# Estudio de la plataforma ONOS para la Gestión de asociaciones de seguridad IPsec en entornos SDN


Septiembre 2018

Trabajo presentado por Valiantsin Kivachuk para la obtención del Grado en Informática en la Universidad de Murcia

---

Supervisado por:

Rafael Marín López

Gabriel López Millán

# Declaración firmada sobre la originalidad del trabajo

D. Valentín KIVACHUK, estudiante de la titu-lación de Grado en Ingeniería Informática de la Universidad de Murcia y autor del TFG titulado "*Estudio de la plataforma ONOS para la Gestión de asociaciones de seguridad IPsec en entornos SDN*".

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015 y modificado 22-04-2016), así como la normativa interna para la oferta, asignación, elaboración y defensa de los Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial.

Murcia, a 5 de septiembre de 2018

Fdo.:
Autor del TFG

# Índice general

**Resumen** **8**

**Extended Abstract** **9**

**1. Introducción** **13**

**2. Estado del Arte** **15**
2.1. Software Defined Networking SDN . . . . . 15
2.1.1. Motivación . . . . . 15
2.1.2. Arquitectura . . . . . 16
2.1.3. Interfaces de comunicación . . . . . 17
2.1.4. Escenario de uso . . . . . 19
2.2. Network Configuration Protocol (NETCONF) . . . . . 20
2.2.1. Definición . . . . . 20
2.2.2. Arquitectura . . . . . 20
2.2.3. Modelo de mensajes (*Message layer*) . . . . . 20
2.2.4. Modelo de operaciones (*Operations layer*) . . . . . 22
2.2.5. Modelo de datos (*Content Layer*) . . . . . 23
2.3. Internet Protocol Security (IPsec) . . . . . 25
2.3.1. Definición . . . . . 25
2.3.2. Arquitectura . . . . . 25
2.3.3. Modos de funcionamiento . . . . . 26
2.3.4. Procesamiento de tráfico . . . . . 27
2.3.5. Internet Key Exchange (IKE) . . . . . 28
2.4. Gestión de IPsec SAs utilizando SDN . . . . . 29
2.4.1. Caso 1 . . . . . 30
2.4.2. Caso 2 . . . . . 31

**3. Objetivos y Metodología Empleada** **32**

**4. Solución SDN** **34**
4.1. Estudio de ONOS . . . . . 34
4.1.1. Arquitectura . . . . . 34
4.1.2. Análisis de otras alternativas . . . . . 37
4.2. Diseño escenario SDN . . . . . 40
4.2.1. Requisitos de escenario . . . . . 40

4.2.2. Comunicación Southbound . . . . . 42
4.2.3. Comunicación Northbound . . . . . 43
4.2.4. App ONOS . . . . . 44
4.2.5. Modelo y configuración ietf-ipsec . . . . . 47
4.2.6. Flujo de mensajes . . . . . 49

**5. Desarrollo e implementación** **52**
5.1. ONOS SDK . . . . . 54
5.2. Despliegue del escenario . . . . . 56
5.3. Prueba del escenario . . . . . 57

**6. Conclusión y vías futuras** **60**

**Siglas** **63**

**Glosario** **64**

**Bibliografía** **69**

**A. Proyecto onos-ipsec** **70**

**B. Estructura ietf-ipsec** **71**

**C. Configuración NETCONF basada en ietf-ipsec** **76**

**D. Docker** **80**

# Índice de tablas

4.1. Comparativa de distintas soluciones SDN. . . . . 39

# Índice de figuras

2.1. Diferencia modelo tradicional frente al SDN. . . . . 17
2.2. Arquitectura SDN. . . . . 18
2.3. Escenario de uso de una arquitectura SDN. . . . . 19
2.4. Arquitectura NETCONF. . . . . 21

4.1. Subsistemas de ONOS [1]. . . . . 35
4.2. Arquitectura del subsistema de ONOS [1]. . . . . 36
4.3. Escenario SDN propuesto. . . . . 41
4.4. Arquitectura de la app de ONOS. . . . . 45
4.5. Flujo de mensajes en el escenario propuesto. . . . . 50

5.1. Topología del escenario propuesto. . . . . 53
5.2. Traza de solicitud a ONOS al arrancar el dispositivo. . . . . 57
5.3. Intercambio protegido de mensajes Echo Request. . . . . 59

D.1. Logo Docker. [2] . . . . . 80

# Índice de códigos


1. Ejemplo solicitud RPC. . . . . 21
2. Respuesta RPC a la solicitud del Código 1. . . . . 22
3. Ejemplo modelo YANG. . . . . 24
4. Modelo YIN. . . . . 24
5. Dar de alta un dispositivo NETCONF en ONOS. . . . . 44
6. Sesión NETCONF de un dispositivo. . . . . 46
7. Esqueleto app ONOS. . . . . 47
8. Filtro NETCONF para la IP del plano de datos. . . . . 51
9. Carga módulos al arrancar ONOS. . . . . 54
10. Método que ONOS llama cuando se activa la app. . . . . 54
11. Obteneción de una sesión NETCONF del dispositivo. . . . . 55
12. Abstracción de Docker para desplegar el escenario. . . . . 56
13. Fragmento del registro de Netopeer. . . . . 58
14. Docker compose del proyecto onos-ipsec . . . . . 82

# Resumen

Desde hace años las redes tradicionales no están capacitadas para satisfacer la gran demanda por parte de millones de usuarios: ancho de banda, velocidad, latencia, calidad de servicio, etc. La complejidad para gestionar las redes y asegurar esos requisitos ha ido incrementandose a la par que su uso masivo. El panorama ha evidenciado que los dispositivos clásicos carecen de la flexibilidad necesaria para adaptarse en tiempo real a la imprevisible demanda.

Esto impulsa la necesidad de enfocar el problema de las redes modernas desde una perspectiva diferente. Software Defined Networking (SDN) [3] es un paradigma que pretende revolucionar el ámbito de las redes. Facilita la implementación e implantación de servicios de red, evitando al administrador gestionarlo directamente. Lejos de ser un concepto nuevo y poco desarrollado, gran cantidad de organizaciones e investigadores centran sus esfuerzos desde hace años en investigar su potencial y definir diversos ámbitos en los que puede ser aplicado.

La línea de investigación por la que se rige este Trabajo Fin de Grado está desarrollando la estandarización de un escenario que habilite la gestión de asociaciones de seguridad en IPsec en entornos SDN. Estudiar el potencial y las capacidades del software de libre distribución ONOS para ser integrado como Controlador SDN en ese escenario es el cometido de este documento.

Un análisis de la plataforma de código libre ONOS [4] y sus características ha concluido en el desarrollo de la solución. Crear un prototipo de aplicación que establezca automáticamente las asociaciones de seguridad IPsec [5] en entornos SDN permitirá demostrar la gran flexibilidad y adaptación que tiene un paradigma SDN. No obstante, la puesta en marcha de un prototipo funcional no es más que una pequeña demostración del potencial de un entorno SDN.

# Extended Abstract

For several decades, networks management was a hard work. In addition, the use of networks worldwide grows overwhelmingly, registering traffic of 1.2 ZB in 2016 and future estimations of 3.3 ZB for 2021 [6]. This has shown that classic devices lack the necessary flexibility to adapt in real time to unpredictable demand.

This demand drives the need to solve the problem of modern networks from a different perspective. Software Defined Networking (SDN) [3] is a paradigm that aims to revolutionize this field. It makes easier the implementation and implantation of network services, avoiding the administrator to manage them directly.

Traditional networks are based on *data plane* (packet switching) and *control plane* (traffic routing). The SDN paradigm performs a physical separation, shifting the functionality of the control plane to an entity called Controller. Decoupling the functionality to different planes, this model converts the Controller into a centralized management point, which manages dynamically the network domain.

The SDN environment consists of different entities, which are grouped into logical interfaces. The Southbound API is the communication interface between the Controller and the network devices, using protocols such as NETCONF [7] or Openflow [8]. The Northbound API offers an interface to the different applications and services that manage the network through the Controller, such as a REST API, a graphical interface (GUI) or through a command line (CLI), among others. If implemented, the Eastbound/Westbound API is used in distributed SDN controller schemes to allow communication between them and sharing information.

A common example of scenario to visualize the potential of an SDN environment consists of an SDN Controller and two switches. The controller, in the control plane, satisfies the configuration of the switches (the data plane) so that traffic can transit through the network. The process is carried out automatically following the established policies without the intervention of an administrator.

The object of study of this work is very similar to that example, but establishing IPsec [5] security associations into network devices via NETCONF,

based on security policies previously defined by ONOS open source distribution software [4]. Specifically, the study analyze the possibilities this SDN Controller offers and creates an application prototype (*app*) for that purpose. It is necessary to analyze the ONOS architecture, the possibilities it offers to develop on this platform and design and implement a software prototype that, through ONOS, performs the automatic management of IPsec security associations.

However, it is essential to carry out a thorough analysis of two crucial technologies in our work: NETCONF and IPsec.

NETCONF is a protocol that allows to install, manipulate and eliminate the configuration of network devices. Using Remote Procedure Call (RPC), the configuration of a network device is edited and consulted, whose data is represented in XML format [9]. Using the client-server model, the client executes the RPCs on a server to configure them. Said communication is established through the use of a session, which will have a series of parameters according to the functionalities (capabilities) supported by the network device.

However, in order to be able to operate with NETCONF servers from a client, it is essential to use a common model that specifies the type of data and how it is structured. YANG (Yet Another Next Generation) [10] is a modeling language that organizes data hierarchically, using a tree structure where each node can contain a set of child nodes (container) or a name and value. Using this language allows both the client and the NETCONF server to validate the messages exchanged. To do this, a conversion of the defined model to an XML representation, also known as YANG Independent Notation (YIN), is performed internally.

On the other hand, Internet Protocol Security (IPsec) [5] is a set of mechanisms to secure IP (Internet Protocol) communications through the authentication and encryption of each packet of the data flow. IPsec also includes protocols to establish mutual authentication between agents at the beginning of a session and negotiate cryptographic keys that will be used in it. Maintaining the confidentiality, authentication and integrity at the network level of the OSI model implies the protection of all the upper layers, regardless of the protocol used in it.

IPsec is essentially divided into 3 areas: authentication, confidentiality and key management. These elements are managed through two non-exclusive protocols. The IP extension header that gives authenticity and integrity to the packets is called Authentication Header (AH), which using a secret shared by both ends of the communication, generates a Hash Message Authentication Code (HMAC) of the contents of the IP packet and some immutable parts. Encapsulating Security Payload (ESP) in addition to the features described in AH, enables confidentiality. To do this, encrypt the IP packet completely and encapsulate it in another or add an encryption header to

the original package, depending on the operating mode in which it operates. These modes can be: tunnel mode or transport mode. Both AH and ESP are able to work with both modes, manipulating and encapsulating the packets in a particular way in each case.

Internally, IPsec manages all the security associations and the mechanisms associated with them with a set of structures: the SAD (Security Association Database) stores the parameters for each one of the established SAs. Each entry consists of a unique identifier (SPI), the protocol responsible for protecting traffic (ESP or AH), the mode, algorithms and authentication keys and / or encryption among others. The SPD (Security Policy Database) allows to know what to do with a given package (apply an SA or not), because every subset of IPs has an action (Protect, Discard or Bypass). Finally, the PAD (Peer Authorization Database) stores the information necessary to perform authentication between two devices. It acts as a link between the SPD and a protocol for the management of security associations, such as IKE [11]. Usually there is confusion between the terms IKE (Internet Key Exchange) [11] and IPsec, which despite being related, are independent concepts. IKE is a protocol for management and distribution of Security Associations, used by IPsec. The latest version uses an exchange of cryptographic keys of the Diffie-Hellman type to establish a shared secret between both ends of the communication. Based on this, the entities create a security association called IKE SA, which will be used to securely negotiate the SAs that IPsec will use to protect the data traffic, also known as CHILD SA.

Currently, the research line by which this Final Degree Project is governed is developing the standardization of a scenario that enables the management of security associations in IPsec in SDN environments. To do this, two use cases have been defined, case 1 and case 2. Both provide mechanisms to distribute and configure IPsec both in simple scenarios (IPsec tunnel between devices in the same network) and in more complex (devices separated by one or more gateways).

In case 1, it is assumed that the devices support both IPsec and the administration of security associations through IKE. In this scenario, the Controller manages both IPsec (SPD and PAD) and the IKE configuration. A security association is established by negotiating the IKEv2 protocol. On the other hand, case 2 does not use IKE as a mechanism to negotiate the different parameters that allow the establishment of security associations, since it is the Controller himself who will be responsible for managing the IPsec SAs through the monitoring and administration of the SPD and SAD.

The main objective of the project is the study of free ONOS distribution software to be integrated as an SDN Controller in a scenario of automatic management of IPsec security policies in network devices through the NETCONF protocol. It is necessary to analyze the architecture of ONOS, the various possibilities it offers to develop on this platform and design and im-

plement a software prototype that, through ONOS, performs the automatic management of IPsec security associations

The development solution is based on Case 2, where a design composed of an ONOS SDN Controller and $n$ network devices has been proposed. The communication between the Controller and the devices belonging to the control plane will be made using the NETCONF protocol and the YANG *ietf-ipsec* data model, while the traffic flow between the devices belonging to the data plane will be encrypted and authenticated with each of the other network devices.

In order that ONOS Controller could manage the IPsec configuration of the network devices, it has been necessary to develop the logic in an application (*app*), which will subscribe to the events generated by ONOS through the component *NetopeerListener*. This will allow us to execute actions when a NETCONF device connects to the Controller and the generation of NETCONF messages will be done directly from the app.

An analysis of the ONOS open source platform and its features have resulted in the development of the project. We can conclude that it is a mature platform, with a wide support of different forms of interaction with the Controller, either from the Northbound API (GUI, CLI and API REST) or the Southbound API, such as OpenFlow, NETCONF or BGP [12].

Nevertheless, one of the main shortcomings was the documentation regarding the interpretation and generation of NETCONF messages through YANG models with the ONOS SDK.

However, this work provides a solution to the need for a Controller that automates the management of IPsec security policies in SDN environments. The development of this work not only meets the objective, but also provides a flexible and dynamic implementation of the entire scenario proposed for case 2 of the research line. And despite everything, it is only a small demonstration of the potential of an SDN environment for problems such as the management of IPsec security associations in a complex network.

# Capítulo 1

# Introducción

El uso de las redes de comunicación a nivel mundial crece de forma abrumadora. Se estima que el tráfico IP anual en todo el mundo alcanzará un volumen de 3.3 ZB[1] en 2021, frente a los 1.2 ZB registrados en 2016 [6]. Para hacernos una idea, 3.3 ZB podrían ser distribuidos en porciones de aproximadamente 30 GB[2] de información entre todos los *Homo sapiens* de la historia de la humanidad hasta la fecha [13]. Este constante incremento introduce una gran complejidad para gestionar y administrar las redes de comunicación, pero, además, se espera mantener o mejorar los servicios que éstas brindan: el ancho de banda, la latencia o la priorización de determinados flujos de datos para asegurar una calidad de servicio, entre otros.

Este objetivo, aparentemente imposible, ha evidenciado con el paso de los años que la arquitectura tradicional de las redes de comunicaciones limitan dicho objetivo: los dispositivos clásicos carecen de la flexibilidad necesaria para adaptarse en tiempo real a la imprevisible demanda. Esta necesidad ha impulsado la creación de nuevas técnicas y revolucionarios paradigmas, como es el caso del Software Defined Networking (SDN) [3].

Un entorno SDN no es más que el desacople de los distintos planos lógicos por los que está compuesto un dispositivo de red tradicional: el plano de control y el plano de datos. La entidad *Controlador* se encargará de tomar las decisiones de enrutamiento de tráfico, correspondiente al plano de control, mientras que el switch realizará la conmutación de paquetes en base a las instrucciones proporcionadas por dicho Controlador, pertenecientes al plano de datos. El uso más común de las soluciones de Controlador, tipo ONOS o OpenDayLight, es para el control de tramas y paquetes IP que tienen que atravesar una red. Es decir, la gestión de tráfico de la capas 2 (enlace) y 3 (red) del modelo OSI.

Aunque un entorno SDN puede ser aplicado en una infinidad de escenarios

[1] 1 zettabyte = $10^{21}$ bytes
[2] 1 gigabyte = $10^{9}$ bytes

para distintos propósitos, este trabajo es fruto de una línea de investigación previamente definida: el establecimiento y la aplicación automática de políticas de seguridad IPsec en entornos SDN [14]. Con ello se pretende suplir una falta de experiencia y puesta en marcha de un Controlador para su gestión, creando una referencia que sirva de estándar para las redes definidas por software (SDN).

Su aplicación se han definido en dos casos de uso distintos, caso 1 y caso 2. La gestión de asociaciones de seguridad IPsec en el caso 1 se delega en el protocolo IKEv2, mientras que en el caso 2 es el controlador quién se encarga de administrarlo completamente.

No obstante, las pruebas realizadas hasta ahora simulaban el uso de un controlador, pero no se habían hecho pruebas con soluciones reales. La carencia de un *Controlador* SDN que automatice la gestión de políticas de seguridad en los distintos escenarios que plantea dicha línea de investigación, ha motivado dos trabajos: un estudio del potencial de ONOS, que concierne a este trabajo, y de OpenDaylight.

Como en otro trabajo se ha realizado el análisis de OpenDayLight [15], éste se centrará en la plataforma de ONOS con el mismo propósito. Para ello, será necesario hacer uso de NETCONF [7] y modelos YANG [10], que es dónde nos centraremos dentro de la extensión de ONOS.

ONOS ha demostrado, mediante este trabajo, ser una solución perfectamente válida para desarrollo y despliegue en escenarios reales. Su estructura y mecanismos de integracion no son intuitivos a primera vista y se ha requierido de un profundo aprendizaje y práctica para utilizarlo en el marco de nuestro trabajo.

El análisis de la plataforma y sus características han concluido en el desarrollo de la solución. Las capacidades y el potencial que ONOS es capaz de desplegar están a la altura de una solución íntegra para gestionar políticas de seguridad IPsec en entornos SDN [3].

Relativo a este documento, el conjunto de los capítulos a continuación pretenden reflejar todos los conocimientos adquiridos en la realización de este trabajo y justificar las decisiones tomadas. La organización sigue el siguiente esquema: el capítulo 2 analiza las tecnologías, técnicas y protocolos de suma relevancia para nuestro proyecto; en el capítulo 3 se establecen los objetivos para la finalización de nuestro trabajo y la metodología utilizada; el capítulo 4 analiza ONOS en el contexto de nuestro trabajo, define el diseño y las directrices a seguir para crear una solución; el capítulo 5 se centra en el desarrollo y puesta en marcha de la solución. Finalmente, el capítulo 6 reflexiona sobre el trabajo realizado y las conclusiones obtenidas, junto al planteamiento de las posibles futuras vías que quedan abiertas por realizar.

# Capítulo 2

# Estado del Arte

## 2.1. Software Defined Networking SDN

SDN ha irrumpido en el mundo de las comunicaciones como un nuevo paradigma capaz de dotar de mayor flexibilidad y robustez en complejos escenarios. Constituye una tendencia a nivel de investigación y desarrollo en las redes de comunicación. En este apartado se explicará los aspectos más importantes para entender esta tecnología: la motivación para definir este nuevo paradigma, su arquitectura y cómo es su funcionamiento.

### 2.1.1. Motivación

Desde hace varias décadas, administrar las redes de comunicaciones era sinónimo de un trabajo costoso. Todos los routers y switches implicados en el proceso tenían que ser configurados manualmente, uno por uno, para satisfacer las necesidades. Y conforme la infraestructura escalaba de forma vertical u horizontal, realizar modificaciones penalizaba proporcionalmente en tiempo o disponibilidad del servicio.

Este aspecto se acentúa con los drásticos cambios en las redes de comunicaciones de las últimas décadas, lo que fuerza a reconsiderar la arquitectura de las redes tradicionales [16]. Entre los factores más influyentes, podemos destacar:

- **Incremento de la demanda**. Conceptos como *Cloud computing*, *Big data*, *The Internet of Things (IoT)* o el auge de los dispositivos móviles como una extensión más de nosotros, han impulsado el uso masivo de las infraestructuras de redes.
- **Patrones de tráfico complejos**. El tráfico se vuelve mucho más complejo y dinámico, por lo que resulta muy difícil lidiar con la congestión.

Esto es producto de los continuos avances que podemos encontrar en grandes infraestructuras de redes. Algunos ejemplos:

- La transferencia de voz, datos y vídeo generan tráfico muy difícil de predecir.
- Aplicaciones cliente/servidor requieren comunicarse entre ellos (horizontal) y con otros clientes/servidores (vertical).
- La popularización de los servicios en la nube han desplazado gran cantidad de tráfico, comunmente local, hacia el exterior.
- La Calidad de Servicio, *QoS*, requiere pensar en *flujos* de tráfico, algo que choca con los *datagramas* de las redes tradicionales.

- **Políticas inconsistentes**. Una política de red requiere administrar dispositivos mediante el uso de distintos mecanismos y herramientas. Un cambio crítico en determinadas infraestructuras puede consumir días para reconfigurar las Access Control List (ACL) en toda la infraestructura.
- **Dependecia del fabricante**. Las soluciones a los diversos problemas implican depender de un proveedor y sus equipos, quedando limitado a los ciclos de productos relativamente lentos de los equipos de los proveedores.

Por todos estos motivos, crece la necesidad de enfocar el problema de las redes modernas desde una perspectiva diferente. Software Defined Networking (SDN) [3] es un paradigma que pretende revolucionar este ámbito, facilitando la implementación e implantación de servicios de red de una manera determinista, dinámica y escalable, evitando al administrador de red gestionar dichos servicios a bajo nivel.

Este paradigma de las redes cobra aun más importancia a partir del 2012, con la Introducción de Network Function Virtualization (NFV) [17] y la necesidad de arquitecturas más flexibles sin perder los niveles de servicio y exigencia.

### 2.1.2. Arquitectura

Desde sus inicios, los dispositivos de red administran dos planos lógicos: el *plano de datos* y el *plano de control*. El plano de datos se encarga de la conmutación de paquetes, mientras que el plano de control decide sobre el enrutamiento del tráfico.

A diferencia de las redes tradicionales, SDN realiza una separación física, desplazando la funcionalidad del *plano de control* a una entidad denominada *Controlador*. Desacoplando la funcionalidad de los distintos planos, este

modelo convierte al *Controlador* en un punto de gestión centralizado, capaz de gestionar de forma dinámica el dominio de la red sobre la que opera.

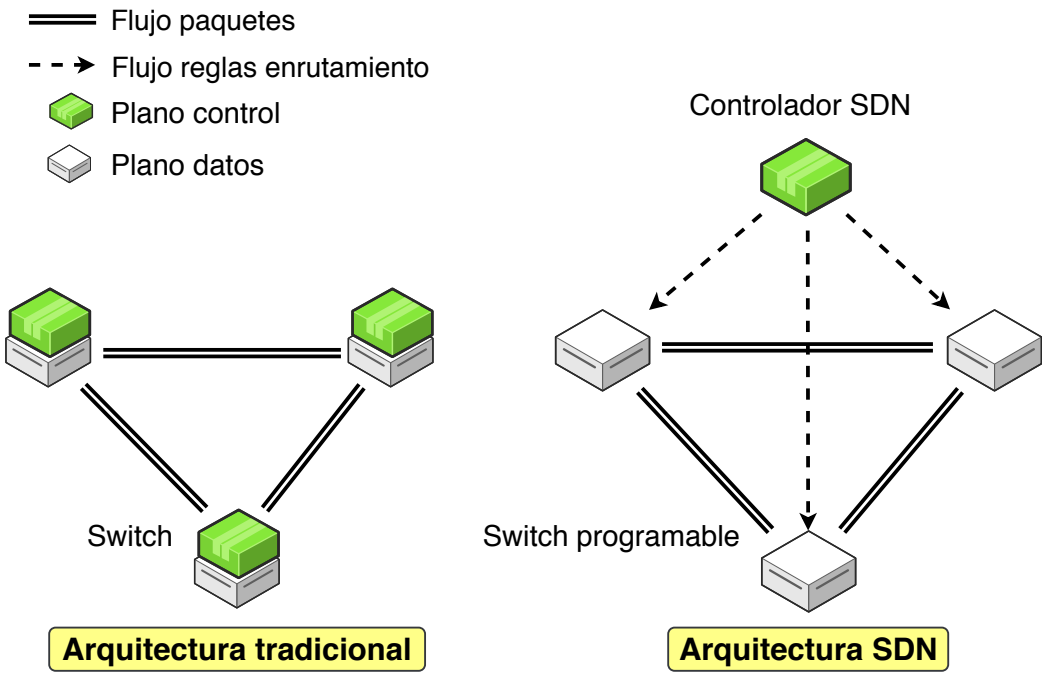


Figura 2.1: Diferencia modelo tradicional frente al SDN.

En el **plano de datos**, son los dispositivos de red (habitualmente, los switch o routers) quienes realizan las acciones de encaminamiento, fragmentación, reensamblado de paquetes y protección de flujos de tráfico.

En el **plano de control** existen uno o varios *Controladores*, quienes toman decisiones sobre el flujo de tráfico de los distintos dispositivos de red. Esta entidad tiene pleno conocimiento de la red.

Por último, en el nivel más alto del modelo SDN, surge el **plano de aplicación**. Ofrece al administrador de red una sencilla interfaz, constituida por distintos servicios y aplicaciones, con los que programar el comportamiento de la red y gestionar sus recursos.

### 2.1.3. Interfaces de comunicación

Las distintas capas que conforman la estructura del Software Defined Networking requieren distintas interfaces de comunicación. En la Figura 2.2 podemos ver un esquema general de todas las interfaces en un escenario SDN.

**Southbound API** constituye el API de comunicación entre el *Controlador* y los dispositivos de red. A pesar de que el protocolo más extendido es OpenFlow [8], a propuesta de la Open Network Foundation (ONF) [18],

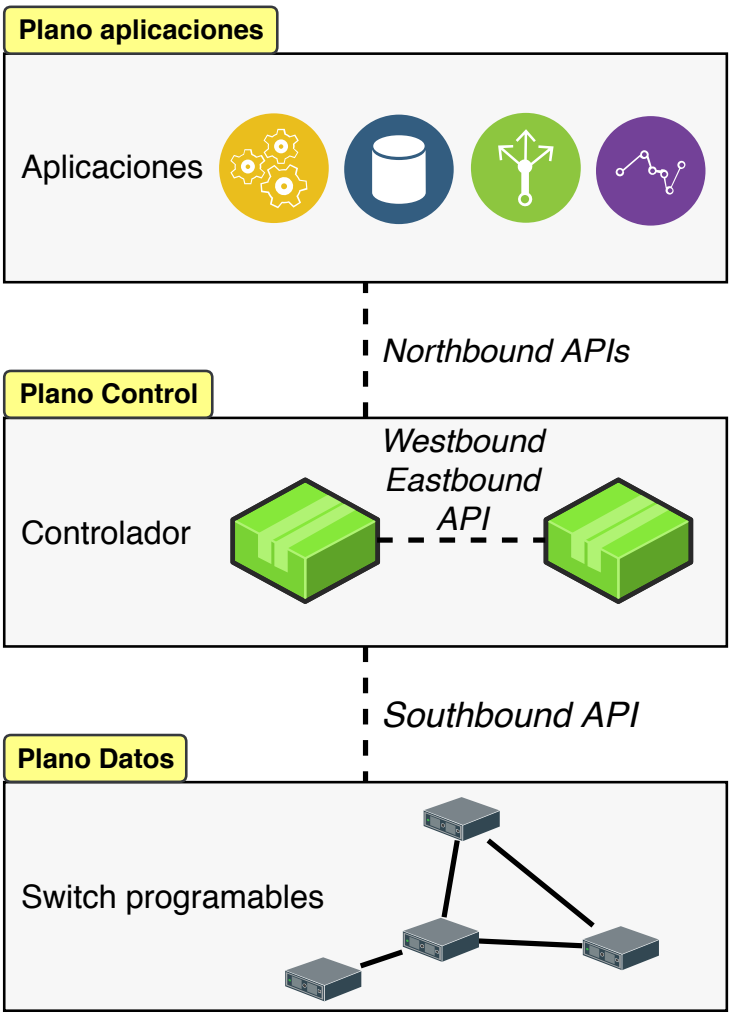


Figura 2.2: Arquitectura SDN.

existen APIs abiertas y también propietarias. Entre los estándares abiertos, podemos encontrar NETCONF [7], OSPF [19], MPLS [20], BGP [12] o OVSDB [21].

**Eastbound/Westbound API** se utiliza en esquemas de controladores SDN distribuidos para permitir la comunicación entre ellos y compartir información. Para conseguir la compatibilidad e interoperabilidad entre los controladores, se requieres protocolos estandarizados, como SDNi [22], PCEP [23] o BGP.

Por último, **Northbound API** ofrece una interfaz a las distintas aplicaciones y servicios que gestionan la red (la variedad es inmensa: visualización de toplogías, gestión de rutas, análisis de flujos, etc). Dependiendo del software SDN, podemos utilizar dicha interfaz a través de una REST API [24], sistemas de archivos, lenguajes de programación o implementación propias.

### 2.1.4. Escenario de uso

Tras describir los componentes, interfaces y la interacciones que conforman un escenario SDN, ilustraremos un escenario de uso real. En la Figura 2.3 se muestra un escenario básico que consta de un Controlador SDN y dos switch. El Controlador, en el plano de control, debe satisfacer la configuración de los switch, en el plano de datos, para que el tráfico pueda transitar a través de la red.

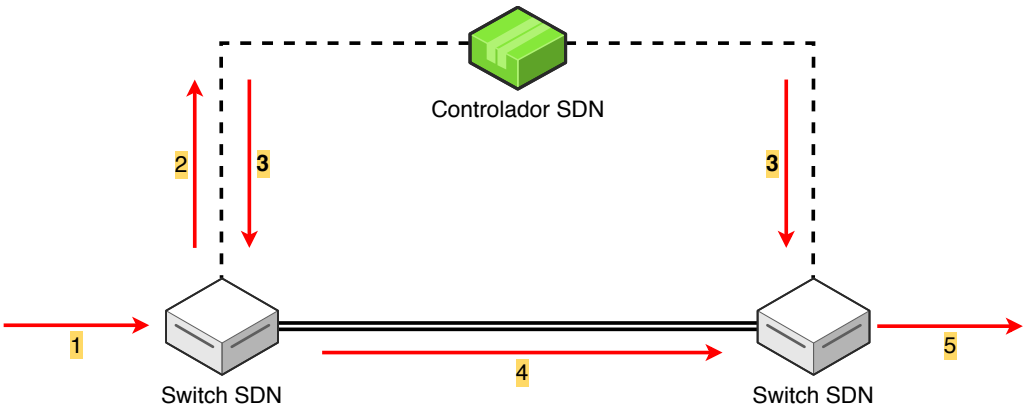


Figura 2.3: Escenario de uso de una arquitectura SDN.

Nuestro escenario de la Figura 2.3 muestra cómo se realizaría *forwarding* de un paquete que llega a la red:

1. Cuando un paquete llega al switch, éste comprueba la acción que debe tomar en base a las reglas que tiene definidas para distintos flujos de datos. Tomará una de las siguientes opciones:
   - Descartar y dar por finalizado el procesamiento.
   - Realizar *forwarding* hacia otro switch.
   - Enviarlo al Controlador para que defina la acción a realizar por el switch.
2. El switch, al recibir un paquete y no tener ninguna política de gestión asociada a ese tipo de tráfico, detiene ese flujo y reenvía el paquete al Controlador. El Controlador tendrá previamente definido el comportamiento deseado.
3. Al recibir el paquete, el Controlador establece una serie de parámetros a todos los switch que considere para que ese tipo de tráfico fluya.
4. Una vez recibida la política de gestión para ese tipo de tráfico, el switch puede desbloquear ese paquete y realizar el *forwarding* hacia el siguiente switch. Para los próximos paquetes de ese tipo, no necesitará consultar más veces al Controlador la acción a tomar.

5. De igual modo, el switch que recibe el paquete dispone de unas políticas ya establecidas para ese tráfico de datos, por lo que realizará el *forwarding* sin consultar al Controlador.

## 2.2. Network Configuration Protocol (NETCONF)

### 2.2.1. Definición

Definido por el IETF [25], NETCONF es un protocolo que permite instalar, manipular y eliminar la configuración de dispositivos de red. Utilizando Remote Procedure Call (RPC), se edita y consulta la configuración de un dispositivo de red, cuyos datos se representan en formato XML [9].

Bajo el modelo cliente-servidor, el cliente es quien ejecuta los RPCs sobre un servidor para configurarlos. Dicha comunicación se establece mediante el uso de una *sesión*, que tendrá una serie de parámetros según las funcionalidades (*capabilities*) que soporta el dispositivo de red.

### 2.2.2. Arquitectura

La arquitectura está organizada en 4 capas: *contenido*, *operaciones*, *mensajes* y *transporte seguro*. Tal como ilustra la Figura 2.4, los aspectos más importantes de cada capa son:

1. **Content layer.** Es el nivel más alto del protocolo. Contiene la información que transmitirá el cliente NETCONF a los servidores.
2. **Operations layer.** Define el modelo de operaciones que el cliente NETCONF puede utilizar. Éstas son invocadas mediante llamadas remotas (RPC) en formato XML.
3. **Message layer.** Constituye el nivel más bajo de un mensaje NETCONF. En esta capa se definen las llamadas remotas (RPC) que se utilizarán entre las entidades NETCONF.
4. **Secure Transport layer.** Se trata de la capa más baja del protocolo NETCONF. Delega en otros protocolos (como TLS o SSH) para garantizar el intercambio seguro de los mensajes NETCONF.

### 2.2.3. Modelo de mensajes (*Message layer*)

Un programa puede utilizar una llamada a procedimiento remoto (RPC) para solicitar algún servicio de un equipo remoto a través de la red, sin conocer las tecnologías de red subyacentes.

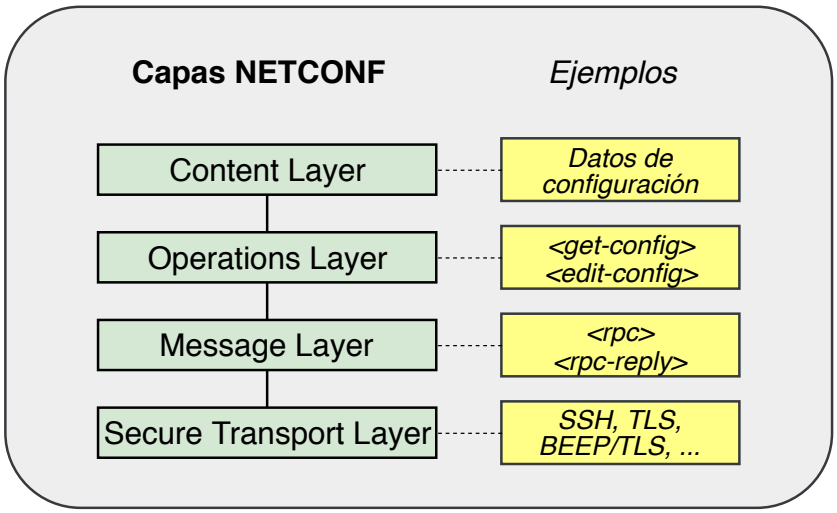


Figura 2.4: Arquitectura NETCONF.

NETCONF utiliza una comunicación basada en el modelo RPC, mediante el intercambio de elementos <rpc> y <rpc-reply> para solicitudes y respuestas independientemente del protocolo de transporte que se esté utilizando.

### Elemento <rpc>

El elemento <rpc> encapsula un mensaje de solicitud al servidor NETCONF por parte del cliente NETCONF. El formato de un mensaje RPC se ilustra en el fragmento de Código 1:

### Elemento <rpc-reply>

Un mensaje <rpc-reply> encapsula una respuesta a la solicitud <rpc>. En el Código 1 se invoca la operación <get> del atributo used-id.

```
1 <rpc message-id="101"
2      xmlns="urn:idtf:params:xml:ns:netconf:base:1.0">
3      xmlns:op="http://example.net/content/1.1"
4      op:user-id="bob"
5     <get/>
6 </rpc>
```

Código 1: Ejemplo solicitud RPC.

A lo que se obtiene una respuesta <rpc-reply>, representada en el Código 2. Es importante remarcar que en la respuesta aparecen todos los atributos de la solicitud.

```xml
<rpc-reply message-id="101"
     xmlns="urn:idtf:params:xml:ns:netconf:base:1.0">
     xmlns:op="http://example.net/content/1.1"
     op:user-id="bob"
    <data>
       <!-- Contenido -->
    <data>
</rpc-reply>
```

Código 2: Respuesta RPC a la solicitud del Código 1.

### 2.2.4. Modelo de operaciones (*Operations layer*)

La capa de operaciones es el núcleo del protocolo NETCONF. Se define el conjunto de operaciones para manejar las configuraciones de un dispositivo, una vez que se ha establecido una sesión con él.

Las operaciones que un dispositivo soporta varían en función de sus *capabilities*, que indican las operaciones que implementa. Sin embargo, todas las posibles operaciones NETCONF parten de un conjunto mínimo:

- <*get-config*>: Solicita la configuración indicada en el parámetro o como parte del mismo.
- <*edit-config*>: Modifica los parámetros de una configuración con los datos que se incluyen en la operación. NETCONF habilita mecanismos para indicar qué hacer con la configuración previa.
- <*copy-config*>: Copia el contenido de una configuración en otra del mismo tipo. Se explicará el concepto de configuración en la sección 2.2.5.
- <*delete-config*>: Indica mediante parámetro una configuración a eliminar.
- <*lock*>: Bloquea el uso de una configuración para cualquier otro usuario. Permite evitar conflictos de concurrencia.
- <*unlock*>: Desbloquea el uso de una configuración. Los usuarios sólo pueden desbloquear aquellas que hayan sido bloqueadas previamente por ellos mismos.
- <*close-session*>: Cerrar una sesión NETCONF.
- <*kill-session*>: Fuerza a cerrar una sesión NETCONF. Solo el administrador puede realizar esta operación.

### 2.2.5. Modelo de datos (*Content Layer*)

Los dispositivos de red que trabajan con el protocolo NETCONF almacenan sus configuraciones en un concepto lógico denominado *datastore*, sobre el que aplican las operaciones.

La configuración actual de un dispositivo se almacena en el datastore <*running*>. Dicha configuración se aplica inmediatamente y se vacía al arrancar el dispositivo. En el caso del datastore <*candidate*>, los cambios efectuados no se aplican en <*running*> hasta que no se ejecute el procedimiento <*commit*>. Además, podemos establecer una configuración con la que arrancará el dispositivo mediante el datastore <*startup*>.

Sin embargo, para poder poder operar con los servidores NETCONF desde un cliente, es imprescindible utilizar un modelo común que especifique el tipo de dato y cómo se estructura. Dicho cometido recae sobre el lenguaje de modelado YANG.

#### YANG

YANG (Yet Another Next Generation) [10] es un lenguaje de modelado que organiza jerárquicamente los datos, utilizando una estructura de árbol donde cada nodo puede contener un conjunto de nodos hijos (contenedor) o un nombre y valor. Además de implementar tipo de datos primitivos (*integer, decimal64, string, boolean, enumeration, bits*, etc.), posibilita definir datos más complejos mediante operadores (*typedef, type, container, leaf, list*).

Utilizar este lenguaje permite tanto al cliente como al servidor NETCONF validar los mensajes intercambiados. Para ello, se realiza internamente una conversión del modelo definido a una representación XML, también conocida como YANG Independent Notation (YIN).

En el Código 3 tenemos un modelo YANG y su correspondiente representación YIN, en el Código 4. Se trata de una definición simple de una lista de interfaces de red, compuesta únicamente por el nombre de la interfaz (del tipo *string*) y su *mtu* (del tipo *uint32*). Cuando el cliente realice un RPC, el mensaje se representará en formato XML, que será comparado con el modelo YIN (generado previamente del YANG) para encontrar posibles errores.

Por su parte, el servidor realizará la misma comprobación con su respectivo modelo YIN (generado de su modelo YANG). De esta forma, los mensajes pueden ser verificados y validados utilizando los mecanismos intrínsecos que proporciona XML.

```
module acme-foo {
   namespace "http://acme.example.com/foo";
   prefix "acfoo";

   import my-extensions {
       prefix "myext";
   }

   list interface {
       key "name";
       leaf name {
           type string;
       }

       leaf mtu {
           type uint32;
           description "The MTU of the interface.";
           myext:c-define "MY_MTU";
       }
   }
}
```

Código 3: Ejemplo modelo YANG.

```
<module name="acme-foo" xmlns="urn:ietf:params:xml:ns:yang:yin:1"
↪  xmlns:acfoo="http://acme.example.com/foo"
↪  xmlns:myext="http://example.com/my-extensions">

  <namespace uri="http://acme.example.com/foo"/>
  <prefix value="acfoo"/>

  <import module="my-extensions">
    <prefix value="myext"/>
  </import>

  <list name="interface">
    <key value="name"/>
    <leaf name="name">
      <type name="string"/>
    </leaf>
    <leaf name="mtu">
      <type name="uint32"/>
      <description>
        <text>The MTU of the interface.</text>
      </description>
      <myext:c-define name="MY_MTU"/>
    </leaf>
  </list>
</module>
```

Código 4: Modelo YIN.

## 2.3. Internet Protocol Security (IPsec)

### 2.3.1. Definición

Internet Protocol Security (IPsec) [5] es un conjunto de mecanismos para securizar las comunicaciones IP (Internet Protocol) mediante la autenticación y cifrado de cada paquete del flujo de datos. IPsec también incluye protocolos para establecer autenticación mutua entre agentes en el comienzo de una sesión y negociar claves criptográficas que se utilizarán en la misma.

Mantener la confidencialidad, autenticación e integridad en el nivel de red del modelo OSI implica la protección de todas las capas superiores, sin importar el protocolo utilizado en él.

A diferencia de otros sistemas de seguridad para proteger las comunicaciones, tales como Transport Layer Security (TLS), Secure Shell (SSH), las aplicaciones no necesitan ser diseñadas o adaptadas para beneficiarse de IPsec, gracias a que las comunicaciones están protegidos en capas inferiores, transparente para protocolos y aplicaciones de capas superiores.

### 2.3.2. Arquitectura

IPsec se divide esencialmente en 3 áreas: autenticación, confidencialidad y gestión de claves. Respectivamente, cada área se lleva a cabo por los siguientes protocolos (tanto en IPv4 como en IPv6):

- Authentication Header (AH): Cabecera de extensión IP que dota autenticidad e integridad a los paquetes. Utilizando un secreto compartido por ambos extremos de la comunicación, genera un Hash Message Authentication Code (HMAC) del contenido del paquete IP y algunas partes inmutables.
- Encapsulating Security Payload (ESP): Además de las características descritas en AH, posibilita la confidencialidad. Para ello, cifra el paquete IP completamente y lo encapsula en otro o añade una cabecera de cifrado al paquete original, dependiendo del modo de funcionamiento (descrito en 2.3.3). Es habitual que la confidencialidad vaya acompañada de autenticidad para solventar algunos problemas.

La gestión del tráfico IPsec se centra principalmente en dos campos:

1. **Security Association** (SA): La asociación de seguridad es un conjunto de parámetros acordados previamente entre dos entidades para el establecimiento de un canal seguro de comunicación. Es decir, las claves de cifrado, algoritmos utilizados, número actual de secuencia, etc.

Es un canal unidireccional, por lo que es común tener establecidas al menos dos SAs en un dispositivo (*inbound* para la entrada y *outbound* para la salida).

2. **Security Parameters Index** (SPI): Atributo que asocia un paquete con una asociación de seguridad (SA). El valor de dicho atributo junto con la dirección de destino del datagrama y el protocolo usado (ESP o AH, descritos a continuación), identifican de forma unívoca la SA.

Para hacer efectiva dicha gestión, se definen una serie de estructuras y mecanismos. Éstas son:

- **Security Association Database** (SAD): Base de datos almacenada en el sistema que contiene los parámetros de cada una de las SAs establecidas. Cada entrada estará compuesta por un identificador único (SPI), el protocolo encargado de proteger el tráfico (ESP o AH), el modo de funcionamiento, algoritmos y claves de autenticación y/o cifrado, tiempo de vida, etc.
- **Security Policy Database** (SPD): Base de datos que permite saber qué hacer con un paquete determinado (aplicar una SA o no). Cada entrada define un subconjunto de tráfico IP que apunta a una SA, para dicho tráfico. Cuando se recibe un paquete, se toma una de las siguientes acciones:
  - *Protect*: Aplicar los atributos de la SA almacenada en la SAD.
  - *Discard*: Descartar el paquete. La percepción en las capas superiores es de no haber recibido nunca ese paquete.
  - *Bypass*: No se aplica ninguna medida IPsec, permitiendo continuar el tráfico. Este tipo de paquetes no tendrán asociados ninguna SA.
- **Peer Authorization Database** (PAD): Almacena la información imprescindible para realizar la autenticación entre dos dispositivos. Actúa como nexo entre la SPD y un protocolo para la gestión de Asociaciones de Seguridad. Un ejemplo de ello sería IKEv2 [26].

### 2.3.3. Modos de funcionamiento

En IPsec existen dos modos de funcionamiento, el modo túnel y el modo transporte. Tanto AH como ESP son capaces de trabajar con ambos modos. A continuación entraremos más en detalle sobre su funcionamiento.

#### Modo transporte

El modo transporte ofrece únicamente protección a los protocolos de las capas superiores, pues utiliza mecanismos de seguridad en el contenido del

paquete (*payload*) por regla general. Esto se debe a que la construcción del paquete varía según el protocolo usado:

- **AH**: Autentica todo el paquete excepto los campos mutables.
- **ESP**: Cifra únicamente el *payload*.
- **ESP con autenticación**: Cifra únicamente el *payload* y autentica las cabeceras ESP.

Este modo se utiliza habitualmente en escenarios host-to-host.

#### Modo túnel

A diferencia del modo transporte, el modo túnel protege el paquete IP completo. Para ello, convierte el datagrama original en *payload* de otro paquete IP, mediante la encapsulación. El modo túnel habitualmente es utilizado cuando la comunicación entre dos entidades atraviesa routers o firewalls (gateway-to-gateway). En base al protocolo, tenemos el siguiente comportamiento:

- **AH**: Autentica tanto el *payload* como las cabeceras.
- **ESP**: Cifra el *payload* del paquete nuevo, es decir, todo el paquete original.
- **ESP con autenticación**: Cifra el *payload* y autentica las cabeceras ESP.

### 2.3.4. Procesamiento de tráfico

Mediante las entradas de la SAD y SPD se procesa el tráfico en IPsec. Sin embargo, esto varía si es de entrada (*inbound*) o salida (*outbound*).

#### Inbound

Cuando un dispositivo recibe un paquete IP, éste debe ser procesado para verificar que tiene asociada alguna política y poder autenticarlo y/o descifrarlo si corresponde. Este proceso consta de las siguientes fases:

1. Se recibe un paquete IP y comienza el procesamiento IPsec.
2. IPsec determina si el paquete recibido es del tipo IPsec (contiene alguna cabecera ESP o AH) o del tipo IP (ninguna cabecera de seguridad).
3. Si el paquete no contiene ninguna cabecera ESP o AH, se comprueba que exista una política asociada a ese tráfico (una entrada en la SPD). En caso de encontrar una entrada con la acción BYPASS, IPsec deja

pasar el paquete a la capa superior (transporte). De no existir una entrada en la SPD o ser distinta a BYPASS, el paquete es descartado.

4. Si el paquete sí contiene alguna cabecera de seguridad, IPsec busca una entrada en la SAD. En caso de encontrar una entrada para dicho paquete, se aplicará el procesamiento ESP o AH que corresponda para entregarlo a la capa superior. De lo contrario, el paquete es descartado.

#### Outbound

De un modo similar, se ilustra el procesamiento del tráfico saliente en IPsec.

1. IPsec recibe un paquete IP y comienza el procesamiento.
2. IPsec determina si el paquete recibido tiene una entrada asociada en la SPD.
3. Si se encuentra una entrada en la SPD, el paquete es procesado en base a la acción que tenga asociada.
   - DISCARD: El paquete es descartado.
   - BYPASS: IPsec deja pasar el paquete a la capa inferior (enlace).
   - PROTECT: Se consulta en la SAD si existe una SA asociada al paquete.

   Si no se encuentra ninguna entrada en la SPD, el paquete se descarta y se informa del error.
4. Si exista una SA asociada al paquete en la SAD, se aplicará el procesamiento ESP o AH y el modo que corresponda para entregarlo a la capa inferior (enlace). De lo contrario, se invoca al protocolo de gestión de Asociaciones de Seguridad, IKE.

### 2.3.5. Internet Key Exchange (IKE)

Tal como se ha descrito en el apartado anterior, IKE [11] es un un protocolo de gestión y distribución de Asociaciones de Seguridad usado por IPsec. Este sistema permite que dos dispositivos negocien todos los parámetros para establecer automáticamente las respectivas entradas en la SAD.

La primera versión de IKE, IKEv1, fue publicada en 1985, en el que se hacía uso de Oakley y de ISAKMP para realizar el intercambio y negociación de las claves. Sin embargo, presentaba limitaciones para acordar una clave (ambas partes debían usar exactamente la misma SA), problemas de escalabilidad, dificultades para depurar, múltiples interpretaciones de la especificación, etc. En diciembre de 2005, se introduce IKEv2, con la misma

funcionalidad que IKEv1 pero con importantes cambios: Se prescinde del uso de Oakley y de ISAKMP, menor número de RFCs o soporte de movilidad, entre otros.

IKEv2 utiliza un intercambio de claves criptográficas de tipo Diffie-Hellman para establecer un secreto compartido entre ambos extremos de la comunicación. Con ello, las entidades crean una asociación de seguridad denominada IKE SA, que se usará para negociar de forma segura las SA que utilizará IPsec para proteger el tráfico de datos, también conocidas como CHILD SA. Todo este proceso se compone esencialmente de tres fases:

1. **IKE_SA_INIT**: Negociación de algoritmos criptográficos con los que se derivará el secreto compartido, necesario para crear una IKE SA.
2. **IKE_SA_AUTH**: Las entidades involucradas en el proceso intercambian sus identidades para autenticarse.
3. **CREATE_CHILD_SA**: Creación de las SA que serán utilizadas para la protección de tráfico IPsec.

## 2.4. Gestión de IPsec SAs utilizando SDN

SDN cambia completamente la forma de gestionar el tráfico de las redes. Es capaz de ofrecernos un increíble abanico de posibilidades para administrar el tráfico de forma flexible y eficaz. No hay límites en extender este paradigma a la infraestructura y los servicios de una red. Y es precisamente en los servicios donde despliega un gran potencial en el ámbito de la seguridad: firewalls, detección y prevención de intrusos (IDS/IPS), establecimiento de canales seguros (IPsec) o filtros de correo basura. Estas entidades son conocidas como Network Security Fundation (NSF) [27].

Históricamente, las NSFs se desplegaban en dispositivos físicos de la propia red para ofrecer dichos servicios de seguridad. Mantener una infraestructura segura (despliegue de dispositivos, cumplimiento con las normativas, auditorías, personal cualificado, etc.) suponía una gran barrera para la mayoría de las organizaciones.

Para resolver este problema, numerosas empresas están desarrollando soluciones de seguridad en la nube desde hace varios años, permitiendo a otras organizaciones disponer de un servicio de seguridad sin necesidad de afrontar el coste que ello implica. La tendencia de este mercado es alcista donde se prevee un incremento de demanda de hasta el 14 % en los próximos años [28].

Este rápido crecimiento plantea todo tipo de retos para la estandarización de este tipo de soluciones [27]:

- Coexistencia de diversos tipos de NSFs desplegados en condiciones y entornos singulares.

- Gestión de las entidades mediante distintas interfaces de comunicación.
- Despliegue muy distribuido debido a la virtualización de las entidades NSF.
- Adaptar el comportamiento de forma dinámica.
- Carencia de estándar para describir un NSF independientemente del vendedor.
- Falta de mecanismos para efectuar cambios de configuración basados en alertas externas
- Ausencia de mecanismos para la distribución dinámica de claves criptográficas a los NSFs.

Las redes son entornos distribuidos por su propia definición. Acordar distintos parámetros entre dos o más nodos para establecer asociaciones de seguridad en IPsec fuerza que el administrador configure uno a uno todos los dispositivos implicados, ya sea para dichas asociaciones o un protocolo que se encargue de ello (IKE).

En un entorno SDN es posible gestionar todo este proceso desde una entidad central denominada Controlador, por lo que un administrador puede centrarse únicamente en definir las políticas que quiere en la red y el Controlador se encargará de implantarlas.

A fecha de hoy, se está desarrollando la estandarización de un escenario que habilite la gestión de asociaciones de seguridad en IPsec en entornos SDN [14]. Para ello, se han definido dos casos de uso, caso 1 y caso 2. En ambos se proporcionan mecanismos para distribuir y configurar IPsec tanto en escenarios simples (túnel IPsec entre dispositivos de la misma red) como en más complejos (dispositivos separados por uno o mas *gateways*).

### 2.4.1. Caso 1

En primer escenario se asume que los dispositivos soportan tanto IPsec como la administración de asociaciones de seguridad mediante IKE. En este escenario, el Controlador gestiona tanto IPsec (SPD y PAD) como la configuración IKE. Se establece una asociación de seguridad mediante la negociación del protocolo IKEv2.

Este escenario habitualmente será más sencillo llevarlo a cabo porque una gran mayoría de NSFs disponen de una implementación IKEv2.

### 2.4.2. Caso 2

Al igual que en el caso 1, se presupone que el dispositivo de red soporta IPsec. Sin embargo, en este escenario no se usa IKE como mecanismo para negociar los distintos parámetros que permitan el establecimiento de asociaciones de seguridad, pues es el propio Controlador quien se encargará de gestionar las IPsec SAs mediante el seguimiento y la administración la SPD y SAD.

El caso 2 posibilita NSFs más livianos, pues carecen de una implementación IKE y toda la complejidad que ello conlleva.

# Capítulo 3

# Objetivos y Metodología Empleada

En el anterior capítulo se han explicado algunas tecnologías que nos permiten afrontar el proyecto desarrollado en este trabajo fin de grado. Este proyecto continúa la línea de investigación de dos Trabajos Fin de Grado [29] [30] realizados previamente. Estos se enfocan en el desarrollo de una implementación para los dispositivos de red que permitiese establecer asociaciones de seguridad IPsec por parte de una entidad centralizada. La comunicación entre la entidad y los dispositivos se realiza mediante el protocolo NETCONF, en la que la entidad es un cliente NETCONF en el que se introduce manualmente la configuración para cada dispositivo de red.

El principal objetivo del proyecto es el estudio del software de libre distribución ONOS, para ser integrado como Controlador SDN en un escenario de gestión automática de políticas de seguridad IPsec en dispositivos de red, mediante el protocolo NETCONF. Es necesario analizar la arquitectura de ONOS, las diversas posibilidades que ofrece para desarrollar sobre esta plataforma y diseñar e implementar un prototipo de software que, mediante ONOS, realice la gestión automática de asociaciones de seguridad IPsec.

Para llegar a una solución que cumpla el objetivo propuesto, se ha seguido una metodología consistente en una serie de pasos que van desde el análisis hasta la implementación del prototipo:

1. Investigación de todos los aspectos del paradigma SDN y análisis de la problemática asociada a la gestión de seguridad.
2. Lectura de RFCs y *drafts* de IETF, así como distintos artículos de investigación y trabajos relacionados con la línea del proyecto realizado.
3. Estudio de los escenarios desarrollados previamente para la gestión de asociaciones de seguridad IPsec en entornos SDN.
4. Estudio detallado del Controlador SDN de ONOS, su plataforma de

desarrollo y soporte para NETCONF en la capa Southbound.

5. Pruebas y análisis de software y librerías potenciales para el desarrollo de la solución que se plantea.
6. Análisis de las alternativas existentes de un Controlador SDN y las diversas características que ofrecen.
7. Diseño de un prototipo de solución acorde a las características que ofrece el Controlador SDN de ONOS para gestionar automáticamente las SAs de IPsec.
8. Diseño e implementación de un prototipo de aplicación *(app)* de ONOS que permita automatizar el proceso de configuración de los dispositivos de red.
9. Integración de la *app* en el escenario y validación del correcto funcionamiento.
10. Validación del completo funcionamiento del escenario, verificando que se establecen las asociaciones de seguridad IPsec en los dispositivos de red por parte de la *app* que se encuentra integrada en el Controlador SDN de ONOS.
11. Planteamiento de vías futuras de trabajo que permitan continuar el desarrollo de la solución planteada.

# Capítulo 4

# Solución SDN

## 4.1. Estudio de ONOS

Para que el escenario planteado sea posible, es imprescindible que exista una entidad central que gestione la red, siendo ONOS en nuestro caso. Sin embargo, un Controlador SDN generalmente no es un software estático, diseñado exclusivamente para realizar una tarea concreta. Por el contrario, gestiona una extensa cantidad de protocolos de comunicación y distintas capas del modelo OSI. Gran parte del comportamiento deseado por los usuarios se desarrolla de forma complementaria a un Controlador, por lo que es necesario conocer cómo funciona ONOS y de qué manera se puede utilizar para nuestro propósito.

En esta sección se realizará un estudio de la plataforma de ONOS y las capacidades que puede ofrecer en un entorno SDN basado en NETCONF como protocolo Southbound y en YANG como lenguaje de modelado. Para ello se analizará distintos aspectos que serán de especial interés en las siguientes partes del proyecto.

### 4.1.1. Arquitectura

ONOS es un proyecto modular basado en Open Services Gateway initiative (OSGi) [31] y desarrollado en Java. Desde sus inicios, ONOS se ha desarrollado teniendo en cuenta los siguientes aspectos [32]:

- ***Código modular***: El proyecto está compuesto por distintos componentes, donde cada uno de ellos es un proyecto independiente. Para ello, se aprovecha las características que ofrece Maven [33].
- ***Configurable***: Gracias al uso de Apache Karaf [34], es posible activar o desactivar distintas características tanto en el arranque como en tiempo de ejecución.

- ***Delimitado***: Se definen claras separaciones entre los distintos componentes para ofrecer dicha modularidad.
- ***Agnóstico en las capas***: Tanto las aplicaciones como los protocolos no deben considerar en ningún momento implementaciones específicas de los dispositivos. Para evitarlo, se crean módulos independientes que se comunican con la Southbound API de ONOS y se integran en el núcleo.

Para entender cómo se estructura ONOS, es importante remarcar el término *servicio*, formado por uno o más componentes que definen una unidad de funcionamiento dentro del paradigma de ONOS. Este conjunto de componentes se denomina *subsistema*. Una ilustración de algunos subsistemas que existen en ONOS la podemos encontrar en la Figura 4.1:

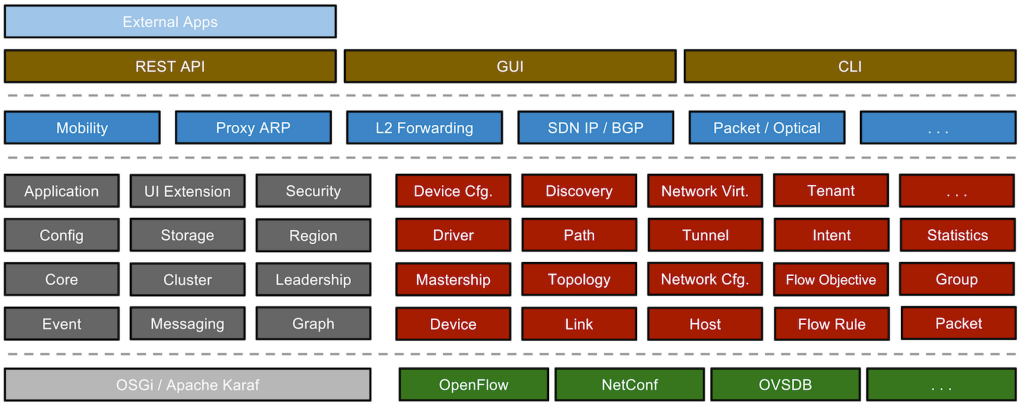


Figura 4.1: Subsistemas de ONOS [1].

Existen muchos subsistemas dentro del núcleo de ONOS, pero tienen especial relevancia los descritos a continuación:

- *Device*: Administra el conjunto de dispositivos en la infraestructura.
- *Host*: Maneja los dispositivos físicos y sus ubicaciones en la topología
- *FlowRule*: Administra el inventario de las reglas y acciones de flujos en los dispositivos y provee interfaces para estadísticas.
- *Link*: Gestiona y representa los enlaces físicos entre los dispositivos de red.
- *Packet*: Administra la comunicación a nivel de paquete de los dispositivos.

Estas entidades que hemos remarcado, a pesar de ser independientes dentro del núcleo de ONOS, son invocadas explícitamente (por ejemplo, cuando queremos acceder a los dispositivos registrados en ese momento) o de forma implícita (por otras entidades) en casi cualquier escenario que gestione el Controlador de ONOS.

Todos los *subsistemas* pertenecen a uno de los 3 niveles que hay definidos en el paradigma de ONOS: aplicación (*App*), manejador (*Manager*) o proveedor (*Provider*). La Figura 4.2 representa la relación que hay entre las distintas partes de un subsistema, siendo las marcas de puntos la delimitación de Northbound API y Southbound API.

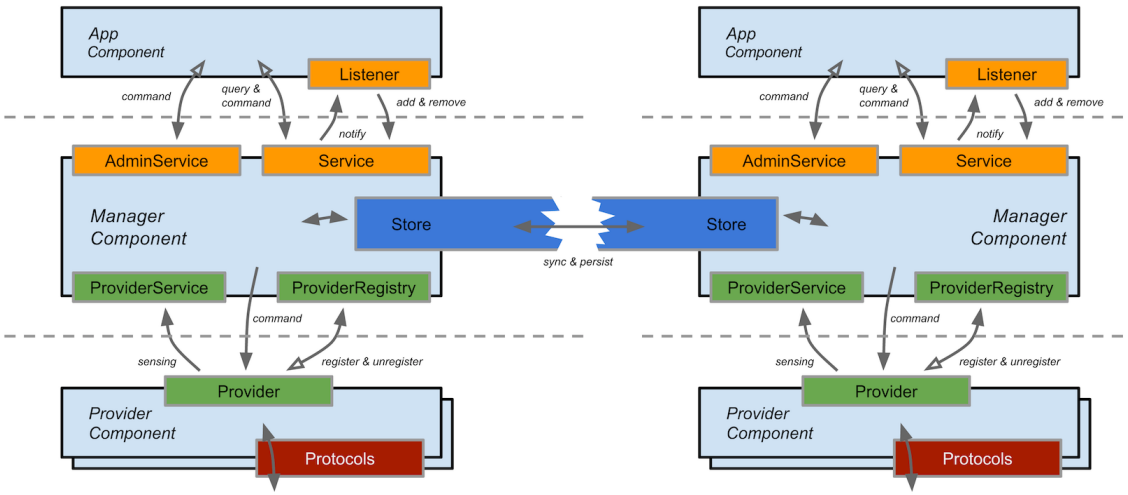


Figura 4.2: Arquitectura del subsistema de ONOS [1].

Los componentes se caracterizan por:

- **Provider**: Se trata de la capa más baja ONOS. Interacciona directamente con la red a través de librerías e implementaciones específicas de distintos protocolos mediante la interfaz *Protocols* y con la capa del manejador mediante la interfaz *Provider*.
- **Manager**: La capa del manejador es el núcleo del SDN. Recibe información de los *Providers* y lo presenta en la capa de Aplicación u otros servicios. Define las siguientes interfaces:
  - La interfaz *AdminService* recibe ordenes de administración y las aplica en la red o el sistema.
  - *Service* es la interfaz Northbound con la que las aplicaciones pueden obtener información sobre un aspecto particular del estado de la red. Además, engloba toda la gestión relativa a los disparadores de eventos y su gestión.
  - La interfaz Southbound *ProviderService* mantiene la comunicación e intercambia datos con la capa Provider.
  - *ProviderRegistry* permite a los distintos componentes de la capa Provider darse de alta con el Manager e interactuar con él.

  A su vez, muy relacionado con el Manager pero aislado como todo componente, se encuentra la entidad *Store*. Actúa como Eastbound/-

Westbound API y asegura la consistencia y robustez de la información entre las distintas instancias de ONOS y el Manager.

- **App**: En la capa de aplicaciones se consume la información suministrada por los Managers a través de sus interfaces Northbound, *AdminService* y *Service*. Las funcionalidades que puede ofrecer una aplicación son casi ilimitadas; desde visualizar diversos aspectos de una topología en el navegador web hasta establecer rutas para tráfico de red. Además, recibe notificaciones cuando un dispositivo se registra o se elimina de ONOS mediante la interfaz *Listener*.

### 4.1.2. Análisis de otras alternativas

ONOS es una solución de controlador SDN, pero no es la única. Existe una gran variedad de soluciones que se adaptan a distintas necesidades. Para el análisis de otras alternativas, se han considerado los siguientes aspectos:

- **Licencia**: Una de las características a destacar es el tipo de licencia con el que se utiliza el software. Distinguimos dos categorías básicas: soluciones comerciales y soluciones libres.

  En las soluciones comerciales, el código fuente del software no está disponible para el público, pues es una empresa quien lo desarrolla y ofrece sus servicios a través de dicho producto. En el caso del software libre, el código fuente sí está disponible públicamente, pero también puede haber una empresa detrás de un proyecto libre, que ofrezca diversos servicios relacionados con el software y que lo desarrolle con el apoyo de una comunidad.

- **Arquitectura**: En lo que se refiere a arquitectura, entendemos como distribuida aquella en el que existan interfaces Eastbound/Westbound API que permitan tener múltiples Controladores operando sobre la red, de tal forma que cada uno de ellos es completamente autónomo de los demás, pero que trabaja de forma conjunta. De no ser así, implica una estructura centralizada donde, existe un Controlador único que se encarga de gestionar todo.

- **Interfaces de comunicación**: Otro aspecto interesante a destacar es el soporte que existe para las interfaces **Northbound API** y **Southbound API**. En el caso de la primera, determina la complejidad para desarrollar aplicaciones que generen la lógica de un escenario. Por otro lado, el soporte de protocolos en la interfaz Southbound puede llegar a ser un factor crítico para decantarnos por un software concreto, pues está directamente ligado a los mecanismos con los que trabajan los dispositivos de red.

- **Lenguaje**: Por último, el lenguaje de programación en el que se ha escrito el Controlador SDN puede determinar la facilidad o complejidad

para extender nuevas funcionalidades o integrarlo con otras entidades NSF.

Toda esta información es la que se contrasta en la Tabla 4.1. Cabe remarcar la gran aceptación que tiene OpenFlow en la Southbound API y la tendencia de implantar una interfaz REST API en la Northbound API.

| *Producto* | *Licencia* | *Arquitectura* | *Northbound API* | *Southbound API* | *Lenguaje* |
|---|---|---|---|---|---|
| Beacon [35] | GPLv2 | centralizada | Java API | OpenFlow | Java |
| DISCO [36] | —— [1] | distribuida | Java API | OpenFlow | Java |
| Floodlight [37] | Apache-2.0 | centralizada | REST API, Java API | OpenFlow | Java |
| HPE SDN [38] | comercial | distribuida | REST API, Java API | OpenFlow | Java |
| Kandoo [39] | —— [1] | jerárquica distribuida | C, Python, Java, RPC API | OpenFlow | C, C++, Python |
| Onix [40] | comercial | distribuida | NIB API | OpenFlow, OVSDB | C++ |
| Maestro [41] | LGPL v2.1 | centralizada | Java API | OpenFlow | Java |
| OpenMUL [42] | GPLv2 | centralizada | REST API, C, Python | OpenFlow, NETCONF, OVSDB | C |
| NOX [43] | GPLv3 | centralizada | C++, Python | OpenFlow | C++ |
| OpenContrail [44] | Apache-2.0 | centralizada | REST API | BGP, NETCONF | C++ |
| OpenDaylight [45] | EPL v1.0 | distribuida | REST API, Java API | OpenFlow, BGP, Netconf, OVSDB | Java |
| ONOS [4] | GPLv3 | distribuida | REST API, Java API | OpenFlow, NETCONF, BGP, SNMP | Java |
| POX [46] | GPLv3 | centralizada | Python | OpenFlow | Python |
| Ryu [47] | Apache-2.0 | centralizada | Python API | OpenFlow, Netconf, OVSDB, BGP | Python |
| SMaRtLight [48] | —— [1] | distribuida | REST API | OpenFlow | Java |
| Trema [49] | GPLv2 | centralizada | Ruby | OpenFlow | Ruby |

Tabla 4.1: Comparativa de distintas soluciones SDN.

[1] No existe información acerca de la licencia para este software

## 4.2. Diseño escenario SDN

El objetivo de esta sección es poner en contexto todo el paradigma SDN en un escenario real. La finalidad es diseñar e implementar una prueba de concepto de un escenario SDN que gestiona de forma automática una serie de políticas relacionadas con el establecimiento de asociaciones de seguridad IPsec desde un controlador centralizado, ONOS. Estas políticas de seguridad son definidas por un administrador de red.

Tal como se ha comentado en el apartado 2.4 referente a líneas de trabajo relacionadas con este TFG, existen dos casos de uso propuestos. La diferencia esencial entre ellos es que el caso 1 utiliza el protocolo IKEv2 mientras que el caso 2 prescinde de ello. A pesar de que el mecanismo IKE facilita la configuración automática de asociaciones de seguridad, prescindir de él en un entorno centralizado no es tan grave. El Controlador SDN tiene pleno conocimiento de la topología y sus dispositivos en todo momento, lo que permite gestionar las asociaciones de seguridad directamente desde el Controlador, evitando la sobrecarga de una implementación IKE completa en los dispositivos de red.

Pese a que ambos casos son susceptibles de ser implantados en el escenario de este trabajo, nos hemos decantado por el caso 2 para realizar la puesta en marcha de la solución, porque resulta más interesante a la hora de gestionar las SAs de cara al escenario y por la carga de trabajo que ello implica, pues no ha habido tiempo suficiente para probar ambos casos.

Por otro lado, el escenario estará definido por una política de seguridad muy simple: Todo dispositivo debe establecer una canal de comunicación cifrado y autenticado con cada uno de los demás dispositivos en la red.

### 4.2.1. Requisitos de escenario

Para el desarrollo de la solución, se ha planteado un diseño compuesto por un Controlador SDN y $n$ dispositivos de red, tal como muestra la Figura 4.3. La comunicación entre el Controlador y los dispositivos pertenece al plano de control, mientras que el flujo de tráfico entre los dispositivos pertenece al plano de datos.

Para poder realizar la prueba de concepto de una implementación real que gestione IPsec, nuestro escenario debe cumplir una serie de requisitos:

1. Dos o más dispositivos deben ser capaces de procesar tráfico del nivel de red del modelo OSI y disponer de una implementación IPsec funcional. Además, es necesario que dispongan de al menos dos interfaces de red: una para la comunicación con el Controlador en el plano de control y otra para el tráfico entre los dispositivos.

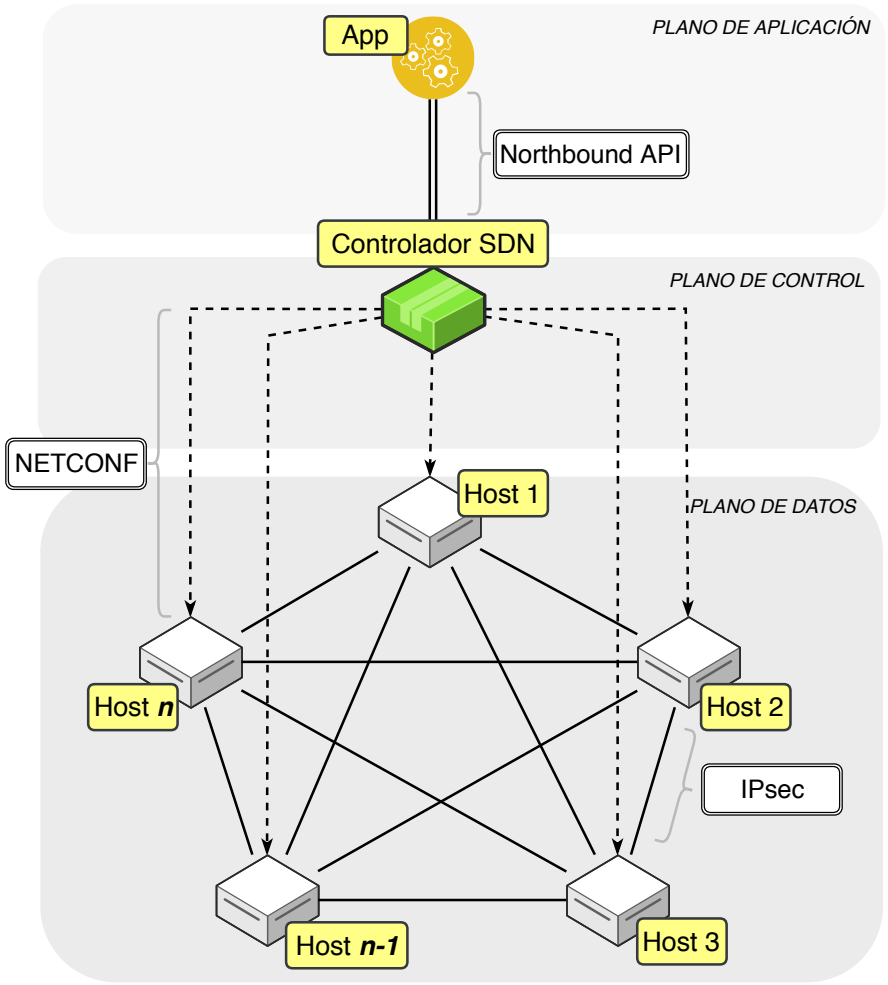


Figura 4.3: Escenario SDN propuesto.

2. Un dispositivo que ejerza de Controlador, siendo ONOS en nuestro proyecto. Éste debe ser capaz de establecer asociaciones de seguridad en los dispositivos y disponer de dos interfaces de comunicación: una para los dispositivos de red (plano de control) y otra para la gestión del propio Controlador. El Controlador deber ser capaz de procesar las configuraciones y aplicarlas a los dispositivos en los planos de datos.

3. Debe existir un protocolo de comunicación *Southbound* que permita el intercambio de información entre el Controlador y los dispositivos de red. Será NETCONF quien ejerza esa función en nuestro escenario.

4. Un modelo de datos entre el Controlador y los dispositivos de red que permita la gestión de IPsec en los dispositivos de red. Para este proyecto se usará el modelo YANG *ietf-ipsec* [14].

5. El servidor NETCONF de los dispositivos de red debe ser capaz de gestionar todas las estructuras IPsec para aplicar la configuración que reciba del Controlador en el plano de control, acorde al modelo de datos

compartido entre ambas entidades.

6. Disponer de un protocolo de comunicación *Northbound* para poder definir las distintas configuraciones que queremos que se apliquen al escenario.
7. Un componente o aplicación que se encargue de la lógica del escenario. Ésta, a través del Controlador, debe ser capaz de recibir información del estado del escenario y ejecutar acciones sobre los dispositivos.

La satisfacción de estos requisitos para nuestro escenario se discuten en los apartados a continuación.

### 4.2.2. Comunicación Southbound

Para nuestro escenario es necesario que el Controlador de ONOS realice la comunicación Southbound mediante el protocolo NETCONF. El núcleo de ONOS dispone de una funcionalidad limitada. Sin embargo, el potencial reside en su estructura modular mediante el uso de subproyectos independientes. Estos subproyectos o módulos dotan al Controlador capacidad de comunicarse con diversos protocolos, mostrar estadísticas, recalcular rutas, gestionar alarmas, etc.

Dentro del núcleo de ONOS, existen una serie de módulos que en conjunto brindan soporte NETCONF. Éstos son:

- **netconf**: *Provider* que proporciona los medios para que ONOS descubra y active el procedimiento inicial de handshake con NETCONF a partir de la información proporcionada por la configuración de red.
- **netconfsb**: Expone APIs para establecer conexiones NETCONF a dispositivos y para enviar y recibir mensajes sobre dicha conexión.
- **drivers.netconf**: Implementación genérica del protocolo.

Este desglose de las características NETCONF que ofrece el Controlador SDN de ONOS se debe a su estructura modular, pues se utilizan todos de forma implícita.

#### IETF-IPsec

Los dispositivos de nuestro escenario necesitan disponer de una implementación de NETCONF junto a un servicio que aplique la configuración IPsec recibida en el sistema operativo. Para el servidor NETCONF, haremos uso de Netopeer [50]. Este servidor permite integrar en su arquitectura distintos módulos, que serán los encargados de ejecutar las operaciones definidas por un módulo YANG.

Nuestro servicio, que aplicará la configuración IPsec en el sistema, se basará en el modelo YANG *ietf-ipsec*, propuesto por el *draft ietf-i2nsf-sdn-ipsec-flow-protection* [14] que posibilita representar toda la información relacionada con IPsec e IKE. La implementación de dicho servicio será proporcionada por el módulo IETF-IPsec, que utiliza el framework *transAPI* de la librería *libnetconf*. Ésta, a través de la invocación de los procedimientos remotos, aplicará los cambios mediante llamadas al sistema operativo acorde a las llamadas RPC del modelo YANG *ietf-ipsec*. En el apéndice B podemos ver la estructura que presenta este modelo de datos.

En lo que concierne a nuestro escenario, los únicos cambios necesarios para integrar dichas herramientas son la adaptación de los mensajes enviados por el Controlador, relativos a su topología.

### 4.2.3. Comunicación Northbound

Tal como se muestra en la Figura 4.1, ONOS permite que un usuario interactue con el Controlador a través de 3 interfaces distintas:

- **REST API**: Una interfaz REST nos permite utilizar métodos HTTP para gestionar distintos aspectos del Controlador y todo su entorno. ONOS internamente realiza una conversión de las interfaces Java pertinentes y modelos YANG a rutas HTTP, por lo que no es necesario realizar una implementación específica para poder usarlo [51].
- **GUI**: La interfaz GUI es la aplicación web que expone ONOS, por defecto en el puerto 8181. Si la app con la que trabajamos dispone de una implementación para la interfaz gráfica, podemos realizar distintas gestiones directamente desde la aplicación web.
- **CLI**: Algunas apps pueden implementar una interfaz de línea de comandos que permita realizar alguna gestión. Como ONOS está basado en Apache Karaf [34], es posible utilizar la consola para invocar un operación concreta.

Esto permite al administrador gestionar distintos aspectos de ONOS, la topología, los dispositivos, etc.

En cualquier caso, es preciso darlos de alta para que el Controlador pueda conocer de su existencia. Esto puede ser efectuado con cualquiera de las 3 interfaces que implementa ONOS en la Northbound API, aunque es común usar la interfaz REST API mediante una solicitud POST a la ruta */onos/v1/network/configuration* del Controlador, con el siguiente mensaje JSON:

La configuración básica con la que daremos de alta un dispositivo de red en el Controlador SDN de ONOS está compuesto por 4 elementos:

base/netopeer/conf/netconf-cfg.sh

```
{
  "devices": {
    "netconf:$netopeerIP": {
      "netconf": {
        "ip": "$netopeerIP",
        "port": 830,
        "username": "$NETCONF_USER",
        "password": "$NETCONF_PASSWORD"
      },
      "basic": {
        "driver": "ovs-netconf"
      }
    }
  }
}
```

Código 5: Dar de alta un dispositivo NETCONF en ONOS.

- **ip**: La dirección IP del dispositivo de red a la que el Controlador debe conectase.
- **port**: El puerto del dispositivo de red al que ONOS se debe conectar. Por defecto NETCONF utiliza SSH como capa de transporte seguro y el puerto 830 tal como indica el RFC 6242 [52].
- Las creedenciales de acceso al dispositivo, mediante los argumentos **username** y **password**.
- **driver**: Especificamos la app que se encargará de la comunicación southbound con este dispositivo. Además de *ovs-netconf*, podemos indicar una implementación específica de nuestro dispositivo.

### 4.2.4. App ONOS

Para que el Controlador de ONOS sea capaz de gestionar la configuración IPsec de los dispositivos de red, es necesario desarrollar la lógica en una app, la cual integraremos posteriormente en el núcleo de ONOS.

ONOS dispone de una amplia documentación, entre la que tenemos ejemplos y plantillas para crear una app [53]. Crearemos una app de ejemplo, denominada *foo-app*, mediante el uso de Maven. Dentro de la arquitectura de ONOS, nuestra app trabajará en la capa del manejador, interactuando con el componente de NETCONF para establecer SAs a los dispositivos que se conecten al Controlador.

El funcionamiento de la app se compone de dos fases. La primera, realizar

una solicitud en el plano de control al dispositivo NETCONF para obtener su dirección IP del plano de datos. La segunda, inyectar asociaciones de seguridad al resto de los dispositivos NETCONF para establecer túneles IPsec.

Nuestra aplicación se suscribirá a los eventos que genera ONOS a través del *NetopeerListener*, lo que nos permitirá ejecutar acciones cuando un dispositivo NETCONF se conecte al Controlador.

La gestión y asignación de asociaciones de seguridad IPsec se gestionan desde *IPsecController*, quien hará uso del componente de NETCONF *ovs-netconf* de ONOS para establecer las SAs, siguiendo el modelo *ietf-ipsec*.

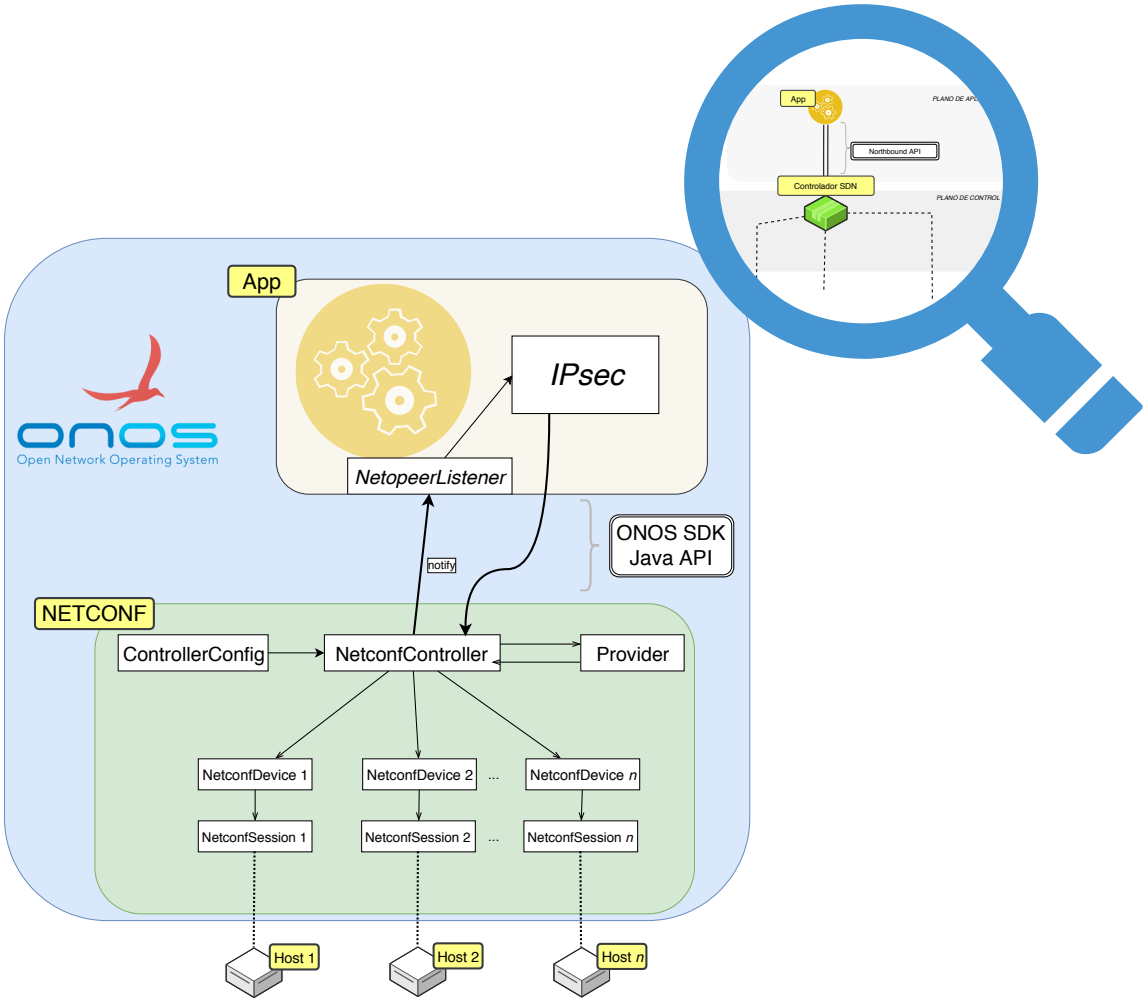


Figura 4.4: Arquitectura de la app de ONOS.

En ONOS, la arquitectura de *ovs-netconf* se basa en 5 entidades [54], representado en la Figura 4.4:

- ***ControllerConfig***: Entidad que se encarga de gestionar los Controladores de ONOS asignados a dispositivos NETCONF, ya que ONOS es capaz de trabajar de forma conjunta con otros Controladores de ONOS.

- ***Provider***: Gestiona todas las interacciones de los dispositivos NETCONF con el núcleo de ONOS. Es responsable de comprobar periódicamente los cambios de disponibilidad de los dispositivos NETCONF.
- ***NetconfController***: Entidad principal de la gestión NETCONF. Monitoriza los dispositivos NETCONF, actúa como nexo para acceder a los dispositivos y gestiona las suscripciones a los eventos NETCONF.
- ***NetconfDevice***: Representación de un dispositivo NETCONF conectado al núcleo de ONOS.
- ***NetconfSession***: Interfaz asociada a todo dispositivo NETCONF. Representa el punto de acceso a cualquier operación en el dispositivo.

En el Código 6. se muestra cómo se obtiene la sesión *s* del dispositivo *device*, que posteriormente se utiliza para realizar una solicitud NETCONF y obtener la respuesta.

base/onos/src/foo-app/src/main/java/org/foo/app/IPsec.java

```
89          NetconfSession s =
            ↪  controller.getNetconfDevice(device).getSession();
90
91          String reply = s.requestSync(request);
```

Código 6: Sesión NETCONF de un dispositivo.

Todas las apps de ONOS tienen una extensión *.oar*, las cuales se pueden instalar y cargar de forma dinámica en ONOS mediante el siguiente comando:

```
/bin/bash
onos-app <IP-ONOS> install! <app.oar>
```

Un aspecto importante durante el desarrollo de la app en ONOS es la invocación de otros componentes que conforman ONOS mediante llamadas SDK que ofrece la plataforma. Un claro ejemplo sería acceder al Controlador de NETCONF (*NetconfController*) desde nuestra app.

Para invocar dicho componente, nuestra app no debe realizar ninguna llamada especial debido a que ONOS realiza una inyección automática de instancias de dichos componentes cuando son declarados. Por tanto, únicamente debemos declarar *NetconfController* como un atributo más y añadir la anotación *@Reference()*, tal como aparece en la línea 41-42 del Código 7. De esta forma, podemos usar *NetconfController* directamente, sin preocuparnos qué forma acceder a él.

Al tratarse de una prueba de concepto, no se ha profundizado en la serialización y deserialización de mensajes NETCONF a partir de modelos YANG

base/onos/src/foo-app/src/main/java/org/foo/app/AppComponent.java

```
35  @Component(immediate = true)
36  public class AppComponent {
37
38      @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
39      protected CoreService coreService;
40
41      @Reference(cardinality = ReferenceCardinality.MANDATORY_UNARY)
42      protected NetconfController deviceController;
43
44      private final Logger log = LoggerFactory.getLogger(getClass());
45
46      private ApplicationId appId;
47      protected NetopeerListener netListener = null;
```

Código 7: Esqueleto app ONOS.

mediante la SDK de ONOS. La generación de mensajes NETCONF se realizará directamente desde la app. Sin embargo, cabe mencionar lo siguiente:

- ONOS internamente transforma el modelo YANG en un esqueleto XML (que no es el modelo YIN), mediante el uso de **pyang** [55], y lo gestiona con una herramienta interna denominada *YangXMLUtils* [56]. Por tanto, trabajar con modelos YANG en el SDK de ONOS continúa siendo engorroso, necesitando conocer cómo se estructura un modelo cuando se manipula los mensajes XML.
- El driver *ovs-netconf* no está completamente implementado. No gestiona la capa de mensaje (*message layer*) de NETCONF ni tampoco dispone de algunas características básicas (descubrimiento de puertos, topologías, etc.), por lo que es necesario encapsular manualmente todas las operaciones NETCONF.

### 4.2.5. Modelo y configuración ietf-ipsec

Para que se puedan establecer correctamente una asociación de seguridad IPsec entre dos dispositivos de red dentro de nuestro escenario SDN, es necesario disponer de una serie de elementos clave:

- El mismo **modelo de datos** (YANG) en los dispositivos de red y en el Controlador, que proporcione la capacidad de gestionar los distintos parámetros de una asociación de seguridad IPsec. Este requisito lo proporciona *ietf-ipsec* en nuestro escenario.
- Una configuración NETCONF acorde al modelo *ietf-ipsec* que el Controlador envíe a los dispositivos de red.

- Un software en el dispositivo de red que interprete la configuración recibida por el Controlador (acorde al modelo *ietf-ipsec*) y aplique las modificaciones para efectuar esa configuración IPsec. Este software se describe en el apartado 4.2.2.

El modelo YANG *ietf-ipsec* cubre las posibles configuraciones tanto del caso 1 como del caso 2 como se muestra en el apéndice B. No obstante, analizaremos únicamente las referentes al caso 2.

Tal como se ha explicado previamente en el apartado 2.2.5, YANG organiza los datos jerárquicamente. Toda configuración hereda del nodo padre <ietf-ipsec>, del cual hereda un nodo <ikev2> (caso 1) o <ipsec> (caso 2). A su vez, dentro de ese nodo <ipsec>, existe un subnodo <spd> y otro <sad>, que estarán formados por un indeterminado número de nodos <spd-entry> y <sad-entry>, respectivamente.

En el caso de <spd-entry>, cada nodo representa una entrada en la SPD del dispositivo de red. Su funcionamiento se compone de 3 partes:

1. El nodo <condition> se compone por uno o más <traffic-selector-list> que actúan como filtros, pues indican a qué tráfico IP va asociada esta entrada SPD. Dada una trama IP del dispositivo de red, una SPD se asocia o no en base a:
   - <local-addresses>: Dirección IP origen.
   - <remote-addresses>: Dirección IP destino.
   - <next-layer-protocol>: Protocolo de la capa superior.
   - <local-ports>: Puerto origen.
   - <remote-ports>: Puerto destino.

   También se establece la dirección del tráfico según el procesamiento IPsec (INBOUND, OUTBOUND).
2. En el nodo <processing-info> se asocia el tipo de acción a realizar para ese tráfico. En el subnodo <ipsec-sa-cfg> se define el protocolo IPsec y el modo de funcionamiento. Además, si el modo de funcionamiento IPsec (establecido en el subnodo <mode>) es del tipo túnel, se define la configuración relativa a ese modo en el subnodo <tunnel>).
3. De forma opcional, en el subnodo <spd-lifetime> se puede establecer un temporizador para dicha regla SPD.

Por otro lado, para cada entrada <sad-entry>, se define un SPI, filtros para el tráfico IP, información de los algoritmos de cifrado y autenticación en el subnodo <esp-sa>, temporizador para la entrada (subnodo <sa-lifetime>), el modo de funcionamiento IPsec, etc.

Esta completa jerarquía está adjunta en el apéndice B, teniendo una descripción más detallada en el *draft ietf-i2nsf-sdn-ipsec-flow-protection* [14]. Para ilustrar este modelo, en el apéndice C tenemos un mensaje NETCONF transmitido desde el Controlador ONOS a un dispositivo de red. Basándonos en el modelo *ietf-ipsec*, este mensaje establece la siguiente configuración IPsec:

1. Se definen dos entradas para la SPD en el dispositivo de red: una para el tráfico de entrada y otro para el tráfico de salida. La asociación de seguridad se realiza entre la dirección 192.169.0.3 (el dispositivo objetivo) y la 192.169.0.2, para todos los puertos de entrada y salida.
2. En ambas entradas SPD, la acción a efectuar es PROTECT, por lo que se aplicarán los servicios que ofrece IPsec. El protocolo es ESP en modo de funcionamiento transporte.
3. De igual modo, se crean dos entradas para la SAD, con el modo de funcionamiento y los filtros de tráfico acordes a los ya definidos para la SPD. Para el cifrado de tráfico, se utiliza el algoritmo *3des*, una clave *ecr_secret* y un vector de inicialización *vector*. Para la autenticación del tráfico, se utiliza el algoritmo *hmac-md5-128* con clave *auth*. Se usan las mismas claves y algoritmos para ambos sentidos del tráfico.

En este ejemplo, no se ha llegado a establecer ningún temporizador para las entradas de la SPD y la SAD, por lo que no caducarán una vez establecidas.

Suponiendo que el Controlador también establezca una configuración similar en el otro dispositivo NETCONF (192.169.0.2) pero invirtiendo los selectores de tráfico (entrada y salida), ambos podrán comunicarse de forma cifrada y autenticada entre si.

### 4.2.6. Flujo de mensajes

La configuración IPsec en los distintos dispositivos por parte del Controlador ONOS involucra distintas fases y entidades. En la Figura 4.5 observamos el flujo de mensajes:

Suponiendo que ONOS ha cargado nuestra aplicación y previamente existe un host, se siguen los siguientes pasos.

1. Se registra el nuevo dispositivo de red en el Controlador de ONOS usando el plano de control. Puede ser dado de alta por un administrador de red o algún método automático, como por ejemplo una llamada a la API REST con su propia información, tal como se detalla en el apartado 4.2.3.
2. El controlador ONOS inicializa internamente el nuevo dispositivo en su estructura de datos. Mediante los distintos eventos que suceden en ese

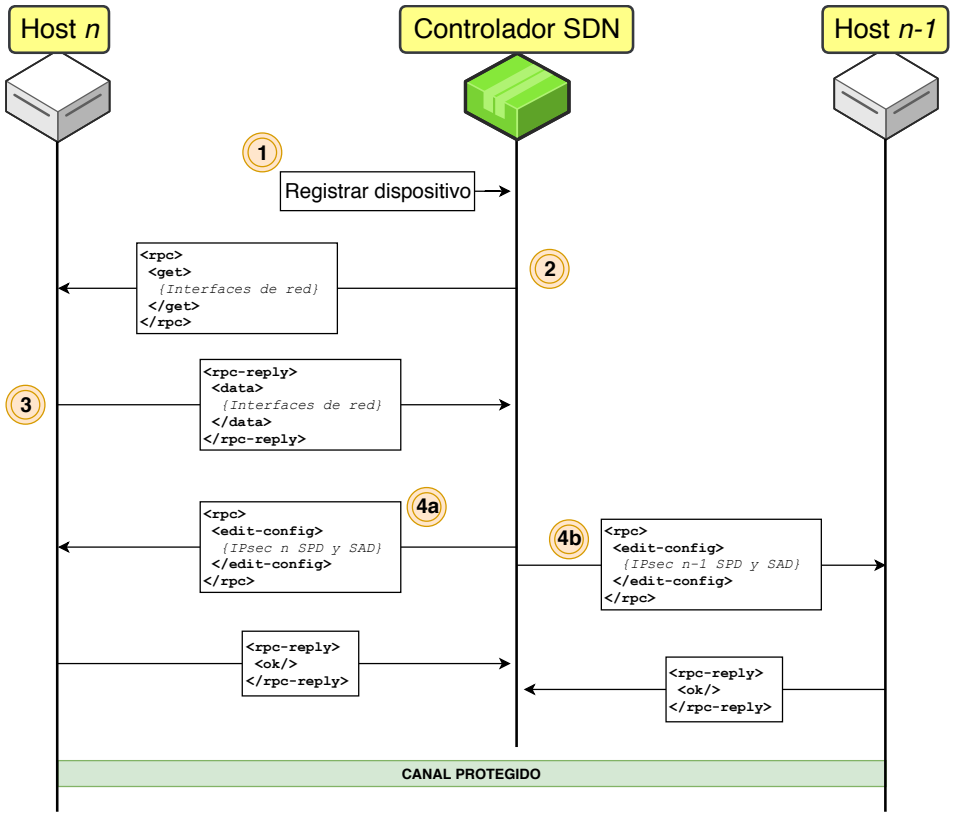


Figura 4.5: Flujo de mensajes en el escenario propuesto.

proceso, nuestra app recibe un *callback* del dispositivo registrado.

Inmediatamente después, se realiza una RPC NETCONF al nuevo dispositivo para obtener su dirección IP del plano de datos. Esto se hace mediante un simple filtro NETCONF, basandose en el modelo de datos YANG *ietf-interfaces* [57]. En el Código 8 se muestra el mensaje completo.

3. El nuevo dispositivo responde a la consulta del Controlador de ONOS con toda la información de la interfaz de red que utiliza para el plano de datos.

4. Si es el primer dispositivo NETCONF que nuestra app registra, no se realiza ningún paso más, pues no hay nadie con quien pueda establecer un túnel IPsec. De lo contrario, se realiza lo siguiente:

   *a*) Con la información del plano de datos, el Controlador realiza una llamada a procedimiento remoto para establecer las asociaciones de seguridad de todos los demás dispositivos de red en el recién conectado. La configuración se establece según la estructura del modelo YANG *ietf-ipsec*, representado en el apéndice B.

   *b*) De igual forma, se establecen las SAs en el otro dispositivo de red, usando el mismo modelo de datos YANG. No se consultan las

```xml
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get>
    <filter type="subtree">
      <interfaces-state
      ↪  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
        <interface>
          <name>eth1</name>
        </interface>
      </interfaces-state>
    </filter>
  </get>
</rpc>
```

Código 8: Filtro NETCONF para la IP del plano de datos.

interfaces de red del plano de datos del otro dispositivo porque ya han sido consultadas previamente por el Controlador.

A partir de este punto, la comunicación entre los dos dispositivos de red se realiza utilizando los mecanismos IPsec.

# Capítulo 5

# Desarrollo e implementación

Para llevar a cabo el escenario propuesto en la fase de diseño, se ha desarrollado una prueba de concepto en un entorno virtualizado. Este prototipo consta de los siguientes elementos:

- Un controlador SDN de ONOS, el cual hemos descrito en el apartado 4.1.
- Una aplicación de prueba, escrita en Java, que utiliza el SDK de ONOS para administrar la lógica del Controlador en nuestro escenario.
- Hosts virtuales que actuarán de dispositivos de red. Estos hosts estarán capacitados de un servidor NETCONF virtual, basado en el proyecto Netopeer [50], y soporte IPsec.

Todos estos elementos se desplegarán en forma de contenedores, con la ayuda de Docker [58], descrito en mayor profundidad en el D. Durante la implementación del diseño, se ha tenido muy en cuenta la flexibilidad, para permitir escalar y manejar un número dinámico de dispositivos de red. Sin embargo, usaremos 6 por simplicidad.

Siguiendo la política de seguridad definida en el diseño, todos los hosts deben crear dos SAs (inbound y outbound) con cada host de la red, de modo que la comunicación entre cualquier dispositivo esté protegida. En lo que respecta a las características desplegadas de IPsec, se ha optado por el modo de funcionamiento transporte, ya que nuestro escenario utiliza una única subred. En lo referente a protocolos de IPsec, es necesario cifrar y autenticar los mensajes, por lo que se hará uso de *AH + ESP*.

Haremos uso de dos subredes distintas para representar la separación entre el plano de datos y el plano de control. El plano de datos usará un rango de IPv4 *192.169.0.0/24*, al que estarán conectados todos los dispositivos de red. Por otro lado, el plano de control usará el rango *10.100.3.0/24*, que será la interfaz Southbound para el Controlador SDN de ONOS (*10.100.3.204*) con los dispositivos de red. Una representación gráfica de esta topología la

podemos observar en la Figura 5.1:

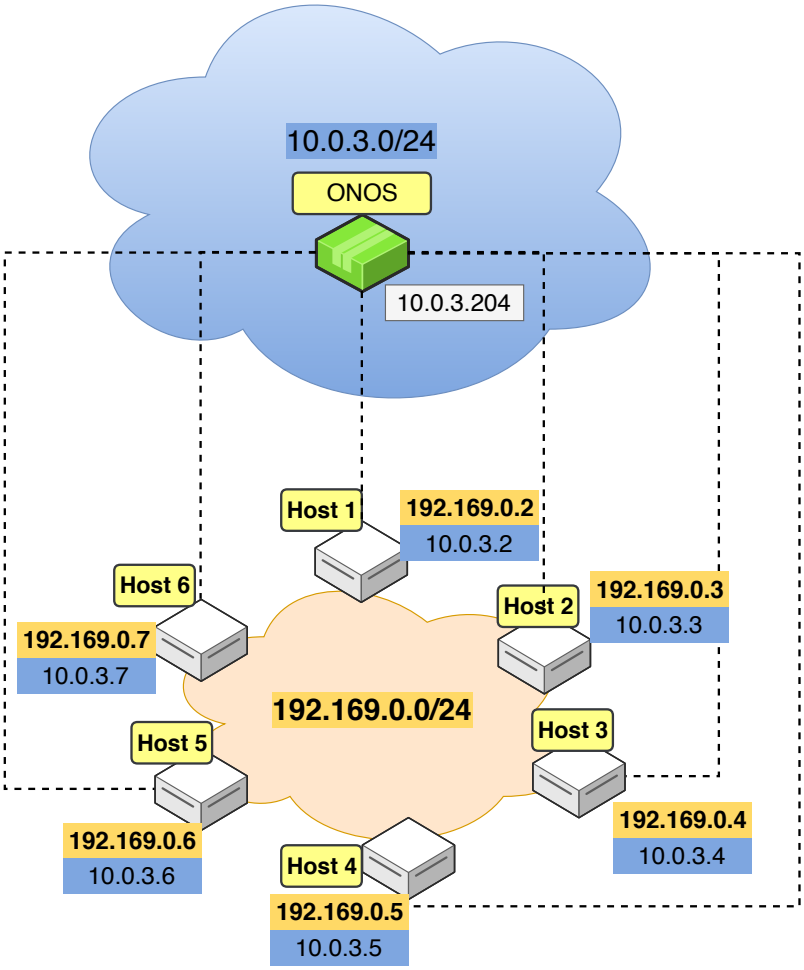


Figura 5.1: Topología del escenario propuesto.

El núcleo de nuestro escenario se encuentra en el archivo *docker-compose.yml*, en el que se declara las imágenes que se van a usar, las variables que se pasan a las imágenes, las redes que se quieren definir y la relación que existe entre las entidades. Tanto la imagen de ONOS como la de Netopeer tienen un directorio propio, donde se declara cómo se debe construir dicha imagen: las dependencias que necesita, los comandos que deben ejecutarse y qué debe ejecutarse cuando se vaya a lanzar en un contenedor.

Además, podemos activar los distintos componentes de ONOS de forma automática durante el arranque, indicando exactamente qué módulos queremos cargar dentro de la variable *ONOS_APPS*, tal como muestra el Código 9:

docker-compose.yml

```
4   onos:
5     build: base/onos
6     environment:
7         - ONOS_APPS='netconf, drivers.netconf, netconfsb, yang-gui,
          ↪  models.common, restconf, protocols.restconfserver'
```

Código 9: Carga módulos al arrancar ONOS.

## 5.1. ONOS SDK

Un aspecto muy importante de este trabajo ha sido la creación y puesta en marcha de un protipo de app que demostrase la capacidad de ONOS como Controlador SDN en entornos complejos.

Es preciso escribir la aplicación en Java, pues es el lenguaje que se usa en ONOS. Por la arquitectura modular e independiente de ONOS, crear una app no requiere compilarla junto a toda la plataforma de ONOS, pues gracias al modelo OSGi en el que se basa, podemos crear la app de forma independiente y cargarla posteriormente.

La app se estructura en 3 partes: *AppComponent.java*, *IPsec.java* y *NetopeerListener.java*.

- En **AppComponent** se definen los dos métodos que todo componente implementa en ONOS. Activar y desactivar una app. Para ello debemos indicarlo mediante la anotación *@Activate* y *@Deactivate*, tal como muestra el Código 10.

base/onos/src/foo-app/src/main/java/org/foo/app/AppComponent.java

```
49      @Activate
50      protected void activate() {
51          appId = coreService.registerApplication("org.foo.foo-app");
52          netListener = new NetopeerListener(deviceController);
53          deviceController.addDeviceListener(netListener);
54
55          log.info("onos-ipsec Started");
56      }
```

Código 10: Método que ONOS llama cuando se activa la app.

La principal tarea que realizamos es crear un objeto **netListener** de nuestro *NetopeerListener*, que implementa la interfaz de *NetconfDeviceListener*. A continuación, le indicamos al Provider de NETCONF de

ONOS que dicha clase se suscribirá a los eventos que genere, mediante la llamada *.addDeviceListener(**netListener**)*.

- En **NetopeerListener**, de una forma similar a *AppComponent*, contiene dos métodos implementados de la interfaz *NetconfDeviceListener*: *public void deviceAdded(DeviceId di)* y *public void deviceRemoved(DeviceId di)*. Ambos son invocados cuando se conecta o desconecta un dispositivo NETCONF, respectivamente. El método *deviceAdded(DeviceId di)* invoca nuestra función *createTunnels(DeviceId new_device)*, encargada de crear los túneles IPsec. Esta función es el núcleo de la app, pues se encarga de generar una SA por cada host registrado en el controlador de NETCONF (la lista de dispositivos se obtiene mediante *controller.getDevicesMap()*).
- Por último, en **IPsec** se concentra todo lo relativo a los mensajes y operaciones NETCONF relacionados con el modelo yang *ietf-ipsec*. Se ha desarrollado una serie de funciones genéricas que nos permiten generar distintas partes de un mensaje NETCONF basado en el modelo *ietf-ipsec*, listo para ser enviado a un dispositivo de red a partir de la función *private String getGenericEntry(IPsecStructure structure, DeviceId device, DeviceId newDevice, TRAFFIC_TYPE type)*.

Un mensaje NETCONF tiene varias partes dinámicas cuando se genera para un dispositivo recién conectado: entrada en la SPD con la IP del plano de datos del dispositivos recién conectado y con el que se quiere establecer el túnel, el modeo de funcionamiento de IPsec (transporte), acción (PROTECT, BYPASS, DISCARD), entrada en la SAD con los algoritmos de autenticación y cifrado, SPI asignado, etc.

Cuando ese mensaje se ha generado completamente a través de las plantillas, procederemos a encapsularlo en la capa de mensaje (*message layer*) de NETCONF mediante *.encapsulateSPDandSADinXML(String SPD, String SAD)*, obtener una sesión NETCONF (*NetconfSession*) de ese dispositivo y, enviar dicho mensaje mediante *.requestSync(Sstring netconf_message)*. Podemos ver esas operaciones en el Código 11, del **NetopeerListener**.

base/onos/src/foo-app/src/main/java/org/foo/app/NetopeerListener.jav

```
259     NetconfSession newDeviceSession =
        ↪ controller.getNetconfDevice(new_device).getSession();
260
        ↪ newDeviceSession.requestSync(IPsec.encapsulateSPDandSADinXML(all_SPD,
        ↪ all_SAD));
```

Código 11: Obteneción de una sesión NETCONF del dispositivo.

## 5.2. Despliegue del escenario

Para desplegar nuestro escenario con 6 dispositivos de red, debemos situarnos en el directorio padre del proyecto y lanzarlo mediante la siguiente orden:

```
/bin/bash
./up.sh 6
```

Este script no es más que una abstracción de docker para desplegar el proyecto, cuyo contenido se muestra en el Código 12.

up.sh

```
1  #!/bin/sh
2
3
4  if [[ -z $1 ]] || ( ! [[ $1 =~ ^[0-9]+$ ]] && [[ $1 -gt 0 ]] ); then
5    docker-compose up -d --build
6  else
7    echo Deploy with $1 instances
8    sleep 0.3
9    docker-compose up -d --build --scale netopeer=$1
10 fi
```

Código 12: Abstracción de Docker para desplegar el escenario.

La primera vez que se ejecute, descargará todas las dependencias necesarias y compilará distintas librerías para tener listas las imágenes de Netopeer y ONOS para nuestro escenario. A continuación, lanzará contenedores de dichas imágenes (1 contenedor de ONOS y 6 contenedores de Netopeer).

Tras arrancar los contenedores, mientras ONOS se está iniciando, los servidores Netopeer se dan de alta en el Controlador ONOS automáticamente mediante la solicitud HTTP POST, analizada previamente en el apartado 4.2.3. De esta forma, durante el arranque se automatiza el registro de un dispositivo en ONOS. En la Figura 5.2 se muestra cómo un dispositivo envía información acerca de su dirección IP y credenciales de acceso al controlador ONOS.

Posteriormente, se realiza un intercambio de mensajes correspondientes a la Figura 4.5 para establecer los túneles IPsec entre todos los dispositivos.

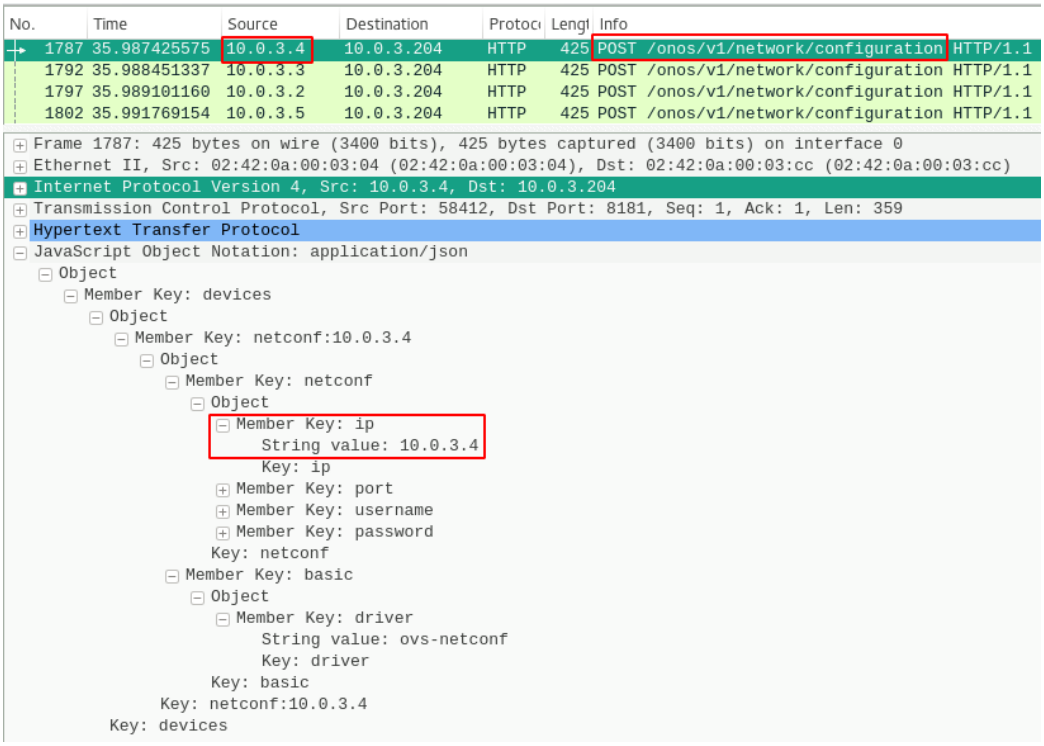


Figura 5.2: Traza de solicitud a ONOS al arrancar el dispositivo.

## 5.3. Prueba del escenario

Como ya se ha comentado previamente, este escenario se ha desarrollado teniendo siempre en cuenta la flexibilidad. Una vez se ha desplegado el escenario, todo el proceso se realiza automáticamente, por lo que después de realizar el despliegue, tendremos nuestros dispositivos de red con túneles ya establecidos por el Controlador de ONOS.

Para verificar que el dispositivo Netopeer ha interactuado con el Controlador ONOS, podemos consultar los registros. En ellos se refleja todos los mensajes NETCONF que se han intercambiado entre el dispositivo y el Controlador de ONOS. Éstos se encuentran en el archivo *cat /var/log/supervisor/netconf-stdout*.log* del Netopeer, tal como muestra Código 13.

Para validar nuestra configuración, en primer lugar debemos comprobar que realmente se han establecido las asociaciones de seguridad. Por un lado, podemos consultar la SPD de algún dispositivo y verificar que tenemos 2 entradas para cada dispositivo: una para la entrada de tráfico y otra para la salida, tal como muestra la Consola 1.

Comprobamos que efectivamente tenemos dos entradas por cada dispositivo, con el distintivo del orden de las IPs y el atributo **dir in** o **dir out**. Así pues, sabemos que las reglas indican que el túnel IPsec entre el netopeer_1 (192.169.0.3) y 192.169.0.2 es mediante el modo de transporte.

```
...
</ietf-ipsec>
</config>
</edit-config>
</rpc>
netopeer-server[59]: Merging the node ietf-ipsec (edit_config.c:2330)
netopeer-server[59]: Creating the node ietf-ipsec (edit_config.c:1492)
netopeer-server[59]: Deleting the node ietf-ipsec (edit_config.c:1008)
netopeer-server[59]: RelaxNG validation on subdatastore 1714636916
netopeer-server[59]: Schematron validation on subdatastore 1714636916
netopeer-server[59]: Transapi calling callback
↪ /A:ietf-ipsec/A:ipsec/A:sad/A:sad-entry with op ADD.
netopeer-server[59]: Transapi calling callback
↪ /A:ietf-ipsec/A:ipsec/A:sad with op CHAIN | ADD.
netopeer-server[59]: Transapi calling callback
↪ /A:ietf-ipsec/A:ipsec/A:spd/A:spd-entry with op ADD.
...
```

Código 13: Fragmento del registro de Netopeer.

```
/bin/bash
# docker exec tfg_netopeer_1 ip x policy
  src 192.169.0.3/32 dst 192.169.0.2/32
        dir out priority 0
        tmpl src 0.0.0.0 dst 0.0.0.0
                proto esp reqid 0 mode transport
  src 192.169.0.2/32 dst 192.169.0.3/32
        dir in priority 0
        tmpl src 0.0.0.0 dst 0.0.0.0
                proto esp reqid 0 mode transport
  ...
```

Consola 1: Contenido de la SPD para un dispositivo Netopeer.

Para saber qué tipo servicios IPsec ofrece para ese flujo de tráfico, consultamos la SAD. Al igual que en la SPD, existen dos entradas para el mismo host (una para entrada y otra para la salida), como podemos observar en la Consola 2. Esta nos indica que el tráfico utiliza los servicios IPsec de autenticación (*auth-trunc hmac(md5)*) y cifrado (*enc cbc(des3_ede)*)

es preciso generar tráfico entre los dispositivos Netopeer en el plano de datos (por ejemplo, el *ping* de la Consola 3) y observar el flujo de mensajes a través de algún software de análisis de protocolos, como Wireshark [59].

En la Figura 5.3 se muestra el flujo de paquetes cuando se realiza ping entre el host 1 y host 2. Al estar completamente protegidos los flujos de mensajes, implica que IPsec se ejecuta en modo transporte:

```
/bin/bash
# docker exec tfg_netopeer_1 ip x state
src 192.169.0.3 dst 192.169.0.2
        proto esp spi 0x00000001 reqid 0 mode transport
        replay-window 0
        auth-trunc hmac(md5) 0
           x617574680000000000000000000000000000000000 96
        enc cbc(des3_ede) 0
           x6563725f736563726574000000000000000000000000

        anti-replay context: seq 0x0, oseq 0x0, bitmap 0
           x00000000
        sel src 0.0.0.0/0 dst 0.0.0.0/0
src 192.169.0.2 dst 192.169.0.3
        proto esp spi 0x00000002 reqid 0 mode transport
        replay-window 0
        auth-trunc hmac(md5) 0
           x617574680000000000000000000000000000000000 96
        enc cbc(des3_ede) 0
           x6563725f736563726574000000000000000000000000

        anti-replay context: seq 0x0, oseq 0x0, bitmap 0
           x00000000
        sel src 0.0.0.0/0 dst 0.0.0.0/0
   ...
```

Consola 2: Contenido de la SPD para un dispositivo Netopeer.

| | | | | | |
|---|---|---|---|---|---|
| 2 | 0.000028968 | 192.169.0.1 | 192.169.0.255 | UDP | 313 53898 → 21027 Len=271 |
| 3 | 2.641584691 | 02:42:c0:a9:00:03 | Broadcast | ARP | 42 Who has 192.169.0.2? T |
| 4 | 2.641643623 | 02:42:c0:a9:00:02 | 02:42:c0:a9:00:03 | ARP | 42 192.169.0.2 is at 02:4 |
| 5 | 2.641660847 | 192.169.0.3 | 192.169.0.2 | ESP | 138 ESP (SPI=0x00000002) |
| 6 | 2.641783587 | 192.169.0.2 | 192.169.0.3 | ESP | 138 ESP (SPI=0x00000001) |
| 7 | 3.648287071 | 192.169.0.3 | 192.169.0.2 | ESP | 138 ESP (SPI=0x00000002) |
| 8 | 3.648399997 | 192.169.0.2 | 192.169.0.3 | ESP | 138 ESP (SPI=0x00000001) |
| 9 | 4.661575570 | 192.169.0.3 | 192.169.0.2 | ESP | 138 ESP (SPI=0x00000002) |
| 10 | 4.661689179 | 192.169.0.2 | 192.169.0.3 | ESP | 138 ESP (SPI=0x00000001) |

Figura 5.3: Intercambio protegido de mensajes Echo Request.

```
/bin/bash
# docker exec tfg_netopeer_1 ping -c4 192.169.0.2
PING 192.169.0.2 (192.169.0.2) 56(84) bytes of data.
64 bytes from 192.169.0.2: icmp_seq=1 ttl=64 time=0.181
   ms
64 bytes from 192.169.0.2: icmp_seq=2 ttl=64 time=0.221
   ms
64 bytes from 192.169.0.2: icmp_seq=3 ttl=64 time=0.157
   ms
64 bytes from 192.169.0.2: icmp_seq=4 ttl=64 time=0.125
   ms

--- 192.169.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time
   3031ms
rtt min/avg/max/mdev = 0.125/0.171/0.221/0.035 ms
```

Consola 3: Ping entre dos dispositivos Netopeer.

# Capítulo 6

# Conclusión y vías futuras

La realización de este trabajo aporta una solución a la necesidad de disponer de un Controlador que automatice la gestión de políticas de seguridad IPsec en entornos SDN. El desarrollo de este trabajo no solo cumple el objetivo de este Trabajo Fin de Grado, sino que aporta una implementación flexible y dinámica de todo el escenario planteado para el caso 2 de la línea de investigación [14]. Sin embargo, no es más que una pequeña demostración del potencial de un entorno SDN con problemas tales como la gestión de asociaciones de seguridad IPsec en una red compleja.

Un análisis de la plataforma de código libre ONOS y sus características ha sido realizado durante el desarrollo del proyecto. Podemos concluir que es una plataforma madura, con un amplio soporte de distintas formas de interacción con el Controlador, ya sea desde la Northbound API (GUI, CLI y API REST) o la Southbound API, tales como OpenFlow [8], NETCONF [7] o BGP [12].

No obstante, no todos los aspectos han sido tal como cabría esperar. Una de las principales carencias fue la documentación respecto a la interpretación y generación de mensajes NETCONF mediante modelos YANG con la SDK de ONOS. Esto ha dificultado bastante el desarrollo.

También cabe remarcar un driver NETCONF insuficiente, incapaz de gestionar la capa de mensaje (*message layer*) de NETCONF o la falta de descubrimento implícita de características y estado de un dispositivo NETCONF (puertos, enlaces con otros dispositivos o reglas de enrutamiento). Este último aspecto sí existe en OpenDayLight, analizado y explotado por otro compañero en su Trabajo Fin de Grado [15].

Además de las dificultades anteriores, que han limitado las capacidades del prototipo de solución, es importante destacar el esfuerzo invertido en tener una app y su completo escenario flexible y escalable. La facilidad de desplegar de forma virtualizada todo el proyecto con comandos, sin importar si queremos un escenario con 6 o 60 dispositivos, ha sido un aspecto a

considerar en todas las fases del desarrollo. Desde el diseño del escenario y la app, hasta su implantación y validación.

La continuidad de este trabajo se puede retomar por distintas vías acorde a la línea de investigación. El prototipo de la app descrita en este trabajo es una prueba de concepto básica que no explota todas las posibilidades que ofrece el SDK de ONOS. Además, otros aspectos muy interesantes para la línea de investigación quedaban fuera del ámbito de este trabajo. A continuación se planteará algunas posibles vías de investigación de cara al futuro partiendo del trabajo ya realizado:

- Pese a que el despliegue y la validación del caso 1 no fue posible por cuestiones de tiempo, nada indica que exista alguna incompatibilidad, ya sea desde la perspectiva de diseño como desde la implementación. Esto otorgaría una importante funcionalidad modular a la app, pudiendo el administrador definir si un determinado grupo de dispositivos deberían delegar en IKEv2 o ser gestionados por él mismo.
- El prototipo de solución en este proyecto se ha limitado a usar la estructura del modelo YANG *ietf-ipsec* con plantillas de mensajes NETCONF, sin realizar ninguna validación de los mensajes. Sería interesante hacer uso real del modelo YANG *ietf-ipsec* como núcleo de la app para el tratamiento de los mensajes NETCONF intercambiados entre el Controlador de ONOS y los dispositivos, sirviendo como utensilio las propias herramientas que proporciona ONOS para ello.
- Una interfaz gráfiaca (GUI) de la app integrada en la interfaz web de ONOS que permitiese controlar aspectos de IPsec al administrador. Esto representaría una clara separación entre el establecimiento de una política de seguridad y su aplicación mediante la creación de asociaciones de seguridad IPsec. Implícitamente supondría adaptar el driver NETCONF de ONOS para una correcta visualización de los puertos y enlaces de un dispositivo NETCONF.
- El flujo de mensajes entre la app y su entorno a través del Controlador de ONOS se basa únicamente en un evento concreto: la conexión de un dispositivo NETCONF. Esto puede ser suficiente para un escenario de laboratorio, pero no bajo una carga de trabajo real. El uso de notificaciones NETCONF permitiría reaccionar a diversos estímulos de un escenario, creando una implementación más robusta.
- La presencia de múltiples Controladores SDN gracias a la Eastbound/Westbound API permite exprimir escenarios realmente complejos, donde aparezcan nuevos factores como el balanceo de carga entre distintos Controladores o la comunicación distribuida que asegure la coherencia. Afrontar este reto no es sólo tarea del Controlador de ONOS, pues se extiende directamente a la app y su gestión de asociaciones de seguridad.

Estos planteamientos de futuras vías de desarrollo **no son excluyentes**. Una implementación completa y consolidada de una app que gestione políticas de seguridad IPsec en entornos SDN requiere de un gran trabajo en distintas áreas.

# Siglas

**ACL** Access Control List. *Glosario:* ACL, 16

**BGP** Border Gateway Protocol. *Glosario:* BGP, 17, 18, 60

**MPLS** Multiprotocol Label Switching. *Glosario:* MPLS, 17

**OSGi** Open Services Gateway initiative. *Glosario:* OSGi, 34

**OSPF** Open Shortest Path First. *Glosario:* OSPF, 17

**OVSDB** Open vSwitch Database Management. *Glosario:* OVSDB, 17

**RPC** Remote Procedure Call. *Glosario:* RPC, 20

# Glosario

**ACL** Es un concepto de seguridad informática usado para fomentar la separación de privilegios. Es una forma de determinar los permisos de acceso apropiados a un determinado objeto, dependiendo de ciertos aspectos del proceso que hace el pedido. 16

**Maven** Maven es una herramienta de compilación automática usada principalmente para proyectos Java [33].. 34, 44

**OpenFlow** Es una tecnología de switching que surgió a raíz del proyecto de Investigación: "OpenFlow: Enabling Innovation in Campus Networks" de 2008 en la Universidad de Stanford. Se define como un protocolo emergente y abierto de comunicaciones que permite a un servidor de software determinar el camino de reenvío de paquetes que debería seguir en una red de switches [8]. 17, 60

**OSGi** Es una tecnología de software distribuido que define una infraestructura extremadamente eficiente para el desarrollo de aplicaciones Java con el objetivo reducir la complejidad de construir, mantener, desplegar y gestionar el ciclo de vida de aplicaciones en cualquier tipo de dispositivo.. 34

**RPC** Técnica que utiliza el modelo cliente-servidor para ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambas.. 20

**Switch** Es el dispositivo digital lógico de interconexión de equipos que opera en la capa de enlace de datos del modelo OSI. Su función es interconectar dos o más host de manera similar a los puentes de red, pasando datos de un segmento a otro de acuerdo con la dirección MAC de destino de las tramas en la red y eliminando la conexión una vez finalizada ésta. 17

# Bibliografía


[1] A. Koshibe, R. Joyce, M. D. Leenheer, T. Vachuska, B. Varkonyi, and Y. Wang, "ONOS System Components." https://wiki.onosproject.org/display/ONOS/System+Components. Último acceso: 16/Jul/2018.

[2] "Docker documentation." https://www.docker.com/what-docker, 2008.

[3] W. Stallings, "Software-Defined Networks and OpenFlow," in *The Internet Protocol Journal*, 2013.

[4] ONOS, "ONOS - SDN controller platform." Último acceso: 13/Aug/2018.

[5] S. Frankel and S. Krishnan, "Ip security (ipsec) and internet key exchange (ike) document roadmap," RFC 6071, RFC Editor, February 2011. http://www.rfc-editor.org/rfc/rfc6071.txt.

[6] Cisco and/or its affiliates, "The Zettabyte Era: Trends and Analysis," tech. rep., Cisco, June 2017.

[7] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network configuration protocol (netconf)," RFC 6241, RFC Editor, June 2011. http://www.rfc-editor.org/rfc/rfc6241.txt.

[8] The Open Networking Foundation, "OpenFlow Switch Specification." https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, Mar. 2015.

[9] W3C, "Extensible Markup Language (XML) 1.0." https://www.w3.org/TR/xml/, Nov. 2008.

[10] M. Bjorklund, "Yang - a data modeling language for the network configuration protocol (netconf)," RFC 6020, RFC Editor, October 2010. http://www.rfc-editor.org/rfc/rfc6020.txt.

[11] D. Harkins and D. Carrel, "The internet key exchange (ike)," RFC 2409, RFC Editor, November 1998. http://www.rfc-editor.org/rfc/rfc2409.txt.

[12] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (bgp-4)," RFC 4271, RFC Editor, January 2006. http://www.rfc-editor.org/rfc/rfc4271.txt.

[13] G. Johnson, "How Many People Ever Lived?." http://blogs.discovermagazine.com/fire-in-the-mind/2013/08/11/how-many-people-ever-lived/, Aug. 2013.

[14] R. Lopez and G. Lopez-Millan, "Software-defined networking (sdn)-based ipsec flow protection," Internet-Draft draft-ietf-i2nsf-sdn-ipsec-flow-protection-02, IETF Secretariat, July 2018. http://www.ietf.org/internet-drafts/draft-ietf-i2nsf-sdn-ipsec-flow-protection-02.txt.

[15] A. M. Palazón, "Estudio de la plataforma OpenDayLight para la Gestión de asociaciones de seguridad IPsec en entornos SDN," resreport, Facultad de Informática de la Universidad de Murcia, June 2018.

[16] W. Stallings, F. Agboma, and S. Jelassi, *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*. The William Stallings books on computer and data communicaitons technology, Pearson, 2015.

[17] IETF, "Network Functions Virtualisation – Introductory White Paper." https://portal.etsi.org/nfv/nfv_white_paper.pdf, Oct. 2012.

[18] "Open Network Foundation." https://www.opennetworking.org/. Último acceso: 22/Aug/2018.

[19] J. Moy, "Ospf version 2," STD 54, RFC Editor, April 1998. http://www.rfc-editor.org/rfc/rfc2328.txt.

[20] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," RFC 3031, RFC Editor, January 2001. http://www.rfc-editor.org/rfc/rfc3031.txt.

[21] B. Pfaff and B. Davie, "The open vswitch database management protocol," RFC 7047, RFC Editor, December 2013. http://www.rfc-editor.org/rfc/rfc7047.txt.

[22] H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi, "Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains," Internet-Draft draft-yin-sdn-sdni-00, IETF Secretariat, June 2012. http://www.ietf.org/internet-drafts/draft-yin-sdn-sdni-00.txt.

[23] J. Vasseur and J. L. Roux, "Path computation element (pce) communication protocol (pcep)," RFC 5440, RFC Editor, March 2009. http://www.rfc-editor.org/rfc/rfc5440.txt.

[24] M. Massé, *REST API design rulebook*. O'Reilly, 2012.

[25] "Internet Engineering Task Force." https://www.ietf.org. Último acceso: 11/Aug/2018.

[26] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet key exchange protocol version 2 (ikev2)," STD 79, RFC Editor, October 2014. http://www.rfc-editor.org/rfc/rfc7296.txt.

[27] S. Hares, D. Lopez, M. Zarny, C. Jacquenet, R. Kumar, and J. Jeong, "Interface to network security functions (i2nsf): Problem statement and use cases," RFC 8192, RFC Editor, July 2017.

[28] Market Research Future, "Cloud Application Security Market Research Report - Global Forecast to 2023," *Market Research Future*, Dec. 2017.

[29] I. M. Alpiste, "Gestión de Políticas IPsec e IKE en entornos SDN," resreport, Facultad de Informática de la Universidad de Murcia, June 2017.

[30] R. R. Sánchez, "Establecimiento de canales seguros IPsec en entornos SDN usando PF KEYv2," tech. rep., Facultad de Informática de la Universidad de Murcia, 2017.

[31] "OSGi™ Alliance – The Dynamic Module System for Java." https://www.osgi.org/. Último acceso: 20/Aug/2018.

[32] A. Koshibe, S. Hunt, and E. Olkhovskaya, "ONOS : An Overview." https://wiki.onosproject.org/display/ONOS/ONOS+%3A+An+Overview. Último acceso: 15/Jul/2018.

[33] "Maven)." https://maven.apache.org/. Último acceso: 25/Jul/2018.

[34] "Apache Karaf - The enterprise class platform." https://karaf.apache.org/. Último acceso: 20/Aug/2018.

[35] D. Erickson, "The beacon openflow controller," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, (New York, NY, USA), pp. 13–18, ACM, 2013.

[36] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *Proc. IEEE Network Operations and Management Symp. (NOMS)*, pp. 1–4, May 2014.

[37] "Floodlight OpenFlow Controller." http://www.projectfloodlight.org/floodlight/. Último acceso: 13/Aug/2018.

[38] "HP Virtual Application Networks SDN Controller." http://h17007.www1.hpe.com/docs/networking/solutions/sdn/4AA4-8807ENW.PDF. Último acceso: 25/Aug/2018.

[39] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, (New York, NY, USA), pp. 19–24, ACM, 2012.

[40] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, (Berkeley, CA, USA), pp. 351–364, USENIX Association, 2010.

[41] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A System for Scalable OpenFlow Control," tech. rep., Rice University, 2011.

[42] "Open MUL SDN Controller - High Performance SDN Platform." http://www.openmul.org/devdoc-center.html. Último acceso: 27/Aug/2018.

[43] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 105–110, July 2008.

[44] "OpenContrail - An open-source network virtualization platform for the cloud." http://www.opencontrail.org/. Último acceso: 13/Aug/2018.

[45] OpenDaylight, "OpenDaylight: A Linux Foundation Collaborative Project," 2013. Último acceso: 13/Aug/2018.

[46] "POX SDN Controller." https://noxrepo.github.io/pox-doc/html/. Último acceso: 13/Aug/2018.

[47] "RYU the Network Operating System(NOS)." https://ryu.readthedocs.io/en/latest/index.html. Último acceso: 13/Aug/2018.

[48] F. Botelho, A. Bessani, F. M. V. Ramos, and P. Ferreira, "SMaRtLight: A Practical Fault-Tolerant SDN Controller," *CoRR*, vol. abs/1407.6062, 2014.

[49] "Trema SDN Controller." https://trema.github.io/trema/. Último acceso: 13/Aug/2018.

[50] CESNET, "NETCONF toolset Netopeer." https://github.com/CESNET/netopeer. Último acceso: 08/May/2018.

[51] A. Campanella and ON.Lab, "NETCONF and YANG integration in ONOS Southbound Interface." https://events17.linuxfoundation.org/sites/events/files/slides/Netconf%20and%20Yang%20ONOS%20SB.pdf, 2016.

[52] M. Wasserman, "Using the netconf protocol over secure shell (ssh)," RFC 6242, RFC Editor, June 2011. http://www.rfc-editor.org/rfc/rfc6242.txt.

[53] "ONOS Template Application Tutorial." https://wiki.onosproject.org/display/ONOS/Template+Application+Tutorial. Último acceso: 13/May/2018.

[54] "ONOS NETCONF." https://wiki.onosproject.org/display/ONOS/NETCONF. Último acceso: 25/Apr/2018.

[55] "Pyang - YANG validator." https://github.com/mbj4668/pyang/wiki. Último acceso: 03/Jul/2018.

[56] "ONOS Southbound Yang Utils." https://wiki.onosproject.org/display/ONOS/Southbound+Yang+Utils. Último acceso: 02/Aug/2018.

[57] "Modelo YANG de ietf-interfaces." http://www.netconfcentral.org/modules/ietf-interfaces. Último acceso: 29/Aug/2018.

[58] D. Inc., "Docker." https://www.docker.com/. Último acceso: 01/Sep/2018.

[59] W. Foundation, "Wireshark." https://www.wireshark.org/. Último acceso: 14/Jun/2018.

[60] "Docker wiki." https://es.wikipedia.org/wiki/Docker_(software), 2008.

[61] I. döt Net, C. C. Evans, and O. Ben-Kiki, "YAML Ain't Markup Language (YAML™) Version 1.2." http://yaml.org/spec/1.2/spec.html, Oct. 2009.

# Apéndice A

# Proyecto onos-ipsec

En este mismo PDF se adjunta un archivo .tar.xz de todo el proyecto desarrollado.

Una copia de este proyecto se encuentra en el repositorio git: https://gitlab.atica.um.es/valiantsin.kivachuk/onos-ipsec

Podemos obtener todo el proyecto ejecutando el siguiente comando:

```
/bin/bash
git clone --recursive https://gitlab.atica.um.es/
    valiantsin.kivachuk/onos-ipsec
```

# Apéndice B

# Estructura ietf-ipsec

```
module: ietf-ipsec
    +--rw ietf-ipsec
       +--rw ikev2 {case1}?
       |  +--rw ike-connection
       |     +--rw ike-conn-entries* [conn-name]
       |        +--rw conn-name          string
       |        +--rw autostartup        boolean
       |        +--rw nat-traversal?     boolean
       |        +--rw version?           enumeration
       |        +--rw phase1-lifetime    uint32
       |        +--rw phase1-authby      auth-method-type
       |        +--rw phase1-authalg*    integrity-algorithm-t
       |        +--rw phase1-encalg*     encryption-algorithm-t
       |        +--rw dh_group           uint32
       |        +--rw local
       |        |  +--rw (my-identifier-type)?
       |        |  |  +--:(ipv4)
       |        |  |  |  +--rw ipv4?            inet:ipv4-address
       |        |  |  +--:(ipv6)
       |        |  |  |  +--rw ipv6?            inet:ipv6-address
       |        |  |  +--:(fqdn)
       |        |  |  |  +--rw fqdn?            inet:domain-name
       |        |  |  +--:(dn)
       |        |  |  |  +--rw dn?              string
       |        |  |  +--:(user_fqdn)
       |        |  |     +--rw user_fqdn?       string
       |        |  +--rw my-identifier    string
       |        +--rw remote
       |        |  +--rw (my-identifier-type)?
       |        |  |  +--:(ipv4)
       |        |  |  |  +--rw ipv4?            inet:ipv4-address
       |        |  |  +--:(ipv6)
       |        |  |  |  +--rw ipv6?            inet:ipv6-address
       |        |  |  +--:(fqdn)
       |        |  |  |  +--rw fqdn?            inet:domain-name
       |        |  |  +--:(dn)
       |        |  |  |  +--rw dn?              string
       |        |  |  +--:(user_fqdn)
       |        |  |     +--rw user_fqdn?       string
```

```
|        |  +--rw my-identifier    string
|        +--rw local-addrs        inet:ip-address
|        +--rw remote-addr        inet:ip-address
|        +--rw pfs_group?         uint32
|        +--rw phase2-lifetime    uint32
|        +--rw phase2-authalg*    integrity-algorithm-t
|        +--rw phase2-encalg*     encryption-algorithm-t
+--rw ipsec
   +--rw spd
   |  +--rw spd-entry* [rule-number]
   |     +--rw rule-number        uint64
   |     +--rw priority?          uint32
   |     +--rw names* [name]
   |     |  +--rw name-type?   ipsec-spd-name
   |     |  +--rw name         string
   |     +--rw condition
   |     |  +--rw traffic-selector-list* [ts-number]
   |     |     +--rw ts-number              uint32
   |     |     +--rw direction?             ipsec-traffic-direction
   |     |     +--rw local-addresses* [start end]
   |     |     |  +--rw start    inet:ip-address
   |     |     |  +--rw end      inet:ip-address
   |     |     +--rw remote-addresses* [start end]
   |     |     |  +--rw start    inet:ip-address
   |     |     |  +--rw end      inet:ip-address
   |     |     +--rw next-layer-protocol*   ipsec-next-layer-proto
   |     |     +--rw local-ports* [start end]
   |     |     |  +--rw start    inet:port-number
   |     |     |  +--rw end      inet:port-number
   |     |     +--rw remote-ports* [start end]
   |     |     |  +--rw start    inet:port-number
   |     |     |  +--rw end      inet:port-number
   |     |     +--rw selector-priority?     uint32
   |     +--rw processing-info
   |     |  +--rw action          ipsec-spd-operation
   |     |  +--rw ipsec-sa-cfg
   |     |     +--rw pfp-flag?            boolean
   |     |     +--rw extSeqNum?           boolean
   |     |     +--rw seqOverflow?         boolean
   |     |     +--rw statefulfragCheck?   boolean
   |     |     +--rw security-protocol?   ipsec-protocol
   |     |     +--rw mode?                ipsec-mode
   |     |     +--rw ah-algorithms
   |     |     |  +--rw ah-algorithm*   integrity-algorithm-t
   |     |     +--rw esp-algorithms
   |     |     |  +--rw authentication*   integrity-algorithm-t
   |     |     |  +--rw encryption*       encryption-algorithm-t
   |     |     +--rw tunnel
   |     |        +--rw local?          inet:ip-address
   |     |        +--rw remote?         inet:ip-address
   |     |        +--rw bypass-df?      boolean
   |     |        +--rw bypass-dscp?    boolean
   |     |        +--rw dscp-mapping?   yang:hex-string
   |     |        +--rw ecn?            boolean
   |     +--rw spd-lifetime
   |        +--rw time-soft?       uint32
   |        +--rw time-hard?       uint32
```

```
|        +--rw time-use-soft?   uint32
|        +--rw time-use-hard?   uint32
|        +--rw byte-soft?       uint32
|        +--rw byte-hard?       uint32
|        +--rw packet-soft?     uint32
|        +--rw packet-hard?     uint32
+--rw sad {case2}?
|  +--rw sad-entry* [spi]
|     +--rw spi                          ipsec-spi
|     +--rw seq-number?                  uint64
|     +--rw seq-number-overflow-flag?    boolean
|     +--rw anti-replay-window?          uint16
|     +--rw rule-number?                 uint32
|     +--rw local-addresses* [start end]
|     |  +--rw start    inet:ip-address
|     |  +--rw end      inet:ip-address
|     +--rw remote-addresses* [start end]
|     |  +--rw start    inet:ip-address
|     |  +--rw end      inet:ip-address
|     +--rw next-layer-protocol*         ipsec-next-layer-proto
|     +--rw local-ports* [start end]
|     |  +--rw start    inet:port-number
|     |  +--rw end      inet:port-number
|     +--rw remote-ports* [start end]
|     |  +--rw start    inet:port-number
|     |  +--rw end      inet:port-number
|     +--rw security-protocol?           ipsec-protocol
|     +--rw ah-sa
|     |  +--rw integrity-algorithm?   integrity-algorithm-t
|     |  +--rw key?                   string
|     +--rw esp-sa
|     |  +--rw encryption
|     |  |  +--rw encryption-algorithm?   encryption-algorithm-t
|     |  |  +--rw key?                    string
|     |  |  +--rw iv?                     string
|     |  +--rw integrity
|     |  |  +--rw integrity-algorithm?   integrity-algorithm-t
|     |  |  +--rw key?                   string
|     |  +--rw combined
|     |     +--rw combined-algorithm?   combined-algorithm-t
|     +--rw sa-lifetime
|     |  +--rw time-soft?       uint32
|     |  +--rw time-hard?       uint32
|     |  +--rw time-use-soft?   uint32
|     |  +--rw time-use-hard?   uint32
|     |  +--rw byte-soft?       uint32
|     |  +--rw byte-hard?       uint32
|     |  +--rw packet-soft?     uint32
|     |  +--rw packet-hard?     uint32
|     |  +--rw action?          lifetime-action
|     +--rw mode?                        ipsec-mode
|     +--rw statefulfragCheck?           boolean
|     +--rw dscp?                        yang:hex-string
|     +--rw tunnel
|     |  +--rw local?          inet:ip-address
|     |  +--rw remote?         inet:ip-address
|     |  +--rw bypass-df?      boolean
```

```
|          |     |  +--rw bypass-dscp?    boolean
|          |     |  +--rw dscp-mapping?   yang:hex-string
|          |     |  +--rw ecn?            boolean
|          |     +--rw path-mtu?                  uint16
|          |     +--rw encap
|          |        +--rw espinudp?   boolean
|          |        +--rw sport?      inet:port-number
|          |        +--rw dport?      inet:port-number
|          |        +--rw oddr?       inet:ip-address
|          +--rw pad {case1}?
|             +--rw pad-entries* [pad-entry-id]
|                +--rw pad-entry-id             uint64
|                +--rw (identity)?
|                |  +--:(ipv4-address)
|                |  |  +--rw ipv4-address?            inet:ipv4-address
|                |  +--:(ipv6-address)
|                |  |  +--rw ipv6-address?            inet:ipv6-address
|                |  +--:(fqdn-string)
|                |  |  +--rw fqdn-string?             inet:domain-name
|                |  +--:(rfc822-address-string)
|                |  |  +--rw rfc822-address-string?   string
|                |  +--:(dnX509)
|                |  |  +--rw dnX509?                  string
|                |  +--:(id_key)
|                |     +--rw id_key?                  string
|                +--rw pad-auth-protocol?       auth-protocol-type
|                +--rw auth-method
|                   +--rw auth-m?          auth-method-type
|                   +--rw pre-shared
|                   |  +--rw secret?   string
|                   +--rw rsa-signature
|                      +--rw key-data?    string
|                      +--rw key-file?    string
|                      +--rw ca-data*     string
|                      +--rw ca-file?     string
|                      +--rw cert-data?   string
|                      +--rw cert-file?   string
|                      +--rw crl-data?    string
|                      +--rw crl-file?    string
|
|  rpcs:
|    +---x sadb_register
|       +---w input
|       |  +---w base-list* [version]
|       |     +---w version       string
|       |     +---w msg_type?     sadb-msg-type
|       |     +---w msg_satype?   sadb-msg-satype
|       |     +---w msg_seq?      uint32
|       +--ro output
|          +--ro base-list* [version]
|          |  +--ro version       string
|          |  +--ro msg_type?     sadb-msg-type
|          |  +--ro msg_satype?   sadb-msg-satype
|          |  +--ro msg_seq?      uint32
|          +--ro algorithm-supported*
|             +--ro authentication
|             |  +--ro name?       integrity-algorithm-t
```

```
211 |              |  +--ro ivlen?      uint8
212 |              |  +--ro min-bits?   uint16
213 |              |  +--ro max-bits?   uint16
214 |              +--ro encryption
215 |                 +--ro name?       encryption-algorithm-t
216 |                 +--ro ivlen?      uint8
217 |                 +--ro min-bits?   uint16
218 |                 +--ro max-bits?   uint16
219 |
220 |   notifications:
221 |     +---n spd-expire
222 |     |  +--ro index?   uint64
223 |     +---n sadb_acquire
224 |     |  +--ro state    uint32
225 |     +---n sadb_expire
```

# Apéndice C

# Configuración NETCONF basada en ietf-ipsec

```
<ietf-ipsec xmlns="http://example.net/ietf-ipsec"
↪  xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ipsec nc:operation="merge">
    <spd>
      <spd-entry>
        <rule-number>1</rule-number>
        <priority>0</priority>
        <names>
          <name>in/netconf:10.100.3.2:830/192.169.0.3</name>
        </names>
        <condition>
          <traffic-selector-list>
            <ts-number>102</ts-number>
            <direction>INBOUND</direction>
            <local-addresses>
              <start>192.169.0.2</start>
              <end>192.169.0.2</end>
            </local-addresses>
            <remote-addresses>
              <start>192.169.0.3</start>
              <end>192.169.0.3</end>
            </remote-addresses>
            <next-layer-protocol>TCP</next-layer-protocol>
            <local-ports>
              <start>0</start>
              <end>0</end>
            </local-ports>
            <remote-ports>
              <start>0</start>
              <end>0</end>
            </remote-ports>
          </traffic-selector-list>
        </condition>
        <processing-info>
          <action>PROTECT</action>
          <ipsec-sa-cfg>
            <security-protocol>esp</security-protocol>
```

```
            <mode>TRANSPORT</mode>
          </ipsec-sa-cfg>
        </processing-info>
      </spd-entry>
      <spd-entry>
        <rule-number>2</rule-number>
        <priority>0</priority>
        <names>
          <name>out/192.169.0.3/netconf:10.100.3.2:830</name>
        </names>
        <condition>
          <traffic-selector-list>
            <ts-number>102</ts-number>
            <direction>OUTBOUND</direction>
            <local-addresses>
              <start>192.169.0.3</start>
              <end>192.169.0.3</end>
            </local-addresses>
            <remote-addresses>
              <start>192.169.0.2</start>
              <end>192.169.0.2</end>
            </remote-addresses>
            <next-layer-protocol>TCP</next-layer-protocol>
            <local-ports>
              <start>0</start>
              <end>0</end>
            </local-ports>
            <remote-ports>
              <start>0</start>
              <end>0</end>
            </remote-ports>
          </traffic-selector-list>
        </condition>
        <processing-info>
          <action>PROTECT</action>
          <ipsec-sa-cfg>
            <security-protocol>esp</security-protocol>
            <mode>TRANSPORT</mode>
          </ipsec-sa-cfg>
        </processing-info>
      </spd-entry>
    </spd>
    <sad>
      <sad-entry>
        <spi>1</spi>
        <rule-number>1</rule-number>
        <local-addresses>
          <start>192.169.0.2</start>
          <end>192.169.0.2</end>
        </local-addresses>
        <remote-addresses>
          <start>192.169.0.3</start>
          <end>192.169.0.3</end>
        </remote-addresses>
        <next-layer-protocol>TCP</next-layer-protocol>
        <local-ports>
          <start>0</start>
```

```
          <end>0</end>
        </local-ports>
        <remote-ports>
          <start>0</start>
          <end>0</end>
        </remote-ports>
        <security-protocol>esp</security-protocol>
        <esp-sa>
          <encryption>
            <encryption-algorithm>3des</encryption-algorithm>
            <key>ecr_secret</key>
            <iv>vector</iv>
          </encryption>
          <integrity>
            <integrity-algorithm>hmac-md5-128</integrity-algorithm>
            <key>auth</key>
          </integrity>
        </esp-sa>
        <mode>TRANSPORT</mode>
      </sad-entry>
      <sad-entry>
        <spi>2</spi>
        <rule-number>1</rule-number>
        <local-addresses>
          <start>192.169.0.3</start>
          <end>192.169.0.3</end>
        </local-addresses>
        <remote-addresses>
          <start>192.169.0.2</start>
          <end>192.169.0.2</end>
        </remote-addresses>
        <next-layer-protocol>TCP</next-layer-protocol>
        <local-ports>
          <start>0</start>
          <end>0</end>
        </local-ports>
        <remote-ports>
          <start>0</start>
          <end>0</end>
        </remote-ports>
        <security-protocol>esp</security-protocol>
        <esp-sa>
          <encryption>
            <encryption-algorithm>3des</encryption-algorithm>
            <key>ecr_secret</key>
            <iv>vector</iv>
          </encryption>
          <integrity>
            <integrity-algorithm>hmac-md5-128</integrity-algorithm>
            <key>auth</key>
          </integrity>
        </esp-sa>
        <mode>TRANSPORT</mode>
      </sad-entry>
    </sad>
  </ipsec>
</ietf-ipsec>
```

# Apéndice D

# Docker


Figura D.1: Logo Docker. [2]

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo en Linux [60]. Gracias a ello, es posible ejecutar de forma virtualizada distinto software usando directamente el sistema operativo del host en vez de una completa máquina virtual.

La primera parte para ejecutar software en contenedores es definir un archivo Dockerfile. En él se explican todos los pasos a seguir para crear una *imagen*, que es la que finalmente se ejecuta dentro del *contenedor*. Por tanto, la imagen es algo estático que se crea a partir de la definición de una serie de comandos, mientras que el contenedor es un entorno aislado para poder ejecutar la aplicación, pudiendo crearse y eliminarse indefinidamente.

Para desplegar el escenario completo y gestionar los distintos contenedores de una forma más sencilla, se hace uso de Docker Compose. Docker Compose es una herramienta dentro del abanico de Docker que nos permite definir y ejecutar escenarios multicontenedores. Haciendo uso del lenguaje de marcado YAML [61], es posible definir las distintas imágenes, sus características y la relación entre ellos.

Como podemos observar en el Código 14 referente a nuestro escenario, además de las dos imágenes docker, la del Controlador ONOS *onos* y Netopeer *netopeer*, definimos las subredes que se usan en el escenario. A ambas imágenes les establecemos distintos parámtros, como los puertos de escucha, creedenciales de acceso, etc. Esto no implica que sólo exista una imagen de cada tipo en el escenario, pues como ya hemos dicho antes, las imágenes se lanzan dentro de contenedores. De esta forma, escalar nuestro escenario es muy sencillo: lanzar 30 dispositivos NETCONF no es más que indicar que queremos ejecutar la imagen *netopeer* en 30 contenedores distintos.

```
version: '3'
services:

  onos:
    build: base/onos
    environment:
        - ONOS_APPS='netconf, drivers.netconf, netconfsb, yang-gui,
        ↪  models.common, restconf, protocols.restconfserver'
    env_file:
      - "onos.env"
    ports:
        -   8181:8181
    networks:
        sdn_control:
            ipv4_address: 10.100.3.204
  netopeer:
    build: base/netopeer
    cap_add:
      - SYS_ADMIN
      - NET_ADMIN
    environment:
      - NETCONF_USER=root
      - NETCONF_PASSWORD=root
      - SDN_NET=10.100.3.0/24
    env_file:
      - "onos.env"
    depends_on:
      - "onos"
    networks:
      - sdn_control
      - sdn_data

networks:
  sdn_control:
    driver: bridge
    ipam:
     config:
       - subnet: 10.100.3.0/24
    driver_opts:
     com.docker.network.bridge.name: sdn_control
  sdn_data:
    driver: bridge
    ipam:
     config:
       - subnet: 192.169.0.0/24
    driver_opts:
     com.docker.network.bridge.name: sdn_data
```

Código 14: Docker compose del proyecto onos-ipsec