https://doi.org/10.1016/j.csi.2019.103357
Received 22 September 2018; Received in revised form 17 May 2019; Accepted 6 June 2019
Available online 07 June 2019

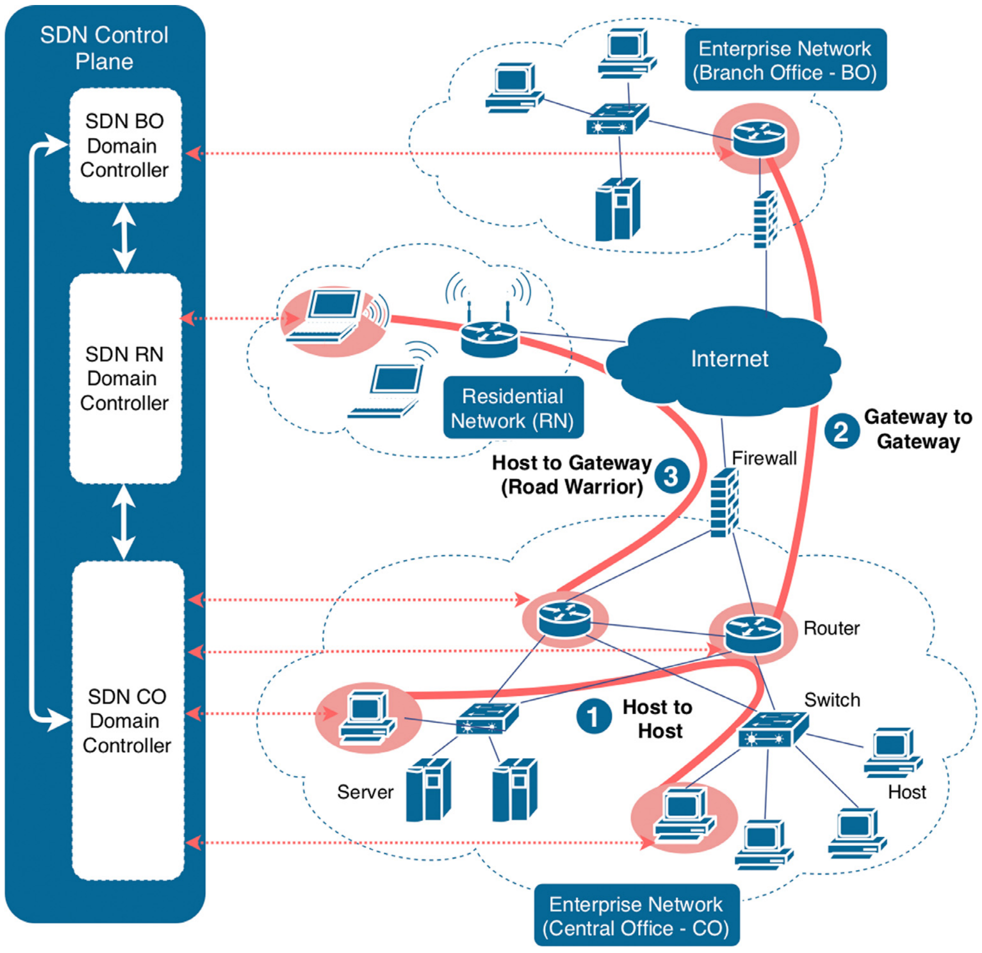


Fig. 1. SDN-based IPsec management.

between entities at IP layer. As observed in Fig. 1, it can be used to protect communications between *(1)* specific endpoints (*host to host*), *(2)* the interconnection of individual networks (*gateway to gateway*) or *(3)* the communication of a particular endpoint with a network (*host to gateway*, also called *road-warrior*).

IPsec requires the establishment of IPsec Security Associations (IPsec SAs) to operate. Roughly speaking, an IPsec SA represents a shared state between two network resources and contains the key material and the cryptographic algorithms to protect the IP packets. Since the manual establishment of IPsec SAs does not scale properly and it is prone to security flaws (e.g. administrators sometimes reuse the same cryptographic keys to configure different IPsec SAs), a key management protocol is strongly recommended to dynamically establish fresh IPsec SAs. IPsec uses *Internet Key Exchange* (IKEv2) [7] for this purpose by default, though the IPsec architecture is open to select other mechanisms. Yet, the network administrator must still provide the IKEv2 configuration manually, which is unmanageable in networks with a high number of entities.

This article proposes a framework, based on solid standards, which avoids the manual configuration for the autonomous management of IPsec SAs following the SDN paradigm (see Fig 1). The framework initially focuses on the *host to host* and *gateway to gateway* scenarios, thus leaving the *road-warrior* usage as future work.

Our solution distinguishes two different operation cases. The first one, the *IKE case*, the complexity of IPsec management is shared between the network resource and SDN controller. The network resource implements IKEv2 to manage the IPsec SA and the component that protects IP packets (*IPsec logic*), while the SDN controller just provides the necessary configuration for IKEv2 to operate. The second one, the *IKE-less case*, the network resource is simplified by only shipping the IPsec logic, delegating the whole key management operations to the SDN controller. Due to the importance of this contribution, the *Internet Engineering Task Force* (IETF) has adopted [8] this idea as a first step for standardization.

This contribution aims to present the design of the SDN-based IPsec management framework, its components and the required interfaces among them, to analyze its operation and to validate the proposed design through a proof-of-concept implementation. Therefore, the main goal of this article is to show the reader the framework, which has led the beginning of a standardization process, in a more comprehensive way. Performance and scalability evaluation of the proposal through an exhaustive analysis is left as future work. Nevertheless, we present a simple model to compare the performance of the two operation cases as starting point for future analysis.

The rest of this article is distributed as follows. Section 2 describes relevant background, Section 3 describes potential usage scenarios of this framework, and Section 4 briefly presents the related work found in the literature. Section 5 details the design of the proposed SDN-based IPsec management framework while Section 6 discusses relevant challenges. Section 7 identifies the security issues arisen in the proposed solution, and Section 8 provides a simple but representative performance analysis of the proposed solution. The correct operation of the proposal is demonstrated through a proof-of-concept testbed in Section 9. Finally, Section 10 outlines some conclusions and future research.

2. Background

2.1. IP Security architecture

IPsec protocol [5] provides confidentiality, authentication and integrity to data flows at IP layer between two network resources (peers), by means of a security architecture that defines security protocols to protect the content of the IP packet. There are two security protocols: the *Authentication Header* (AH) to provide data authentication/integrity, currently deprecated; and the *Encapsulation Security Payload* (ESP) that, additionally, provides data encryption.

The provision of these security services relies on the use of cryptographic algorithms and key material (i.e. an IPsec SA) that must be agreed by the involved peers. Furthermore, protection can be provided to the upper layer protocol data included in the packet's payload (*transport mode*), or to the whole IP packet by encapsulating it within a new outer IP packet (*tunnel mode*).

An inbound/outbound IP packet has to be protected when the packet's fields match a security policy that describes the characteristics of the flow (*traffic selectors*), such as source/destination IP address, port number, etc. Each peer locally stores these policies in a structure known as *Security Policy Database* (SPD), which is located inside the operating system's kernel. If a flow matches a policy entry, the peer needs an IPsec SA to protect/process the flow packets.

An IPsec SA represents a logical state between two peers, describing the kind of protection, the cryptographic algorithms and the key material to protect the flow. Each IPsec SA is uniquely identified by a *Security Parameter Index* (SPI), an *IP Destination Address* and the *Security Protocol Identifier* (ESP). Since each IPsec SA is unidirectional, the protection of a (bidirectional) communication channel between two peers requires the establishment of two IPsec SAs at each peer, one for inbound and other for outbound packets. Each IPsec SA has associated two lifetimes: *soft*, in order to indicate when the IPsec SA is near to expire (e.g. time or bytes processed by the network resource); and *hard*, to indicate when the IPsec SA will expire and, consequently, removed from the system. IPsec SAs are also locally stored, inside the kernel, by each peer in the so-called *Security Association Database* (SAD) and can be either manually or automatically configured by means of a key management mechanism. IKEv2 [7] is the default protocol used for the automated management.

Before negotiating the IPsec SAs, IKEv2 requires a mutual peer authentication that can be based, for example, on digital signatures or shared secrets, and this information is stored in another IPsec database known as *Peer Authorization Database* (PAD). The security is strengthened thanks to the establishment of an IKE Security Association (IKE SA) used by IKEv2 itself to perform the mutual peer's authentication and, then, to negotiate the IPsec SAs (known as CHILD SAs in IKEv2 terminology) used to protect data flows.

Table 1 shows some important parameters which need to be configured in SPD, SAD, PAD entries and IKEv2.

Table 1
Configuration parameters for IKEv2 protocol and SPD/SAD/PAD databases.

SPD	SAD
Remote IP Address	Security Parameter Index (SPI)
Local IP Address	Sequence Number Counter
Next Layer Protocol	Sequence Counter Overflow
Descriptive Name	Anti-Replay Window
Local Port	ESP Crypto Information
Remote Port	Lifetime of SA (soft and hard)
	Protocol Mode
	Path *Maximum Transmission Unit* (MTU)
PAD	**IKEv2**
Peers Identity	Peers Identity
Peers Authentication Methods	Ciphersuite
Authentication Algorithm	IKE SA and CHILD (IPsec) SAs (soft and hard) lifetimes
Authentication Keys	NAT Traversal Negotiation

2.2. Software Defined Networks

The Software Defined Network (SDN) paradigm separates the forwarding decisions (*data plane*) from the network control logic (*control plane*). Network resources implement the data plane just to manage flows based on the decisions made by centralized entities that implement the control plane, called *SDN controllers*.

The network administrator interacts with the SDN controller by means of application services running at the *application plane*. Using a high level description language, application services allow administrators to define policies about routing, filtering or security. These policies are sent to the SDN controller through a *northbound* interface.

Then, they are translated by the SDN controller into low-level and resource-aware configurations that are sent to the network resource through the *southbound interface*. This interface is usually based on standard Application Programming Interfaces (APIs). For large network scenarios where several SDN controllers may be deployed, a controller-to-controller interface (*east/west interface*) is also required.

At the time of writing, neither northbound nor east/west interfaces are standardized in the industry. However, there are a myriad of well defined protocols for the southbound interface such as OpenFlow [19] or NETCONF [23].

2.3. NETCONF and YANG

NETCONF [23] is a network management IETF protocol in charge of the remote configuration of network resources. It follows a traditional client-server model, where the NETCONF client is the management entity and the NETCONF server functionality is located in the device to be managed. Configuration and state data are exchanged in form of XML data over RPC (Remote Procedure Call) operations.

The main operations defined by NETCONF are: *get-config*, to retrieve configuration data from a specific datastore; *get*, to retrieve both configuration and state data; *edit-config*, to apply configuration data to a specific datastore; *copy-config*, to copy configuration data from one datastore to other; *delete-config*, to delete a configuration datastore; and *lock/unlock* to lock or release datastores. Finally, the NETCONF client can subscribe (*create-subscription*) to receive NETCONF server asynchronous *notifications* [24], sent from the server when some relevant event or change takes place in the managed device. With the exception of the notification messages, every NETCONF RPC soliciting a concrete operation must be acknowledged (RPC *ok*) by the server.

NETCONF is being widely used currently together with YANG [26]. YANG models are used to represent, in a human-readable format, configuration and state data. The configuration data, compliant with the model, is then translated to a XML representation (YIN), which is used by NETCONF in the RPC operations. NETCONF clients and servers are able to validate YIN data to ensure it is compliant with the YANG model being used. Currently, there are standardized models for basic system management, network topologies, routing, NAT management, etc.[1]

NETCONF architecture is based on the secure exchange of configuration parameters and state data. To this end, NETCONF protocol requires the usage of a secure transport protocol that guarantees the protection of the communication channel between the NETCONF client and server. Typical security protocols used are TLS and SSH.

One of the main properties of NETCONF is the use of *datastores*. Datastores are conceptual places to store and access configuration data. NETCONF defines three datastores: *running-config*, representing the current configuration data configured and running into the device; *startup-config*, representing the configuration to be applied when the

[1] http://www.netconfcentral.org.

device sets up; and *candidate-config*, to work with configurations in the device before to apply then.

3. Usage scenarios

The application of the proposed IPsec management framework becomes relevant in new emerging networking scenarios that can benefit from the use of SDN-based solutions. One of these scenarios is *cloud virtualization environments*, where cloud providers have to maintain up and running a huge number of Virtual Machines (VMs). IPsec can be used to provide flow protection between VMs and to establish secure overlay networks. An example is VPNaaS (Virtual Private Network as a Service) ,[2] where the network administrator is able to provide IPsec requirements to secure virtualized networks.

Another important usage scenario is *Software-Defined Wide Area Networks* (SD-WAN) [43], where SDN has revealed as a competitive alternative to legacy WAN technologies to facilitate the administration of remote business sites/branches via software [4]. In fact, several vendor specific SD-WAN solutions [44] are attracting much attention from companies suffering the cost associated to the management of their remote sites. SD-WAN technology is not exempt from satisfying the need of protecting communications between remote branches and corporate data center devices, specially when public networks (i.e. Internet) are used. This protection can be assured with IPsec, as it has been traditionally done in WAN deployments. In this scenario, the usage of an automated IPsec management mechanism overcomes the security weaknesses derived from a traditional manual configuration across thousands of branch networks that most of the times consists in re-using the same key material across all sites.

The European Telecommunications Standards Institute (ETSI) has defined a Network Function Virtualization (NFV) framework abstracting network functions from the hardware on which they runs. NFV replaces network services provided by dedicated hardware with virtualized software that runs in commodity hardware. This means that network services, such as routers, firewalls or load balancers, can be replaced with software running on virtual machines. Service providers have shown interest in combining NFV with SDN technology to achieve a flexible environment [35]. However, the successful orchestration of the different NFVs depends on the definition of a secure operation environment that requires, among others, secure communication between Virtual Network Functions (VNFs) [36]. Given this need, the management of SDN-enabled IPsec communications is a key topic under the ETSI perspective.

Finally, the next generation of cellular networks (5G) constitutes another potential usage scenario. SDNs have been recognized as an essential technological enabler in 5G to achieve a flexible programmability of network functions [2]. At the same time, several security challenges arise in 5G networks, mainly derived from the need of protecting communications among the different architectural components [47]. For example, the *3rd Generation Partnership Project* (3GPP) has defined a security architecture for the IP Multimedia subsystem (IMS) proposing the use of IPsec together with IKEv2 for key management [37]. The combination of IPsec and SDN makes 5G networks a prospective candidate that may benefit of this automated IPsec management solution and their standarization.

4. Related work

The idea of a centralized security management framework promoted in this paper is a trend present in standardization bodies, specially in the IETF standards organization. In this way, the IETF *Interface to Network Security Functions* (I2NSF) [30] is working towards the definition of standard interfaces to manage *Network Security Functions* (NSFs) such as firewalls, intrusion detection/protection systems, etc. The envisioned architecture is aligned with the SDN paradigm since it considers the existence of a central *security controller* that receives high-level requirements (from network administrators) that are translated into specific commands sent to NSFs. In fact, as described before, the YANG model used in this work is being discussed in this working group.

The I2NSF working group activity is just an example of the numerous initiatives that have emerged within the IETF aimed at implementing a centralized management of IPsec security associations. For example, authors in [25] consider the configuration of a Virtual Private Network (VPN) as an example of application based on NETCONF/YANG. However, they only provide a high-level example and do not really model the configuration parameters needed to setup an IPsec-based VPN connection.

Another interesting work can be found in [9], where authors proposed a YANG model to configure IKE/IPsec protocols. Unfortunately, this work did not evolve beyond a preliminary design and, unlike this proposal, the model only considers the use of IKE. [14] proposes the use of SDN controllers as trusted entities for distributing Diffie-Hellman (DH) public keys to network resources. However, they focus on the SAD parameters and leave undefined the way SPD is properly configured. This solution is also used in [12] making use of BGP signalling.

[13] presents an analysis of the problem to securely interconnect branch offices to cloud data centers, proposing a set of requirements for the management of dynamic cloud data centers VPNs, but does not provide any specific solution. In this sense, our proposed SDN-based IPsec management framework could be applied there. Finally, it is worth mentioning that the centralized configuration of other security protocols is under study. In fact, at the time of writing, we can find active contributions defining YANG data models for TLS [10] and SSH [11] protocols.

Besides the standardization activity, we can find works analyzing how SDNs can be extended to support advanced security services like policy enforcement [54] or packet inspection [55]. Others are based on the management of TLS channels, such as [45] and [46]. Similarly, the protection of communication channels with IPsec has also attracted attention. Authors in [56] propose a method for integrating IPsec in SDN networks using OpenFlow, but does not detail the necessary extensions. The work presented in [57] proposes a new non-standardized southbound protocol to configure IPsec tunnels in gateways. The same challenge is faced by authors in [58] that propose a modular design consisting of different entities orchestrating the need of establishing IPsec SAs, creating IKE connections and applying IPsec protection to packets. This proposal requires the use of OpenFlow switches, lacks of a clear definition of the different APIs used and makes use of a non standard JSON (JavaScript Object Notation) description for the configuration required by the IPsec peers. Additionally, the solution does not contemplate essential IPsec management tasks such as rekeying.

5. SDN-based IPsec management framework

The proposed solution benefits from the IPsec design principle consisting in decoupling the process of applying security services to IP packets (*IPsec logic*) from the key management mechanism used to build the IPsec SAs [5]. The proposed framework is compliant with the general SDN architecture and, consequently, composed by two entities: the SDN controller, which is responsible for the key management processes; and the network resource, representing any kind of IPsec-capable entity (e.g. host or gateway).

We define two different cases, depending on whether the network resource takes part in the security association establishment (IPsec logic and IKEv2) or it is only in charge of protecting data flows (IPsec logic):

- *IKE case: IPsec logic and IKEv2 implemented in the network resource*. This model follows the traditional network deployment tendency where the network resources implement both the key management

[2] https://wiki.openstack.org/wiki/Neutron/VPNaaS.

protocol (i.e. IKEv2) and the IPsec logic. That is the key management mechanism is implemented between the SDN controller and the network resource. Thus, the SDN controller must provide the required information for SPD and PAD databases, and the IKEv2 configuration to the network resources. On the contrary, the SAD will be populated after the IKEv2 negotiation.
- *IKE-less case: Only IPsec logic is implemented in the network resource*. This model follows a more compliant SDN paradigm in the sense that the network resource is only in charge of processing data flows. Consequently, they only need to implement the IPsec logic. Therefore, the complete automated key management functionality is moved to the SDN controller, which is able to derive the IPsec SAs by itself. The SDN controller provides the required parameters to create valid entries in the SPD and SAD at the network resources. Since IKEv2 is not running in them, the PAD does not need to be configured.

5.1. Required interfaces

In order to develop both cases and make our solution functional, we require the definition of configuration and state data models, as well as different calls between SDN controllers and network administrators, networks resources or, even, with other SDN controllers. More precisely, as explained in Section 2.2, we require the definition of the following interfaces.

5.1.1. Northbound interface

It allows the administrator to configure the SDN controller with the general policies describing how to manage data flows between networks under its control. These policies are known as *Flow-based Protection Policies* (FPP) when they refer to security aspects. An FPP could specify something like *"Provide confidentiality to any data flow between internal network A and internal network B"*.

Administrators typically make use of an application service (implemented in the SDN application plane) that allows them to define those security requirements in a high level description language like, for example, *TSFNet* [20], *Frenetic* [21] or *Kinetic* [22]. Once defined, FPPs are sent to SDN controllers through the *northbound interface*, that is typically implemented using a REST-based API.

Unlike the southbound interface, there exists no standardized northbound modeling language. The Open Network Foundation (ONF) established a working group to focus on the northbound interface, not with the objective to define a new standard, but with the goal of establishing a consensus about this interface [50].

Within the IETF, two important efforts have tried to define high level modeling languages. First, the Network Modeling (NEMO) project [29] had the idea of defining an interface allowing users to express intent for network connectivity. This work was archived and only provided an initial specification of a high level language[15] and associated YANG data models [16]. The Simplified Use of Policy Abstractions (SUPA) working group suffer the same fate after developing an initial version of a generic information and data model [17,18] able to express policies at different levels of abstraction. At the time of writing, the I2NSF WG is working on the definition of YANG data models to allow the application layer to apply high level security policies into a SDN controller [48,49] following the *Event-Condition-Action* paradigm.

5.1.2. Southbound interface

This interface is required to carry out the interaction between the SDN controller and the network resource, so that the former can provide the latter with IPsec configuration, and receive from them IPsec-related notifications and state information.

In particular, to implement the southbound interface, we have chosen YANG as data modeling language along with NETCONF protocol to exchange IPsec-related configuration between SDN controller and the network resource.

According to NETCONF, the *edit-config* RPC is used to edit (add, modify, delete) elements of IPsec configuration data and *get/get-config* RPC to read IPsec configuration and state data. NETCONF *notifications* are also required, in order to inform about particular events from the network resource to the SDN controller (e.g. the imminent expiration of an IPsec SA or the need of creating a new one). As described in Section 2.3, these NETCONF messages are protected by means of protocols such as SSH or TLS.

Although this article has not the goal of showing the details of a YANG model for IPsec (specially because the detailed information can be found in the specification [8] undergoing standardization effort in I2NSF WG), we show in a comprehensive way the main features of the module expressing the information needed to design a proper SDN-based IPsec management. In particular, it is necessary to model the different IPsec databases (SPD, SAD and PAD) and IKE, which implies the definition of *IPsec and IKEv2 configuration data models* and *IPsec and IKEv2 operational (state) data models*. In a general way, these models reflect the information presented in Table 1.

In general, for designing these data models, we have taken as a reference the existing standards [5,7] for the modelling of IKEv2 protocol and the SPD, SAD, PAD databases. Moreover, we have complemented these analyses by studying the different IPsec interfaces for configuring the SPD and SAD: *1)* the standard PF_KEYv2 (RFC 2396) [27] to manage the SAD; *2)* the (de facto standard) KAME extensions to PK_KEYv2 [38] to manage the SPD; and, *3)* XFRM [39] to manage both SPD and SAD (only available in Linux Systems). Those are interfaces already implemented in current kernels of different operating systems. However, there are no existing standard interfaces to manage IKE and PAD. In fact, the PAD implementation is usually abstracted by the IKE implementation instead of being included in the kernel. Without prejudice to the generality of our design, we have taken as reference the *Versatile IKE Control Interface* (VICI) in order to model these elements [28].

Tables 2–5 show the relationship between the different southbound messages (i.e. NETCONF RPCs and notifications) and the implementation-dependent calls in the network resource to the IPsec/IKEv2 APIs.

5.1.3. East/west interface

It is defined between two SDN controllers and it is required in case two network resources managed by different controllers participate in the IPsec protection of a given flow. In that case, both SDNs have to

Table 2
Relationship between southbound messages and SAD configuration calls in the network resource.

Southbound NETCONF message	Local IPsec API (PF_KEYv2/XFRM)	Type	Explanation
edit-config(nc:create,sad)	SADB_ADD / XFRM_MSG_NEWSA	RPC	Creates an entry in the SAD
edit-config(nc:replace,sad)	SADB_UPDATE / XFRM_MSG_UPDSA	RPC	Update an existing entry in the SAD
edit-config(nc:delete,sad)	SADB_DELETE / XFRM_MSG_DELSA	RPC	Delete an existing entry in the SAD
get(sad)	SADB_GET / XFRM_MSG_DELSA	RPC	Retrieve configuration and state data information about an existing entry in the SAD
sad-acquire	SADB_ACQUIRE / XFRM_MSG_ACQUIRE	notification	Notify when a new IPsec SA is required
sad-expire	SADB_EXPIRE / XFRM_MSG_EXPIRE	notification	Notify when a existing IPsec SA is about to expire (soft lifetime reached) or has expired (hard lifetime reached)

agree on aspects such as peer authentication algorithms and keys, IPsec SAs to establish, etc. A potential candidate for this interface could be IKEv2, however, like the northbound interface, the east-west interface has not been standardized. Some proposals can be found in the literature such as [51,52] or [42].

5.2. General operation

In the following subsections we detail the operation for the aforementioned cases assuming a simple scenario with two network resources, hosts A and B. The network administrator, through the application plane, provides the SDN controller with, for example, the following FPP: *"Protect with confidentiality the communications between hosts A and B"*. The FPP implies the establishment of an IPsec SA with ESP in transport mode and will be translated by the SDN controller into specific IPsec policies installed in the involved network resources.

In general, the NETCONF requests received from the SDN controller are translated by the NETCONF server (located in the network resource) into local calls to the existing IPsec and IKEv2 APIs. The results obtained from these APIs will be translated into NETCONF responses and actions over the data models, or NETCONF notifications when receiving asynchronous events from them. Following the notation introduced in Tables 2–5, to describe the operation we abstract the RPC *edit-config* with a call with two parameters: the operation as defined in NETCONF (*nc:create, nc:replace, nc:delete*) and the IPsec database involved (*sad, spd, pad*) or the IKEv2 implementation (*ike*).

5.2.1. IKE case operation: IPsec logic and IKEv2 implemented in the network resource

Fig. 2 shows the architecture and operations (only host A is shown since a similar procedure will happen in host B) involved when the network resources deploy both an IKEv2 and IPsec implementation. For this explanation, the SDN controller makes use of IKEv2 *Pre-Shared Key* (PSK) authentication for both peers, although any other method could be used. Only the IKEv2 daemon, the PAD and the SPD need to be externally configured by the SDN controller. We have placed IKEv2 daemon and the PAD in the same box because, in general, the PAD-related information (e.g. authentication credentials) is typically included in the IKE configuration. The SAD is considered as state data and, therefore, the SDN controller might monitor (using the *get* RPC) the existing IPsec SAs.

As preliminary step, it is assumed that IKEv2 daemon registers to listen IPsec-related notifications from the kernel by means of the PFKEY_v2 SADB_REGISTER message (*0*).

After receiving the FPP through the northbound interface (*1*), the SDN controller generates the required IKE and PAD configuration such as, the peer's authentication method (PSK in this case), the credentials (the pre-shared key), the IKE SA y IPsec SAs characteristics, etc., for each network resource, and translates the FPP into the corresponding SPD and IKE/PAD information. Next, the SDN controller inserts this information in each network resource (host) by means of the southbound protocol. In particular, it invokes the RPC *edit-config(nc:create,ike)* to provide information to create a new IKE SA; *edit-config (nc:create,pad)* to create an entry in the PAD to store the PSK for authentication (*2*) and, finally, *edit-config(nc:create,spd)* to add a new IPsec policy into the SPD. This is translated by the NETCONF server into calls to the local interface provided by the IKEv2 implementation: for example, *load-conn()* to create state to establish the IKEv2 SA and some parameters to establish the IPsec SA (e.g. cryptographic algorithms); *load-shared()* to insert the PSK in the PAD (*3*); and, finally, into a local call to PFKEY_v2 with the message SADB_X_SPDADD to add the SPD entry (*3'*).

At this point, IKEv2/PAD and SPD are configured at the network resource. When an outbound IP packet is analyzed by the kernel (*4*), the IPsec logic checks the packet information (e.g. source and destination IP addresses) against the SPD (*5*) and finds the policy recently configured by the SDN controller. The IPsec logic checks the SAD (*6*) but does not find any entry to process the IP packet since it is the first packet of the flow. Thus, the kernel sends a SADB_ACQUIRE message (*7*) that is captured by the IKEv2 daemon, which delivers a SADB_GETSPI to create a state in the SAD (*8*) and starts an IKEv2 negotiation (*9*) with host B, which must be configured in a similar way by the SDN controller. The successful execution of IKEv2, which is possible thanks to the configuration provided by the SDN controller, allows updating the entry in the SAD using the SADB_UPDATE message (*10*). It is worth noting that steps (*3–10*) happen in the network resource without the involvement of the SDN controller. With the information in the SAD, the IP packet can now be processed (*11*) and the IKEv2 daemon may launch *ike-updown/child-updown* events (*12*) that the NETCONF server optionally uses to notify the controller (*12'*) about the creation of these SAs.

After some time the kernel sends a SADB_EXPIRE message in order to notify the imminent expiration of the IPsec SA (soft lifetime). In this case the IKEv2 daemon captures the notification and starts a new IKEv2 exchange with host B in order to renew the IPsec SA before its hard lifetime is reached. Fig. 2 does not show, for the sake of simplicity, neither NETCONF session establishment between the SDN controller and network resource (i.e. host A) nor notification subscription and events.

5.2.2. IKE-less operation: only IPsec logic is implemented in the network resource

When the network resource is only equipped with the IPsec logic (i.e., without IKEv2), the SDN controller makes use of the southbound API to configure the SPD and SAD (the PAD is not required since there is no IKEv2). Two workflows can be considered to perform this operation. In the *proactive* mode, the SDN controller provides network resources with all the necessary configuration (SPD and SAD) before any data flow to be protected arrives. Alternatively, when a *reactive* mode is followed, the SDN controller initially only configures network resources with the security policies (SPD) and defers the provision of the security associations configuration (SAD) when requested by network resources through the corresponding notification.

Fig. 3 shows the operation for IKE-less case assuming a reactive mode since it requires greater coordination between SDN controller and network resource. As initial step, we assume the NETCONF server (located in Host A) sends a *SADB_REGISTER* message through the PFKEY_v2 API (*0*) to the kernel so that it can capture future notifications.

When the administrator configures the FPP through the northbound interface (*1*), the SDN controller translates this into valid information for the SPD and invokes the RPC *edit-config(nc:create,spd)* to create

Table 3
Relationship between southbound messages and SPD configuration calls in the network resource.

Southbound NETCONF message	Local IPsec API (PF_KEYv2-KAME extensions / XFRM)	Type	Explanation
edit-config(nc:create,spd)	SADB_X_SPDADD / XFRM_MSG_NEWPOLICY	RPC	Creates new entry in the SPD
edit-config(nc:replace,spd)	SADB_X_SPDUPDATE / XFRM_MSG_UPDPOLICY	RPC	Update an existing entry in the SPD
edit-config(nc:delete,spd)	SADB_X_SPDDELETE / XFRM_MSG_DELPOLICY	RPC	Delete an existing entry in the SPD
get-config(spd)	SADB_X_SPDGET / XFRM_MSG_GETPOLICY	RPC	Retrieve an existing entry in the SPD

Table 4
Relationship between southbound messages and PAD configuration calls in the network resource.

Southbound	Local IKEv2 API (VICI)	Type	Explanation
edit-config(nc:create,pad)	load-shared(), load-key(), load-cert(), load-authority(), load-conn()	RPC	Creates new entry in the PAD
edit-config(nc:replace,pad)	load-shared(), load-key(), load-cert(), load-authority(), load-conn()	RPC	Update an existing entry in the PAD
edit-config(nc:delete,pad)	clear-creds(), flush-certs(), unload-authority()	RPC	Delete information in the PAD
get-config(pad)	get-authorities(), list-certs()	RPC	Retrieve information from the PAD

entries in the SPD (*2*). The NETCONF server translates this RPC into a local call to PFKEY_v2 with the message SADB_X_SPDADD (2').

When a first IP packet of an outbound flow (*3*) matches the IPsec policy configured by the SDN controller (*4*), the SAD does not have any information about how the IPsec logic can protect it (*5*). As a consequence, the kernel raises a SADB_ACQUIRE message (*6*). The NETCONF server captures this notification message and, immediately, sends a NETCONF notification *sad-acquire* to the SDN controller (*6'*). With this notification, the SDN controller knows that must configure IPsec SAs in both directions, inbound and outbound, for each network resource (host A and host B). To this end, the SDN controller selects the SPIs, generates the key material (encryption keys in this example) for the IPsec SAs in both hosts and emits a RPC *edit-config(nc:create,sad)* (*7*). The NETCONF server translates this call into a local SADB_ADD message for the kernel (*7'*) that ends up with a new entry in the SAD. After that, the IP packet can be protected (*8*).

Similarly to IKE case, when the IPsec SA soft lifetime expires, the kernel also sends a SADB_EXPIRE message, which is translated by the NETCONF server into a *sad-expire* notification for the SDN controller. Then, the SDN controller can start the process described in Section 6.3 to update the IPsec SA before its hard lifetime expires. As before, NETCONF session establishment, notification subscription and notification messages are omitted in Fig. 3 for simplicity.

6. Discussion and challenges

The IKE case and IKE-less case offer two different alternatives to achieve the SDN-based IPsec management. Now, we analyze some important aspects for the future usage of these cases.

6.1. Deployment

Regarding the network resources, Figs. 2 and 3 show that, the resource has to deploy a NETCONF server shipped with the YANG model for IPsec management, and the corresponding implementation is able to apply configuration data and read state data from the system. In both cases, the resource has to be equipped with the legacy IPsec kernel implementation. Besides, in IKE case, a legacy IKEv2 implementation has to be present in the resource.

Considering that traditional network resources already ship an IKEv2 implementation, we foresee in the shorter-term a quicker and more prevalent deployment of IKE case, since the integration with the SDN controller would only imply sending configuration information and monitoring existing SPD and IKEv2 implementations. Nevertheless, in the longer-term we expect that IKE-less case deployments will start to arise, specially in scenarios where *constrained* devices are required (e.g. Internet of Things).

6.2. NAT Traversal management

IKEv2 implements the support for establishing IPsec SAs between peers that are behind *Network Address Translation* (NAT) networks (*NAT Traversal*). Thanks to this feature some IKEv2 messages and IPsec ESP packets are encapsulated in UDP or TCP to deal with NAT without any problem [33,34].

IKEv2 does not represent any additional challenge beyond the configuration of IKEv2 NAT Traversal support [7]. However, IKE-less case adds challenges to the SDN controller. In particular, the SDN controller needs to know, by means of some mechanism (e.g. the network resource informs the SDN controller), whether both peers are behind a NAT before applying any IPsec configuration and configuring an UDP or TCP tunnel.

6.3. Rekeying process

Rekeying refers to the process of creating a new IPsec SA to renew an old one. As explained in Section 2.1, an IPsec SA must be replaced with a new one when its soft lifetime is completed, and removed from the SAD when its hard lifetime is reached. Thus, rekeying is a critical task because IPsec SAs have to be renewed before they are removed in order to avoid traffic interruption.

IKE case is simple because the SDN controller has only to provide the soft and hard lifetime values to IKEv2 implementation, which will be in charge of the rekeying process. On the contrary, IKE-less case adds some complexity. For example, the SDN controller needs to know when the rekey should start. The network resource's kernel usually provides an event (SADB_EXPIRE in the PF_KEYv2 API) indicating if it is a soft or hard expiration. IPsec makes use of this soft expire notification to solicit the rekey. As explained in Section 5.2.2, in IKE-less case, this event has to travel to the SDN controller to be processed, adding some delay that may affect to the proper processing of this event.

An approach to solve the rekeying process for IKE-less case is described in Fig. 4: let's suppose two network resources (A and B) have the corresponding IPsec SAs required to apply IPsec security to the communication channel. So A has one outbound IPsec SA for the outgoing B traffic (*SPI_i*), and one inbound IPsec SA for the incoming A traffic (*SPI_j*), and analogous for network resource B. Before one IPsec SA expires (e.g. *SPI_j*), the SDN controller is notified (soft lifetime

Table 5
Relationship between southbound messages and IKEv2 configuration calls in the network resource.

Southbound	Local IKEv2 API (VICI)	Type	Explanation
edit-config(nc:create,ike)	load-conn()	RPC	Creates a new IKE SA configuration
edit-config(nc:replace,ike)	load-conn()	RPC	Update an existing IKE SA configuration
edit-config(nc:delete,ike)	unload-conn()	RPC	Remove an existing IKE SA configuration
get-config(ike)	get-conns(), list-sas(), list-policies(), get-algorithms()	RPC	Retrieve information from the PAD
ike-updown	ike-updown	notification	IKE_SA is established or terminated.
ike-rekey	ike-rekey	notification	IKE_SA is rekeyed
child-updown	child-updown	notification	CHILD_SA (i.e. IPsec_SA) is established or terminated
child-rekey	child-rekey	notification	CHILD_SA (i.e. IPsec_SA) is rekeyed

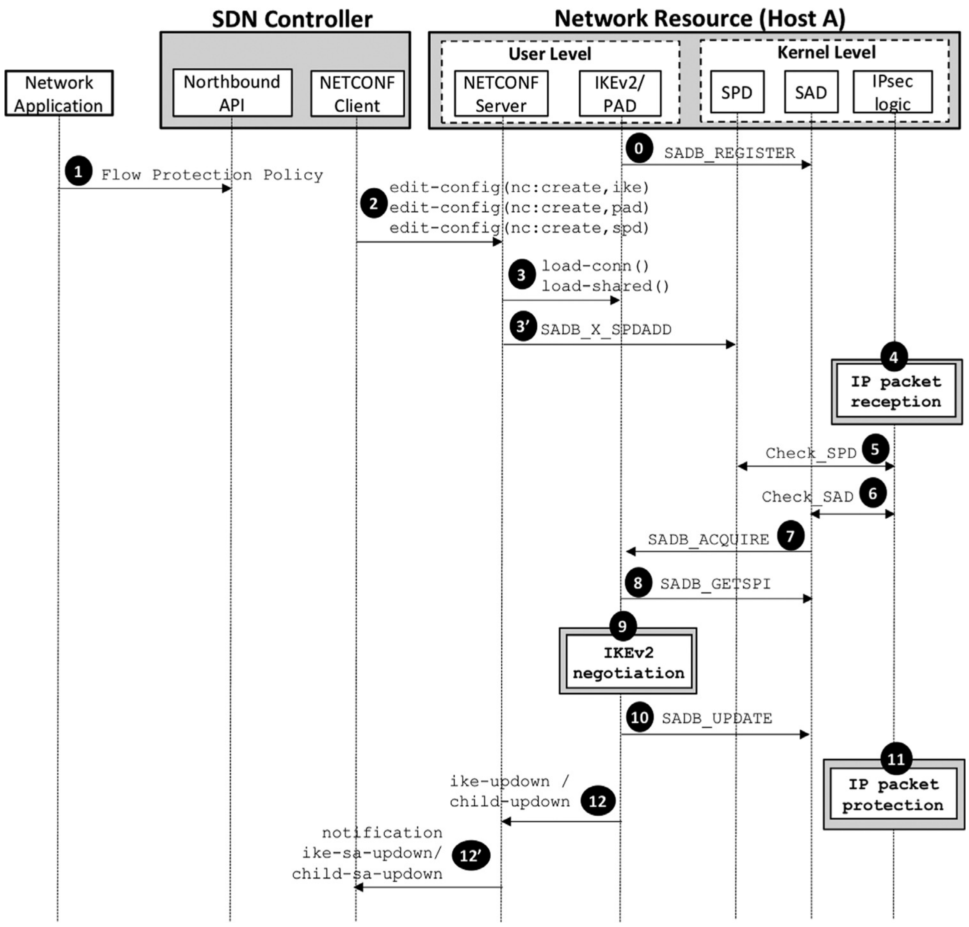


Fig. 2. IKE case: IPsec logic and IKE implemented in the network resource.

notification), then the SDN controller creates and installs, in parallel, two new inbound IPsec SAs in A and B (*SPI_j + 1* and *SPI_i + 1*, respectively) (1). In the same way, once applied, the SDN controller creates and installs two new outbound IPsec SAs in A and B (*SPI_i + 1* and *SPI_j + 1*, respectively) (2). At this point, the traffic in any direction can be protected by the new inbound and outbound IPsec SAs. Finally, the SDN controller deletes the old inbound and outbound IPsec SAs from A and B (3).

As observed, the rekeying process in IKE-less case requires SDN controller to send three RPC (*edit-config*) operations for each network resource, and it has to be finished before the hard lifetime expires. On the contrary, the IPsec SA is removed from the system and network traffic can be lost. This aspect needs to be taken into consideration by SDN controller in order to configure appropriate soft and hard lifetimes for each network resource.

6.4. Network resource starts/reboots

If a network resource restarts the IPsec state (*failed* network resource), the IPsec SAs information may become inconsistent between peers. Thus, a mechanism to detect the loss of this state is required (e.g. the NETCONF/TCP connection is broken) in order to allow the SDN controller to react accordingly by creating a consistent IPsec state in the affected network resources.

For example, in IKE case, the SDN controller will have to configure the failed network resource with new IKEv2, SPD and PAD information. It will also send new parameters (e.g. a new fresh PSK for authentication) to the network resources (non-failed) that have IKEv2 SAs and IPsec SAs with the failed one.

In IKE-less case, it will delete the old IPsec SAs of the non-failed network resources established with the failed one. If the failed network resource comes to live, the SDN controller will configure the new IPsec SAs for the network resource with the rest of network resources as described in Section 5.2.2.

7. Security considerations

Since this proposal is based on the SDN paradigm, it shares the general security concerns derived from the existence of a centralized entity (SDN controller) representing a single point of failure and that can be target of malicious attacks [40]. Thus, the SDN controller needs to be tightly protected. This implies to not only enforce strong access control mechanisms and ensure availability, but also to establish secure communication channels between SDN controller and network resources (southbound interface) and application services (northbound interface) [6].

In this solution, communication between the SDN controller and network resources is based on NETCONF. This protocol provides strong access control and authentication mechanisms. Moreover it requires the usage of a secure transport protocol (e.g. SSH or TLS) providing authentication, confidentiality, integrity and replay protection. This framework takes advantage of this protected channel to exchange not only IPsec configuration parameters but also keying material, thus accomplishing the protection mechanisms required for the distribution of cryptographic information [53]. In this sense, it is worth mentioning the importance of using protection techniques providing, at least, the same security strength of the transported keying material. In other words, the encryption key used by SSH or TLS must be equal or greater than the one being distributed to the network resource.

Additionally, some other security measures must be taken into

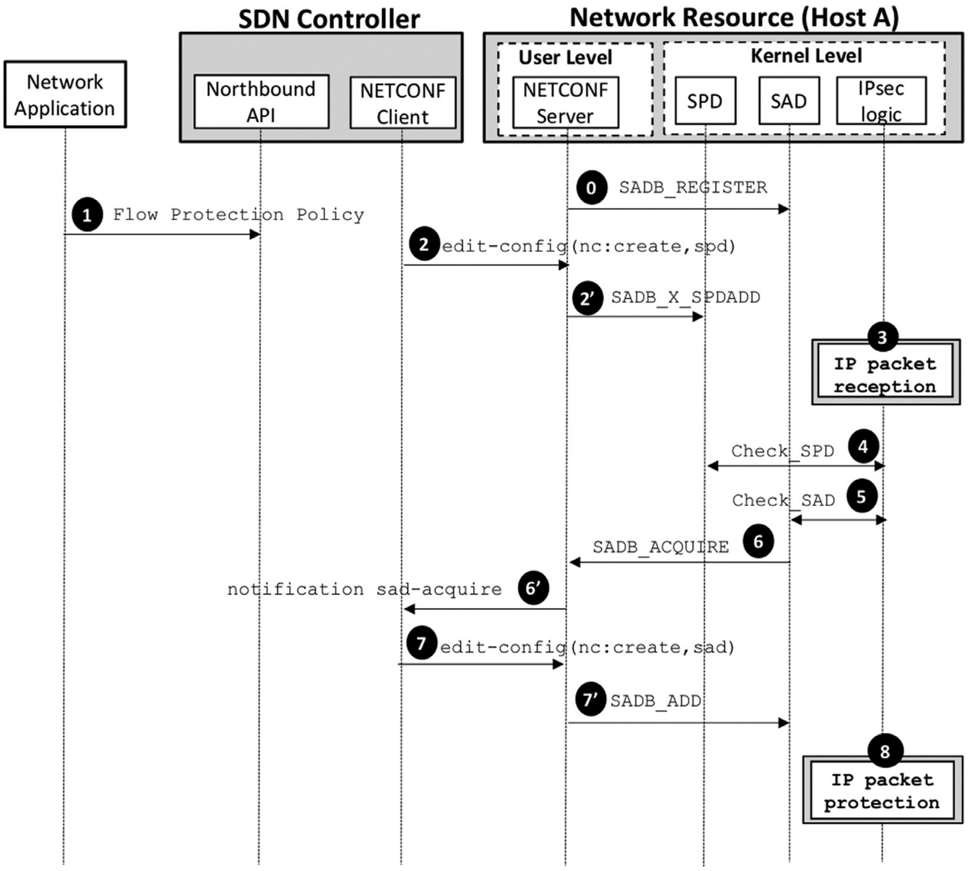


Fig. 3. IKE-less case: Only IPsec logic is implemented in the network resource (reactive mode).

account derived from the fact that the SDN controller manipulates cryptographic material. The specific actions depend on the operation case. In IKE case, the controller applies IKE credentials into the network resource. In this case, the SDN controller must not keep these credentials after distributing them, and the network resource must not allow the SDN controller to read them after been applied. If PSK authentication is used in IKEv2, the SDN controller must remove it immediately after generation and distribution. In the case of raw public keys, the recommendation for the SDN controller is to remove the associated private key immediately after generation and distribution. Finally, if certificates are used, the network resource is able generate the private key and exports the public key for certification to the SDN controller.

Regarding IKE-less case, the SDN controller sends the IPsec SA information to the network resource, that includes the keys for integrity and encryption. The recommendation for the SDN controller is to not store the keys after generation and distribution. Besides, the network resource must not allow the reading of these values once they have been applied.

The research about mechanisms to avoid the SDN controller to know cryptographic material used to generate the IPsec SAs is letf as

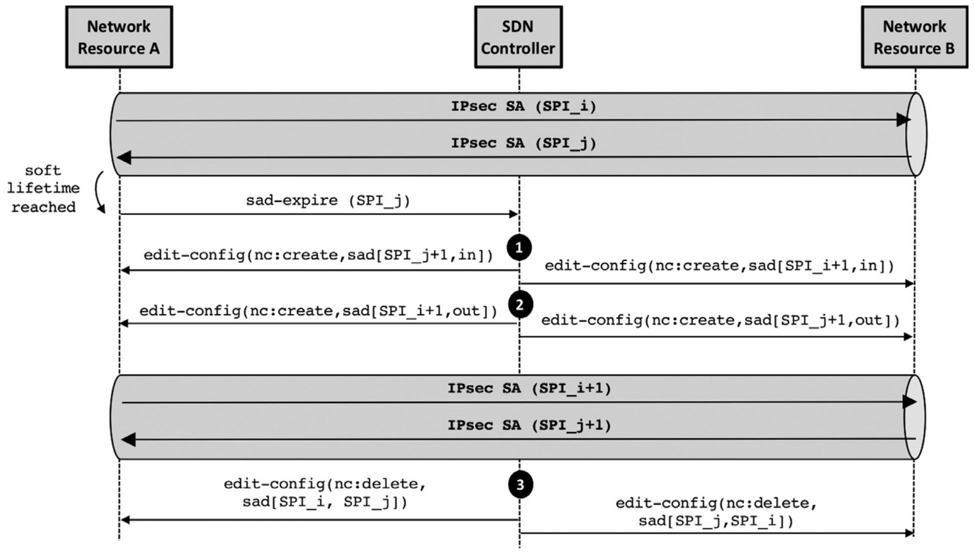

Network Resource A
SDN Controller
Network Resource B
IPsec SA (SPI_i)
IPsec SA (SPI_j)
soft lifetime reached
sad-expire (SPI_j)
1
edit-config(nc:create,sad[SPI_j+1,in])
edit-config(nc:create,sad[SPI_i+1,in])
2
edit-config(nc:create,sad[SPI_i+1,out])
edit-config(nc:create,sad[SPI_j+1,out])
IPsec SA (SPI_i+1)
IPsec SA (SPI_j+1)
3
edit-config(nc:delete, sad[SPI_i, SPI_j])
edit-config(nc:delete, sad[SPI_j,SPI_i])

Fig. 4. Rekey scenario for IKE-less case.

future work. In the literature we can find several options, such as [14], which proposes the use of SDN controllers as trusted entities for distributing public keys between network resources, so that they are able to generate symmetric keys that cannot be neither observed nor inferred by the SDN controller. Unfortunately, this proposal has not evolved beyond a preliminary design.

Finally, another option might be to follow a *secret sharing* [31] strategy where several SDN controllers participate in the generation of the key material, thus forcing an attacker to compromise all them to access the generated key.

8. Performance analysis

This section provides a simple performance analysis of IKE and IKE-less cases, in term of number of *Round-Trip Time (RTT)* involved for the establishment of IPsec SAs between any two network resources and their rekey. The processing time is considered negligible in comparison with the transmission time. First, we analyze the total RTT for IKE case, then, the total RTT for IKE-less case for both reactive and proactive modes. Finally, a short comparison is provided.

Let RTT_{NETCONF} be the RTT for a single NETCONF exchange between a NETCONF client and a NETCONF server. For example, an RPC *edit-config* or *create-subscription*, including the corresponding RPC *ok* response for each operation.

Let $RTT_{\text{NETCONF\_X}}$ be the RTT of operation type X in NETCONF [23], where X is one of the following values:

- E: *edit-config* operation.
- S: *create-subscription* operation.

Then we can roughly assume that:

$$RTT_{\text{NETCONF\_E}} = RTT_{\text{NETCONF\_S}} = RTT_{\text{NETCONF}} \tag{1}$$

We denote NETCONF notifications like *sad-expire*, when a SA entry expires in the SAD, and *sad-acquired*, when a SA entry is required to protect a data flow, but it is not found in the SAD.

Since NETCONF notifications only require one single NETCONF message (i.e. has no associated response), so we can assume that the time required for each notification is $\frac{RTT_{\text{NETCONF}}}{2}$. Note that additional notifications, for example these sent by the server after some change takes place in a datastore, are optional. Therefore they are omitted in this study.

Let $RTT_{\text{IKE\_Y}}$ the RTT of the type Y exchange in IKE [7], where Y can be:

- S: *IKE_SA_INIT* exchange.
- A: *IKE_AUTH* exchange.
- C: *IKE_CREATE_CHILD_SA* exchange.

We can roughly assume that this RTT per each type of exchange has the same value RTT_{IKE}, namely:

$$RTT_{\text{IKE\_S}} = RTT_{\text{IKE\_A}} = RTT_{\text{IKE\_C}} = RTT_{\text{IKE}} \tag{2}$$

Finally, let T_{IKE} the total time for IKE case, $T_{\text{IKE-less-reac}}$ the total time for IKE-less case reactive mode, and $T_{\text{IKE-less-proac}}$ the total time for IKE-less case proactive mode. Taking into account the notation explained above, we proceed to estimate these values assuming that:

- A NETCONF session is already established between the NETCONF client (SDN controller) and NETCONF server along the IPsec SA management process. Since this only happens once before IPsec management service can be carried out and it is common to both cases, this initial session establishment time is not taken into account.
- The NETCONF session establishment includes the NETCONF capabilities exchange and the required schema's requests and responses. Thus, we do not take into account these tasks.
- IPsec SAs are renewed r times (each renewal is initiated after receiving a *soft* lifetime *sad-expire* notification) before they are removed (after receiving the hard lifetime *sad-expire* notification).
- Two IPsec management operations over two network resources are applied in parallel.

8.1. IKE case

To calculate T_{IKE} we have additionally assumed that:

- IKE peers authenticate by using PSK or certificate based authentication. In other words, only one RTT is required for the authentication process.
- Since IKE notifications shown in Fig. 2 are optional, they are not taken into account in this basic study. However, for completeness, we contemplate the necessary subscription requested by the NETCONF client to the server.

In this case the T_{IKE} is:

$$\begin{aligned} T_{\text{IKE}} &= RTT_{\text{NETCONF\_S}} + RTT_{\text{NETCONF\_E}} + RTT_{\text{IKE\_S}} \\ &\quad + RTT_{\text{IKE\_A}} + \textstyle\sum_{i=1}^{r} RTT_{\text{IKE\_C}} \end{aligned} \tag{3}$$

In (3) $RTT_{\text{NETCONF\_S}}$ and $RTT_{\text{NETCONF\_E}}$ include the RTT for *create-subscription* and *edit-config* operations for both network resources, respectively. Once the IKE configuration is applied, when data flow has to be protected, IKE start the IKE_SA_INIT and IKE_AUTH exchanges, and the traffic is protected. Finally, each time the IKE_SA expires a new IKE_CREATE_CHILD_SA exchange is required.

Based on (1) and (2), T_{IKE} can be described as:

$$T_{\text{IKE}} = 2{*}RTT_{\text{NETCONF}} + (2 + r){*}RTT_{\text{IKE}} \tag{4}$$

8.2. IKE-less case

Before calculating $T_{\text{IKE-less-reac}}$ and $T_{\text{IKE-less-proac}}$, it is worth highlighting the following assumptions:

- For the sake of clarification, we assume an IPsec SA only expires in one of the endpoints.
- IPsec SA rekey takes places after receiving the *soft* lifetime *sad-expire* notification. *hard* lifetime *sad-expire* notifications are not taken into account since they simply assume that IPsec SAs are removed from the network resource when this hard lifetime is reached.
- We consider the rekey process explained in Section 6.3.

As we described in Section 5.2.2, there are two operation modes: reactive, where the SDN controller installs the IPsec SA only when the network resource needs them; and proactive, where the SDN controller installs the IPsec SA even before network resource observes data flow that requires IPsec. In reactive mode, the $T_{\text{IKE-less-case-reac}}$ can be calculated as follows:

$$\begin{aligned} T_{\text{IKE-less-reac}} &= RTT_{\text{NETCONF\_S}} + RTT_{\text{NETCONF\_E}} + \frac{RTT_{\text{NETCONF}}}{2} \\ &\quad + RTT_{\text{NETCONF\_E}} + \textstyle\sum_{i=1}^{r} \left(\frac{RTT_{\text{NETCONF}}}{2} + 3{*}RTT_{\text{NETCONF\_E}}\right) \end{aligned} \tag{5}$$

In this case, after the SDN controller's subscription ($RTT_{\text{NETCONF\_S}}$) in the network resource, the SDN controller sends a NETCONF *edit-config* message including the policies to be applied by the network resource (SPD entries), but not the SAD entries. When the data flow to be protected arrives at one of the network resources, it sends a NETCONF *sad-acquire* notification message ($\frac{RTT_{\text{NETCONF}}}{2}$). The SDN controller then inserts the related SAD entries with a new *edit-config* message ($RTT_{\text{NETCONF\_E}}$). Then, for each IPsec SA renewal (*rekey*) a new soft *sad-*

expire notification arrives, and the three NETCONF *edit-config* operations required by the rekey process take place. These operations can be applied in parallel for each network resource.

Based on (1), this equation can be simplified as:

$$T_{\text{IKE-less-reac}} = (3,5 + 3,5r)^{*}RTT_{\text{NETCONF}} \tag{6}$$

If we opt by using a *proactive* configuration from the SDN controller, the $T_{\text{IKE-less-proac}}$ is defined as:

$$T_{\text{IKE-less-proac}} = RTT_{\text{NETCONF\_S}} + RTT_{\text{NETCONF\_E}} + \sum_{i=1}^{r}\left(\frac{RTT_{\text{NETCONF}}}{2} + 3^{*}RTT_{\text{NETCONF\_E}}\right) \tag{7}$$

After the NETCONF subscription exchange (*create-subscription*), the SDN controller applies immediately (without waiting any *sad-acquire*) notification the network resource's configuration (*edit-config*) including SPD and SAD related information so the traffic can be protected. The rekey process is similar to the one described for the reactive mode.

Based on (1), this equation can be simplified as:

$$T_{\text{IKE-less-proac}} = (2 + 3,5r)^{*}RTT_{\text{NETCONF}} \tag{8}$$

8.3. IKE case vs IKE-less case

Let's now compare IKE case and IKE-less case (reactive mode) based on Eqs. (4) and (6). The objective is to know when T_{IKE} is better or worse than $T_{\text{IKE-less-reac}}$ and. If we compare both equations:

$$2^{*}RTT_{\text{NETCONF}} + (2 + r)^{*}RTT_{\text{IKE}} = (3,5 + 3,5r)^{*}RTT_{\text{NETCONF}} \tag{9}$$

$$(2 + r)^{*}RTT_{\text{IKE}} = (3,5 + 3,5r)^{*}RTT_{\text{NETCONF}} - 2^{*}RTT_{\text{NETCONF}} \tag{10}$$

$$(2 + r)^{*}RTT_{\text{IKE}} = 1,5^{*}RTT_{\text{NETCONF}} + 3,5r^{*}RTT_{\text{NETCONF}} \tag{11}$$

$$(2 + r)^{*}RTT_{\text{IKE}} = (1,5 + 3,5r)^{*}RTT_{\text{NETCONF}} \tag{12}$$

$$\frac{(2 + r)^{*}RTT_{\text{IKE}}}{1,5 + 3,5r} = RTT_{\text{NETCONF}} \tag{13}$$

From (13) we can observe that for r ≥ 1 and when the number of rekey processes increases ($lim_{r\to\infty}$) the equation tends to:

$$\frac{RTT_{\text{IKE}}}{3,5} \approx 0,29^{*}RTT_{\text{IKE}} \approx RTT_{\text{NETCONF}} \tag{14}$$

In other words when the RTT_NETCONF is around 29% of the RTT_{IKE} both T_{IKE} and $T_{\text{IKE-less-reac}}$ are equivalent. This also gives a hint about a rough relationship between the speed of the link between the network resource and the SDN controller (control plane network) and the link between network resources (data plane network). Following (14), if $RTT_{\text{NETCONF}} > \frac{RTT_{\text{IKE}}}{3,5}$, then $T_{\text{IKE-less-react}} > T_{\text{IKE}}$. That is, IKE-less case (reactive mode) behaves worse than IKE case. On the contrary, if $RTT_{\text{NETCONF}} < \frac{RTT_{\text{IKE}}}{3,5}$, then, IKE-less case (reactive mode) provides a lower total time than IKE case.

Comparing IKE case and IKE-less case working on proactive mode, using Eqs. (4) and (8), we obtain similar results when the number of rekeys increases over time ($lim_{r\to\infty}$).

9. Proof-of-Concept implementation

We have set up a functional proof-of-concept where the framework described in this article has been put into practice. The main objective has been to show the correct operation of the SPD, SAD, PAD and IKEv2 YANG configuration models and how they can be manipulated in the network resource by using NETCONF. For the model required to represent IPsec configuration parameters we have make use of the YANG model described in [8].

9.1. Implementation details

Using VirtualBox and Linux Containers (LXC), we have deployed a virtual machine running *Ubuntu-16.04.1 Server* that plays the role of SDN controller; and four additional Ubuntu Server's LXC containers playing the role of network resources (2 hosts and 2 gateways). Only the southbound interface based on NETCONF has been deployed.

From the point of view of the SDN controller, we have tested the scenario with different pieces of software. More specifically, we have made use of *netopeer-cli*[3] to simulate the behaviour of a real SDN controller. This is a command line NETCONF client utility that allows to send messages to the server installed in each network resource. Beside, we have deployed *ONOS*[4] and implemented a basic ONOS application to establish IPsec connection between network resources. Finally, the NETCONF Python library *ncclient*[5] is being used to implement a SDN controller specialized in managing IPsec SAs.

From the point of view of the network resource, the NETCONF server should be able to apply the configuration received from the SDN controller in the YANG model into the corresponding IPsec and IKE configurations. To do this, *libnetconf* provides the required callbacks associated with the YANG model elements that can trigger changes in the configuration. For example, to create a new SAD or SPD entry, or to configure a new IKEv2 connection. An implementation must make use of these callbacks to modify the required configuration: *running-config* for the in-memory running configuration, or the *startup-config* for the configuration to be applied during the startup process.

For IKE case, the callbacks implemented by the NETCONF server make use of the VICI API implementation provided by strongswan [32] (*libvici.h*) to manage the IKEv2 (*charon*) daemon. In this case, configuration parameters provided by the SDN controller have to be translated into IKE configuration by means of VICI API calls, which allows hot configuration.

For IKE-less case, in order to apply these configurations and events in the server, we have make use of the kernel utilities and libraries such as the libpfkeyv2 (*linux/pfkeyv2.h*) to manage SAD and SPD operations. In this case, the callbacks implemented by the NETCONF server make use of PKKEY_V2 sockets to send *sadb_msg* elements including structures such as *sadb_x_policy* and *sadb_x_ipsecrequest* for SPD configuration, and *sadb_sa, sadb_x_sa2* and *sadb_lifetime* for SAD configuration. In order to deal with kernel notifications, the NETCONF server is registered for receiving these notifications by means of the *SADB_REGISTER* message sent through the PFKEY_v2 socket. The NETCONF server then listens for notification such as *sadb_acquire*, when a flow in the kernel matches a SPD entry but there is not SAD entry associated, or *sadb_expire*, when the lifetime of a SAD expires. The NETCONF server defines the callbacks to capture these notifications, to extract the relevant information, and to construct the NETCONF notifications messages to be sent to the SDN controller.

It's worth note that, for this proof of concept, we have make use of PFKEY_v2 because it is the current standard for kernel key management, although there are other more powerful solutions, such as NETLINK (*net/netlink.h*) which provides a richer output of kernel state data and additional advanced configurations.

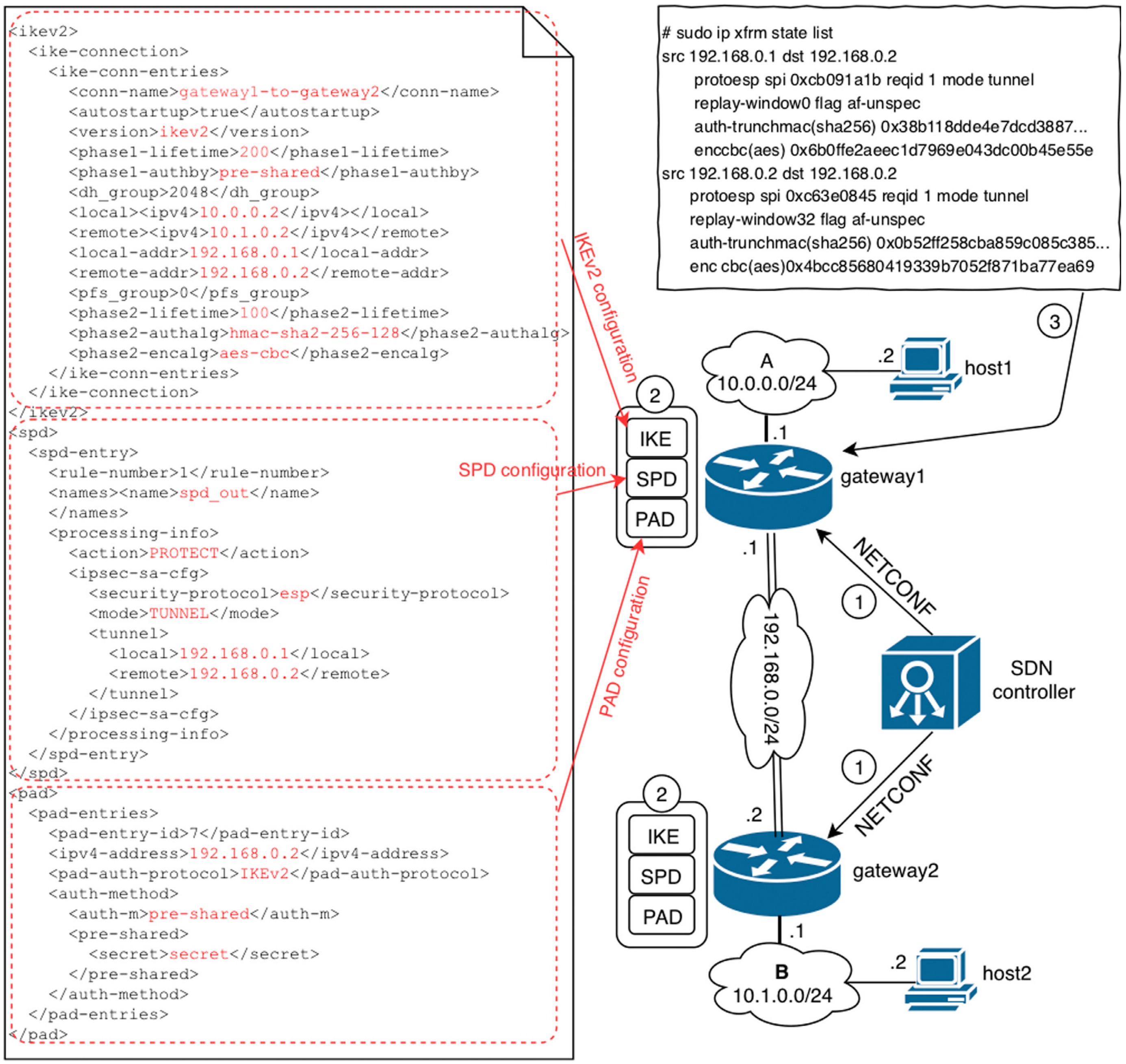
9.2. Deployment scenario

Figs. 5 and 6 show an example of the processes for IKE case and IKE-less case, respectively. Despite these examples assume the use of IPsec in gateway to gateway scenario, it is worth mentioning that the operation would be quite similar in a host to host scenario. Suppose the network administrator wants to configure an IPsec tunnel to provide

[3] https://github.com/CESNET/netopeer.

[4] https://onosproject.org.

[5] https://pypi.org/project/ncclient/.



```
<ikev2>
  <ike-connection>
    <ike-conn-entries>
      <conn-name>gateway1-to-gateway2</conn-name>
      <autostartup>true</autostartup>
      <version>ikev2</version>
      <phase1-lifetime>200</phase1-lifetime>
      <phase1-authby>pre-shared</phase1-authby>
      <dh_group>2048</dh_group>
      <local><ipv4>10.0.0.2</ipv4></local>
      <remote><ipv4>10.1.0.2</ipv4></remote>
      <local-addr>192.168.0.1</local-addr>
      <remote-addr>192.168.0.2</remote-addr>
      <pfs_group>0</pfs_group>
      <phase2-lifetime>100</phase2-lifetime>
      <phase2-authalg>hmac-sha2-256-128</phase2-authalg>
      <phase2-encalg>aes-cbc</phase2-encalg>
    </ike-conn-entries>
  </ike-connection>
</ikev2>
<spd>
  <spd-entry>
    <rule-number>1</rule-number>
    <names><name>spd_out</name>
    </names>
    <processing-info>
      <action>PROTECT</action>
      <ipsec-sa-cfg>
        <security-protocol>esp</security-protocol>
        <mode>TUNNEL</mode>
        <tunnel>
          <local>192.168.0.1</local>
          <remote>192.168.0.2</remote>
        </tunnel>
      </ipsec-sa-cfg>
    </processing-info>
  </spd-entry>
</spd>
<pad>
  <pad-entries>
    <pad-entry-id>7</pad-entry-id>
    <ipv4-address>192.168.0.2</ipv4-address>
    <pad-auth-protocol>IKEv2</pad-auth-protocol>
    <auth-method>
      <auth-m>pre-shared</auth-m>
      <pre-shared>
        <secret>secret</secret>
      </pre-shared>
    </auth-method>
  </pad-entries>
</pad>
```

```
# sudo ip xfrm state list
src 192.168.0.1 dst 192.168.0.2
      protoesp spi 0xcb091a1b reqid 1 mode tunnel
      replay-window0 flag af-unspec
      auth-trunchmac(sha256) 0x38b118dde4e7dcd3887...
      enccbc(aes) 0x6b0ffe2aeec1d7969e043dc00b45e55e
src 192.168.0.2 dst 192.168.0.2
     protoesp spi 0xc63e0845 reqid 1 mode tunnel
     replay-window32 flag af-unspec
     auth-trunchmac(sha256) 0x0b52ff258cba859c085c385...
     enc cbc(aes)0x4bcc85680419339b7052f871ba77ea69
```

Fig. 5. Deployment example for IKE case.

confidentiality and integrity protection to the traffic between organizations A (*10.0.0.0/24*) and B (*10.1.0.0/24*) through *gateway1* (*10.0.0.1*) and *gateway2* (*10.1.0.1*), for example, between two hosts: *host1* (*10.0.0.2*) and *host2* (*10.1.0.2*).

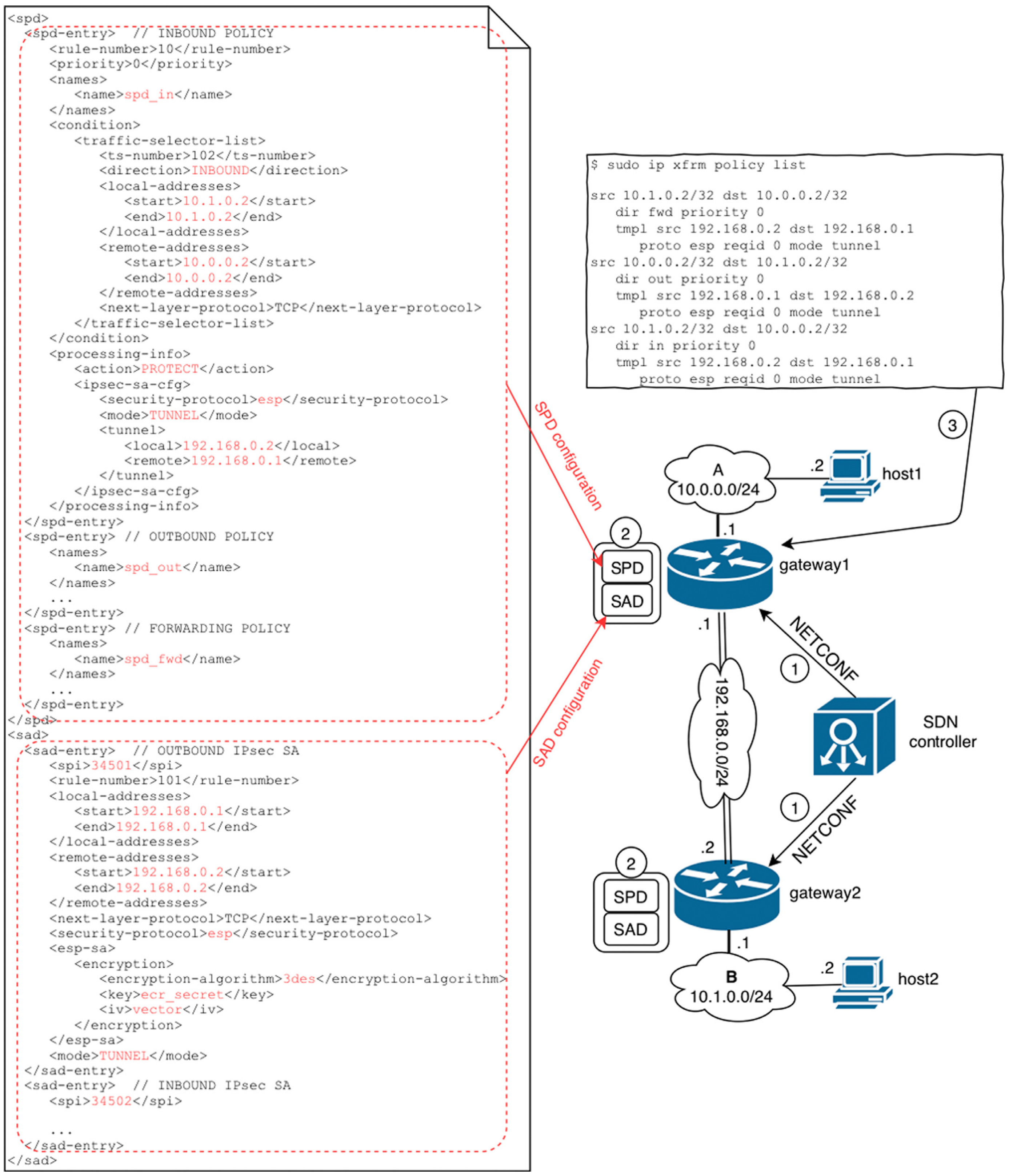
The resulting NETCONF message (RPC *edit-config*), containing the corresponding configurations for each network resource, is sent from the SDN controller using the NETCONF client to the corresponding NETCONF server (1). Once received, each network resource executes the corresponding callback function to apply the required configuration (2).

9.3. IKE case example

Specifically, Fig. 5 shows the IKE case, where we can observe the details for *gateway1* (similar for *gateway2*).

For IKEv2, the *ike-conn-entries* is used to represent the configuration between two IKEv2 peers, including information required for IKE SA (phase1) and the IPsec SAs (phase2). In the example, the configuration establishes the IKE version (*version/v2*), IKE SA lifetime (*phase1-lifetime/200s*), the kind of IKE peers authentication method (*phase1-authby/pre-shared*), DH group (*dh_group/2048*), the traffic that has to be protected (*local/10.0.0.2, remote/10.1.0.2*), the IP addresses of the IKEv2 endpoints identities (*local-addr/ipv4/192.168.0.1, remote-addr/ipv4/192.168.0.2*), IKE SA and IPsec SAs cryptographic suite (*phase2-authalg/hmac-sha2-256-128, phase2-encalg/aes-cbc*), and IPsec SA lifetime (*phase2-lifetime/100*) in seconds.

The SPD configuration element (*spd-entry*) provides additional information about each single IPsec SA. This element is in common for IKE case and IKE-less case. The example shows a descriptive name (*name/spd_out*), the kind of action to be applied to the traffic (*action/PROTECT*), security protocol (*security-protocol/ESP*), and the IPsec mode (*mode/TUNNEL*), including the tunnel endpoints (*local/*



```
<spd>
  <spd-entry>  // INBOUND POLICY
     <rule-number>10</rule-number>
     <priority>0</priority>
     <names>
        <name>spd_in</name>
     </names>
     <condition>
        <traffic-selector-list>
           <ts-number>102</ts-number>
           <direction>INBOUND</direction>
           <local-addresses>
              <start>10.1.0.2</start>
              <end>10.1.0.2</end>
           </local-addresses>
           <remote-addresses>
              <start>10.0.0.2</start>
              <end>10.0.0.2</end>
           </remote-addresses>
           <next-layer-protocol>TCP</next-layer-protocol>
        </traffic-selector-list>
     </condition>
     <processing-info>
        <action>PROTECT</action>
        <ipsec-sa-cfg>
           <security-protocol>esp</security-protocol>
           <mode>TUNNEL</mode>
           <tunnel>
              <local>192.168.0.2</local>
              <remote>192.168.0.1</remote>
           </tunnel>
        </ipsec-sa-cfg>
     </processing-info>
  </spd-entry>
  <spd-entry> // OUTBOUND POLICY
     <names>
        <name>spd_out</name>
     </names>
     ...
  </spd-entry>
  <spd-entry> // FORWARDING POLICY
     <names>
        <name>spd_fwd</name>
     </names>
     ...
  </spd-entry>
</spd>
<sad>
  <sad-entry>  // OUTBOUND IPsec SA
     <spi>34501</spi>
     <rule-number>101</rule-number>
     <local-addresses>
        <start>192.168.0.1</start>
        <end>192.168.0.1</end>
     </local-addresses>
     <remote-addresses>
        <start>192.168.0.2</start>
        <end>192.168.0.2</end>
     </remote-addresses>
     <next-layer-protocol>TCP</next-layer-protocol>
     <security-protocol>esp</security-protocol>
     <esp-sa>
        <encryption>
           <encryption-algorithm>3des</encryption-algorithm>
           <key>ecr_secret</key>
           <iv>vector</iv>
        </encryption>
     </esp-sa>
     <mode>TUNNEL</mode>
  </sad-entry>
  <sad-entry>  // INBOUND IPsec SA
     <spi>34502</spi>

     ...
  </sad-entry>
</sad>
```

```
$ sudo ip xfrm policy list

src 10.1.0.2/32 dst 10.0.0.2/32
   dir fwd priority 0
   tmpl src 192.168.0.2 dst 192.168.0.1
      proto esp reqid 0 mode tunnel
src 10.0.0.2/32 dst 10.1.0.2/32
   dir out priority 0
   tmpl src 192.168.0.1 dst 192.168.0.2
      proto esp reqid 0 mode tunnel
src 10.1.0.2/32 dst 10.0.0.2/32
   dir in priority 0
   tmpl src 192.168.0.2 dst 192.168.0.1
      proto esp reqid 0 mode tunnel
```

Fig. 6. Deployment example for IKE-less case.

192.168.0.1, remote/192.168.0.2).

Finally, the PAD specifies identity information (*pad-entries*) about remote IKE endpoints. In this example, an IP address represents the endpoint identity (*ipv4-address/192.168.0.2*), the authentication method (*auth-m/pre-shared*), and the shared secret (*secret/secret*).

We provide the output of the *ip xfrm state list* command to show the SAs (SAD) configured for *gateway1* (3).

9.4. IKE-less case example

Fig. 6 shows the details for *gateway1* in IKE-less case assuming, for simplicity, a proactive mode operation. As expected, the SDN controller only uses the SPD and SAD configuration elements.

In case of the SPD, as described before, the *spd-entry* element is used to provide configuration data for a single SPD policy. The SDN controller feeds the network resource with the required inbound, outbound and forward policies. According to the IPsec implementation available a Linux kernel [41], three independent policies need to be configured for inbound, outbound and forwarded traffic. For simplicity, Fig. 6 only details the content for the inbound SPD entry since the outbound and forward entries are similar. In this example, it points out the priority (*priority/0*), policy direction (*direction/INBOUND*), traffic selectors with the IP addresses of the endpoints whose traffic has to be protected (*local-addresses/10.1.0.2, remote-addresses/10.0.0.2*). It also specifies the next layer protocol (*next-layer-protocol/TCP*), the action to be applied to this traffic (*processing-info/PROTECT*), the kind of protection (*security-protocol/ESP*), IPsec mode (*mode/TUNNEL*) and tunnel endpoints (*local/192.168.0.1, remote/192.68.0.2*).

The SAD configuration element represents the configuration for an unidirectional SAD entry in the network resource (*sad-entry*). As explained in Section 2.1, IPsec requires at least two SAs (inbound and outbound) to protect the traffic exchanged between two peers, so two *sad-entry* elements are required in the example. Fig. 6 shows the configuration for the outbound SA (similar for the inbound). The example includes the SPI number (*spi/34501*), the peer tunnel endpoints (*local-addresses/192.168.0.1, remote-addresses/192.68.0.2*), the security protocol used in this example (*security-protocol/ESP*), the cryptographic algorithm (*encryption-algorithm/3des*), IV (*iv/vector*), key value (*key/ecr_secret)*, and the IPsec mode (*mode/TUNNEL*).

In this case, we provide the output of the *ip xfrm policy list* command to show the policies (SPD) configured for *gateway1* (2).

10. Conclusion and future work

In this paper, we have presented the first step to build a flexible, dynamic, scalable and standard SDN-based IPsec management solution. These features will be basic in future SDN deployments such as 5G networks, virtualization scenarios and SD-WANs where IPsec is playing a key role.

We have described the general framework, its operation and details of two deployment cases: 1) *IKE case*, where IKEv2 is implemented in the network resource and the SDN controller provides both IKEv2 credentials and IPsec policies to the SPD; 2) *IKE-less case*, where network resource only implements the IPsec logic in the kernel without IKEv2, so the SDN controller manages the SPD and the IPsec SAs, thus directly adding SAD entries. We have explained the general interface based on NETCONF/YANG and its relationship with the existing APIs that deal with IKEv2 and IPsec implementation in current kernels. We have also provided a comprehensive analysis of the proposal, including related security aspects, and developed a basic performance analysis indicating the conditions under which IKE case is better than IKE-less case, and viceversa.

Finally, we have shown a proof-of-concept implementation that proves the validity of our framework.

However, this is just a first effort and further work is required. The most urgent step will evaluate the performance deeper and the scalability of the solution in realistic SDN deployments. After integrating our framework with a consolidated SDN controller software (e.g. ONOS or OpenDayLight), we plan to quantify the impact of using our proposal considering the solutions that have recently arisen to alleviate the scalability problem in softwarized networks. Regarding the design, several aspects need further discussion like the definition of east-west and northbound interfaces or the model of high level policies at the application plane. In this sense, also analysis of *host to gateway* (*road warrior*) scenario is necessary, since it adds complexity due to the host may not be controlled by an SDN controller; the study of distributed key distribution between different SDN controllers in IKE-less case and, finally, the definition of interfaces and YANG models needs to be improved. In fact, at the time of writing, the *Interface to Network Security Functions* (I2NSF) WG [30], where this work is being discussed, is working in the standardization of these interfaces.

Declaration of Competing Interest

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.

Acknowledgment

The authors would like to thank Sowmini Varadhan, Alejandro Perez and the anonymous reviewers for their valuable comments.

Annex A. List of acronyms

Acronym	Meaning
AH	Authentication Header
API	Application Program Interface
ESP	Encapsulating Security Payload
ETSI	European Telecommunications Standards Institute
FPP	Flow-based Protection Policies
I2NSF	Interface to Network Security Functions
IETF	Internet Engineering Task Force
IKEv2	Internet Key Exchange version 2
IP	Internet Protocol
IPsec	Internet Protocol security
LXC	Linux Containers
NAT	Network Address Translation
NFV	Network Function Virtualization
NSF	Network Security Function
ONF	Open Network Foundation
PAD	Peer Authorization Database
PSK	Pre-Shared Key
RPC	Remote Procedure Call
SA	Security Association
SAD	Security Association Database
SDN	Software Defined Network
SPD	Security Policy Database
SPI	Security Policy Index

VICI	Versatile IKE Control Interface
VM	Virtual Machine
VNF	Virtual Network Function
VPN	Virtual Private Network
VPNaaS	Virtual Private Network as a Service
WAN	Wide Area Network

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.csi.2019.103357.

References

[1] Open Network Foundation, SDN architecture 1.1, TR-521, 2016.
[2] NGMN Alliance, 5g white paper, 2015.
[3] W. Xia, Y. Wen, C.H. Foh, D. Niyato, H. Xie, A survey on software-defined networking, IEEE Commun. Surv. Tutor. 17 (1) (2014), https://doi.org/10.1109/COMST.2014.2330903.
[4] ONUG, Software-defined WAN use case, 2014.
[5] S. Kent, K. Seo, Security architecture for the internet protocol, RFC 4301, (2005), https://doi.org/10.17487/RFC4301. http://www.rfc-editor.org/info/rfc4301 .
[6] S. Hares, D. Lopez, M. Zarny, C. Jacquenet, R. Kumar, J. Jeong, Interface to network security functions (i2NSF): Problem statement and use cases, RFC 8192, (2017), https://doi.org/10.17487/RFC8192. https://tools.ietf.org/html/rfc8192 .
[7] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, T. Kivinen, Internet key exchange protocol version 2 (IKEv2), RFC7296, (2014). Https://tools.ietf.org/html/rfc7296
[8] R. Marin-Lopez, G. Lopez-Millan, F. Pereniguez-Garcia, Software-defined networking (SDN)-based IPsec flow protection, draft-ietf-i2nsf-sdn-ipsec-flow-protection-04, (2019). (work in progress)
[9] K. Tran, H. Nagaraj, X. Chen, Yang data model for internet protocol security (IPsec), draft-tran-ipsecme-yang-01, (2016).
[10] K. Watsen, G. Wu, YANG groupings for TLS clients and TLS servers, draft-ietf-netconf-tls-client-server-11, (2019).
[11] K. Watsen, G. Wu, L. Xia, YANG groupings for SSH clients and SSH servers, draft-ietf-netconf-ssh-client-server-11, (2019).
[12] A. Sajassi, S. Thoria, D. Carrel, B. Weis, Secure EVPN, draft-sajassi-bess-secure-evpn-01, (2019).
[13] L. Dunbar, A. Malis, C. Jacquenet, M. Toy, Seamless interconnect underlay to cloud overlay problem statement, draft-ietf-rtgwg-net2cloud-problem-statement-00, (2019).
[14] D. Carrel, B. Weis, IPsec key exchange using a controller, draft-carrel-ipsecme-controller-ike-01, (2019).
[15] Y. Xia, S. Jiang, T. Zhou, S. Hares, Y. Zhang, NEMO (NEtwork MOdeling) language, draft-xia-sdnrg-nemo-language-04, (2016).
[16] T. Zhou, S. Liu, Y. Xia, S. Jiang, YANG data models for intent-based NEtwork MOdel, draft-zhou-netmod-intent-nemo, (2015).
[17] W. Liu, J. Strassner, G. Karagiannis, M. Klyus, J. Bi, C. Xie, SUPA policy-based management framework, draft-liu-supa-policy-based-management-framework-02, (2016).
[18] J. Strassner, J. Halpern, S. van der Meer, Generic policy information model for simplified use of policy abstractions (SUPA), draft-ietf-supa-generic-policy-info-model-03, (2017).
[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008), https://doi.org/10.1145/1355734.1355746.
[20] K. Yap, T. Huang, B. Dodson, M.S. Lam, N. McKeown, Towards software-friendly networks, Proceedings of the first ACM asia-pacific Workshop on systems (APSys '10). New York (USA), (2010).
[21] N. Foster, R. Harrison, M.J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker, Frenetic: a network programming language, Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP '11). New York (USA), (2011).
[22] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, R. Clark, Kinetic: Verifiable dynamic network control, Proceedings of 12th USENIX Symposium on Networked Systems Design and Implementation. Oakland (USA), (2015).
[23] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, Network configuration protocol (NETCONF), RFC6241, (2011). https://tools.ietf.org/html/rfc6241 .
[24] S. Chishol, H. Trevino, NETCONF event notifications, 2008, https://tools.ietf.org/html/rfc5277.
[25] P. Shafer, An architecture for network management using NETCONF and YANG, RFC6244, (2011). https://tools.ietf.org/html/rfc6244 .
[26] M. Bjorklund, YANG - a data modeling language for the network configuration protocol (NETCONF), RFC6020, (2010). https://tools.ietf.org/html/rfc6020 .
[27] D. McDonald, C. Metz, B. Phan, PF_KEY key management API, version 2, RFC 2367, (1998), https://doi.org/10.17487/RFC2367. http://www.rfc-editor.org/info/rfc2367 .
[28] Versatile IKE configuration interface (VICI), strongswan plugins, 2018, https://wiki.strongswan.org/projects/strongswan/wiki/VICI.
[29] Nemo: An applications interface to intent based networks, 2019, http://nemo-project.net/.
[30] Interface to network security functions (i2nsf) IETF WG, 2018, https://datatracker.ietf.org/wg/i2nsf/charter/.
[31] J. Katz, Y. Lindell, Introduction to Modern Cryptography, second, CRC Press, 2014. ISBN: 9781466570269
[32] Strongswan, the opensource IPsec-based VPN solution, 2018, https://www.strongswan.org.
[33] A. Huttunen, et al., UDP encapsulation of IPsec ESP packets, RFC3948, (2005). https://tools.ietf.org/html/rfc3948 .
[34] T. Pauly, et al., TCP encapsulation of IKE and IPsec packets, RFC8229, (2017). https://tools.ietf.org/html/rfc8229 .
[35] Report on SDN usage in NFV architectural framework, 2015, ETSI GS NFV-EVE 005, v1.1.1.
[36] NFV security and trust guidance, 2014, ETSI GS NFV-SEC 003, v1.1.1.
[37] Network domain security (NDS); IP network layer security (release 16), 2019, 3GPP TS 33.210, v.16.1.0.
[38] S. Sakane, PF_KEY extensions for IPsec policy management in KAME stack, 2002.
[39] K. Kaichuan, Why and how to use netlink socket, Linux J. (2005). http://www.linuxjournal.com/article/7356 .
[40] Security Foundation Requirements for SDN Controllers, 2016, Open Networking Foundation (ONF), v1.0.
[41] IPsec HOWTO, ralf spenneberg, 2003, http://www.ipsec-howto.org/ipsec-howto.pdf.
[42] R. Jahan, Software defined networking interface (SDNi), 2019, https://docs.opendaylight.org/en/stable-fluorine/developer-guide/odl-sdni-developer-guide.html.
[43] Gartner, Predicting SD-WAN adotion, 2015, https://blogs.gartner.com/andrew-lerner/2015/12/15/predicting-sd-wan-adoption/.
[44] Packet Pushers, List of SD-WAN vendors, 2019, https://packetpushers.net/virtual-toolbox/list-sd-wan-vendors/.
[45] A. Ranjbar, M. Komu, P. Salmela, T. Aura, An SDN-based approach to enhance the end-to-end security: SSL/TLS case study, NOMS 2016 -2016 IEEE/IFIP Network Operations and Management Symposium, (2016), https://doi.org/10.1109/NOMS.2016.7502823.
[46] M. Vajaranta, J. Kannisto, J. Harju, Implementation experiences and design challenges for resilient SDN based secure WAN overlays, 2016 11th Asia Joint Conference on Information Security, (2016), https://doi.org/10.1109/AsiaJCIS.2016.25.
[47] I. Ahmad, T. Kumary, M. Liyanage, J. Okwuibex, M. Ylianttila, A. Gurtov, 5g security: Analysis of threads and solutions, 2017 IEEE Conference on Standards for Communications and Networking (CSCN), (2017). DOI: 10.1109CSCN.2017.8088621
[48] J. Yang, J. Jeong, J. Kim, Security policy translation in interface to network security functions, Draft-yang-i2nsf-security-policy-translation-03, (2019). (work in progress)
[49] J. Jeong, E. Kim, T. Ahn, R. Kumar, S. Hares, I2NSF consumer-facing interface YANG data model, Draft-ietf-i2nsf-consumer-facing-interface-dm-03, (2019). (work in progress)
[50] Open Network Foundation Working Group, Northbound interfaces, 2013, https://www.opennetworking.org/images/stories/downloads/working-groups/charter-nbi.pdf.
[51] F. Benamrane, B. Redouane, M.B. Mamoun, An east-west interface for distributed SDN control plane: implementation and evaluation, Comput. Electr. Eng. 57 (2017) 162–175, https://doi.org/10.1016/j.compeleceng.2016.09.012.
[52] H. Yu, K. Li, H. Qi, W. Li, X. Tao, Zebra: An east-west control framework for SDN controllers, 44th International Conference on Parallel Processing, (2015), https://doi.org/10.1109/ICPP.2015.70.
[53] E. Barker, Recommendation for key management, NIST Special Publication 800-57 Part 1, Revision 4, (2016).
[54] S.K. Fayazbakhsh, V. Sekar, M. Yu, J.C. Mogul, Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions, Second ACM SIGCOMM workshop on Hot topics in Software Defined Networking, (2013), https://doi.org/10.1145/2491185.2491203.
[55] A. Bremler-Barr, Y. Harchol, D. Hay, Y. Koral, Deep packet inspection as a service, 10th ACM International on Conference on Emerging Networking Experiments and Technologies, (2014), https://doi.org/10.1145/2674005.2674984.
[56] V.H.F. Tafreshi, E. Ghazisaeedi, H. Cruickshank, Z. Sun, Integrating IPsec within openflow architecture for secure group communication ZTE communications, volume 1, 2014, Pages 41–49. DOI: 3939/j.issn.1673-5188.2014.02.007.
[57] W. Li, F. Lin, G. Sun, SDIG: Toward software-defined IPsec gateway, IEEE 24th International Conference on Network Protocols (ICNP), (2016), https://doi.org/10.1109/ICNP.2016.7785316.
[58] M. Vajaranta, J. Kannisto, J. Harju, IPsec and IKE as functions in SDN controlled network, International Conference on Network and System Security NSS 2017, LNCS 10394, (2017), pp. 521–530, https://doi.org/10.1007/978-3-319-64701-2_39.