eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO Y DESPLIEGUE DE UN SISTEMA DE DETECCIÓN DE INTRUSIÓN EN REDES DEFINIDAS POR SOFTWARE

Alumno: Orbegozo, Aldalur, Iban

Directora: Higuero, Aperribay, María Victoria

Director: Jacob, Taquet, Eduardo Juan

Curso: 2017-2018

Fecha: Bilbao, 18 de junio de 2018

Resumen

Las aplicaciones de red actuales tienen requisitos crecientes en términos de ancho de banda, escalabilidad y seguridad. Las Redes Definidas por Software (en inglés *Software-Defined Networking*, SDN) componen una arquitectura emergente que permite cumplir estos requisitos, separando el plano de control del plano de datos y centralizando la inteligencia de red en controladores SDN. Sin embargo, la seguridad de red sigue siendo la principal preocupación, ya que aparecen nuevos tipos de ataques a diario. Este estudio se centra en el uso de Sistemas de Detección de Intrusión (en inglés *Intrusion Detection System*, IDS) como una medida de seguridad novedosa en redes SDN. Utilizar un IDS le permite al administrador de red localizar e identificar actividades maliciosas en el sistema mediante la monitorización del tráfico de red en tiempo real. Además, la arquitectura centralizada de los controladores SDN permite programar la respuesta a los ataques de antemano, de modo que toda la red reaccione de la forma que se desee en cuanto se detecte una amenaza.

Gaur egungo sare-aplikazioek gero eta eskakizun gehiago dituzte banda-zabalera, hazkundea eta segurtasunaren aldetik. Software bidez Definitutako Sareek (ingelesez *Software-Defined Networking,* SDN) eskakizun hauek betetzea ahalbidetzen duten arkitektura bat osatzen dute, kontrol-planoa eta datu-planoa banatuz eta sare-adimena SDN kontrolagailuetan bilduz. Dena den, sareen segurtasuna da ardurarik handiena oraindik, egunero eraso berriak agertzen direlarik. Ikerketa honetan, Intrusioak Detektatzeko Sistemen (ingelesez *Intrusion Detection System*, IDS) erabilera proposatzen da SDN sareetarako segurtasun-neurri berritzaile gisa. IDS bat erabiltzeak sareko trafikoa denbora errealean aztertzeko aukera ematen dio sare-administratzaileari, jarduera maltzurrak aurkitu eta identifikatzeko. Gainera, erasoen aurkako erantzuna aldez aurretik programatu daiteke sare guztiak mehatxua detektatu bezain laster nahi bezala erantzun dezan, SDN kontrolagailuen arkitektura zentralizatuari esker.

Modern network applications have growing requirements in terms of bandwidth, scalability and security. Software-Defined Networking (SDN) is an emerging architecture that allows to meet these requirements, decoupling the control plane from the data plane and centralizing the network intelligence on SDN controllers. However, network security is still the main concern, as new types of attacks appear on a daily basis. The following research focuses on the use of Intrusion Detection Systems (IDS) as an innovative security measure on SDN networks. Using an IDS gives the network administrator the ability to monitor network traffic in real time, and therefore to locate and identify malicious activity in the system. Furthermore, the centralized architecture of the SDN controllers enables to program the response to the attack beforehand, so that all the network reacts as wanted as soon as a threat is detected.

Palabras clave

Sistema de Detección de Intrusión (IDS); Redes Definidas por Software (SDN); Ciberseguridad; OpenFlow

Índice de contenido

Índice de ilustraciones

Índice de tablas

Índice de gráficos

Índice de acrónimos

SDN	*Software-Defined Networking*, Redes Definidas por Software
IDS	*Intrusion Detection System*, Sistema de detección de intrusiones
IPS	*Intrusion Prevention System*, Sistema de prevención de intrusiones
OF	*OpenFlow*
GUI	*Graphical User Interface*, Interfaz gráfica de usuario
STP	*Spanning Tree Protocol*, Protocolo del árbol de expansión
API	*Application Programming Interface*, Interfaz de programación de aplicaciones
CLI	*Command Line Interface*, Interfaz de línea de comandos
IP	*Internet Protocol*
ICMP	*Internet Control Message Protocol*
MAC	*Media Access Control*
BPDU	*Bridge Protocol Data Unit*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
HTTP	*HyperText Transfer Protocol*
QoS	*Quality of Service*, Calidad de Servicio
WBS	*Work Breakdown Structure*, Estructura de descomposición del trabajo
BSD	*Berkeley Software Distribution*
GNU	Acrónimo recursivo de *GNU is not Unix*
GPL	*General Public License*, Licencia Pública General

1 Introducción

El diseño de las redes tradicionales hace tremendamente difícil satisfacer los requerimientos actuales de usuarios, empresas y proveedores. La cada vez mayor relevancia de los dispositivos móviles, la virtualización de los equipos de red o los servicios en la nube se sitúan entre los factores que impulsan al sector de las telecomunicaciones a replantear las arquitecturas de red convencionales.

Con el objetivo de proporcionar una respuesta a esta situación, surgen las Redes Definidas por Software (SDN). Las redes SDN presentan una arquitectura de red dinámica, gestionable y adaptable, lo que las hace adecuadas para la naturaleza dinámica de las aplicaciones actuales. La principal diferencia entre estas redes y las tradicionales es la separación entre las funciones de control y las de reenvío, lo que permite programar directamente sobre el plano de control y abstraerse del plano de datos. Esta característica proporciona una serie de ventajas a las redes SDN: independencia respecto a los fabricantes, visión y control global y centralizado de las redes, programación y automatización de servicios… En resumen, hace posible configurar y gestionar libremente todos los recursos de la red, simplificando su despliegue y reduciendo costes.

La variedad de posibilidades y promesas que ofrecen las redes SDN han hecho que grandes compañías en el ámbito de las telecomunicaciones, como Google [1], hayan apostado por esta tecnología, migrando sus redes tradicionales a redes SDN con el propósito de mejorar prestaciones y reducir costes. Sin embargo, siendo las SDNs un concepto relativamente novedoso, todavía queda mucho camino por recorrer en aspectos importantes como la seguridad de red.

Es en este contexto de seguridad aplicada a las redes SDN donde surge la necesidad de desarrollo de este TFG, que busca precisamente incorporar un sistema de seguridad a estas redes. Más concretamente, este trabajo se centra en la aplicación de la estrategia IDS sobre este innovador escenario, buscando aprovechar los beneficios de este tipo de sistemas, pero adaptándolas a las especificidades de las SDNs. Este TFG ha sido desarrollado en el grupo de investigación I2T, en el marco de sus trabajos en el área de las redes SDN.

2 Contexto

2.1 Tecnologías relacionadas

2.1.1 Redes Definidas por Software (SDN)

En las redes tradicionales, los dispositivos de la red tienen opciones de configuración limitadas, lo que supone un obstáculo a la hora de realizar cambios significativos en los protocolos o en las tecnologías utilizadas. Las redes SDN, en cambio, ofrecen una abstracción de las funcionalidades de nivel inferior, permitiendo así una mayor flexibilidad en el control de la red [2].

La característica principal de las redes SDN es la separación entre las funciones de control de la red (plano de control) y las funciones relacionadas con los paquetes de datos (plano de datos).

El plano de datos cubre todas las funciones relacionadas con el envío de paquetes, como la transmisión, la fragmentación, el reensamblado o la replicación. El plano de control define la lógica de los equipos de red (routers o switches), que determina como se realiza la comunicación entre los diferentes dispositivos. Esta lógica incluye todas las tareas necesarias para operar en el plano de datos, pero sin involucrar los paquetes de los usuarios: construcción de las tablas de rutado, filtrado de paquetes, comunicación entre los dispositivos de la red…

La Ilustración 1 muestra una representación lógica de la arquitectura SDN. La inteligencia (lógica) de la red está centralizada en controladores SDN, que mantienen una visión global de la red. Para la comunicación entre el plano de control y el plano de datos, uno de los protocolos más utilizados es el protocolo OpenFlow. OpenFlow es el primer estándar de interfaz de comunicaciones que se diseñó para gestionar las comunicaciones entre los dispositivos de la red y el controlador.

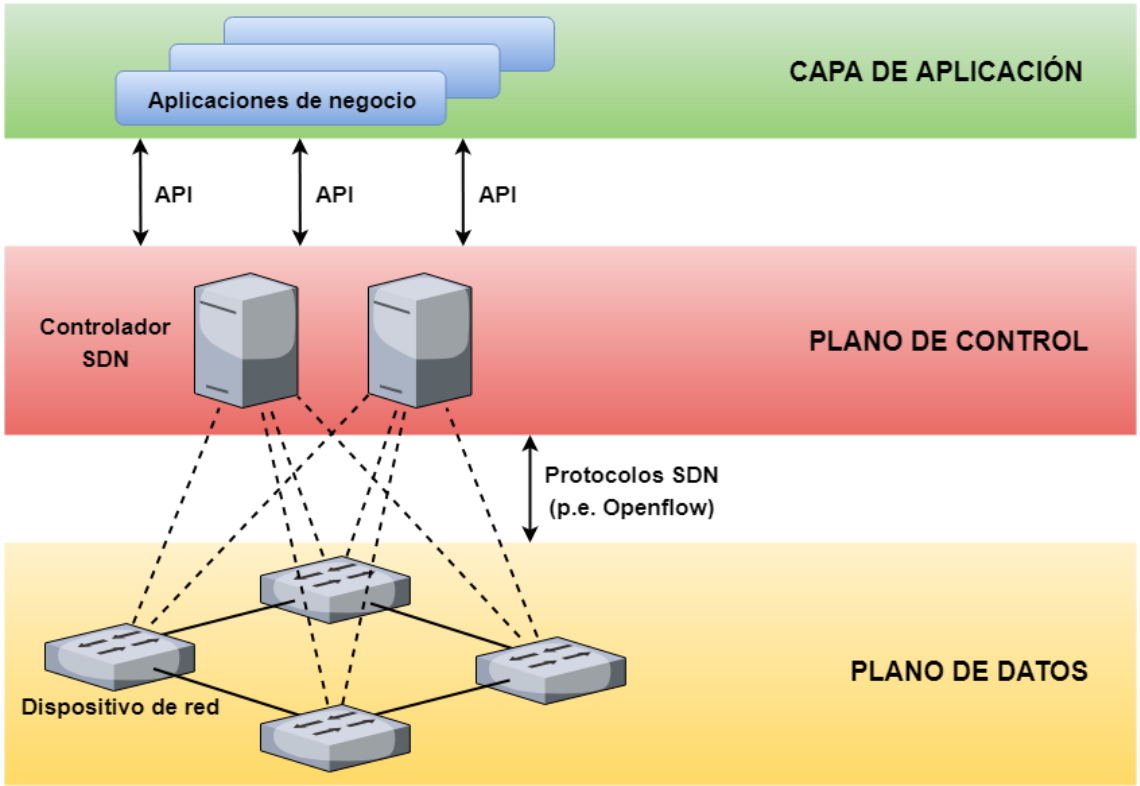

Ilustración 1.- Arquitectura SDN

2.1.2 OpenFlow

OpenFlow es un protocolo de comunicación que proporciona acceso al plano de datos de un switch de una red [3]. Este protocolo permite a los controladores gestionar de manera remota la configuración de los switches, separando así el plano de datos del plano de control. De esta forma, es posible realizar una gestión más avanzada y eficiente del tráfico de red que con los protocolos de rutado tradicionales.

La característica principal de OpenFlow es que la decisión de reenvío de un paquete pasa de estar a cargo del switch a ser gestionado por un controlador. La especificación de OpenFlow define el protocolo de comunicación entre el switch y el controlador, así como un conjunto de operaciones que pueden llevar a cabo entre ellos. El protocolo OpenFlow, tal y como indica su nombre, se basa en flujos. Las instrucciones de reenvío que reciben los switches del controlador se basan en estos flujos, que representan todos los paquetes que comparten una serie de características comunes.

Para poder trabajar con los flujos de OpenFlow, los switches emplean las denominadas *flow-tables* o tablas de flujo. Cada entrada de estas tablas indica la acción que debe realizar el switch para un flujo definido o, lo que es lo mismo, para todos los paquetes que coincidan con dicha entrada. Las acciones más básicas son el reenvío de paquetes a un puerto dado (pueden ser varios puertos), el reenvío del paquete al controlador y el descarte del paquete. El formato de las entradas de las tablas de flujo puede descomponerse en tres segmentos principales:

- **Regla:** la regla que determina la coincidencia de un flujo con la entrada
- **Acción:** la acción que debe realizar el switch en caso de coincidencia
- **Estadísticas:** monitorización del número de paquetes y de bytes de cada flujo, permite implementar operaciones de QoS (Quality of Service)

Es necesario añadir una entrada a la tabla de flujo cuando el switch no encuentra ninguna coincidencia en la tabla para el paquete que ha recibido. El procedimiento habitual es que el switch envíe el paquete en cuestión al controlador. El controlador definirá entonces un nuevo flujo para el paquete y una o varias entradas para la tabla de flujo, que enviará al switch a través del protocolo OpenFlow. Por último, le devolverá el paquete al switch para que sea procesado con las nuevas entradas.

No obstante, que un paquete no coincida con las tablas de un switch no es la única razón para crear nuevas entradas. El controlador puede crear, modificar o eliminar las entradas de las tablas de flujo en cualquier momento, según la lógica programada en el mismo. Estas modificaciones pueden ser debidas a cambios en la topología de red, operaciones de QoS, amenazas de seguridad detectadas en el sistema, etc.

2.1.3 Sistemas de Detección de Intrusión (IDS)

Durante las últimas décadas, los Sistemas de Detección de Intrusión (IDS) se han utilizado junto con otros sistemas de seguridad como firewalls en diferentes ámbitos de la ciberseguridad. Un IDS es un software, hardware o combinación de ambos que monitoriza las actividades de una red o un sistema para detectar actividades maliciosas y notificar los eventos de intrusión al administrador de la red [4]. Como indica su propio nombre y, a diferencia de los cortafuegos, un IDS se encarga de detectar las amenazas en red, pero la tarea de responder a dicha amenaza podría recaer sobre el administrador de red. También existen sistemas de prevención de intrusión (IPS), que van un paso más allá que los IDS y tienen la capacidad de cerrar conexiones ante detecciones de situaciones que pueden constituir un problema, anticipándose de esta forma a las intrusiones que están por llegar. Por tanto, un IPS se podría considerar como un firewall a nivel de aplicación.

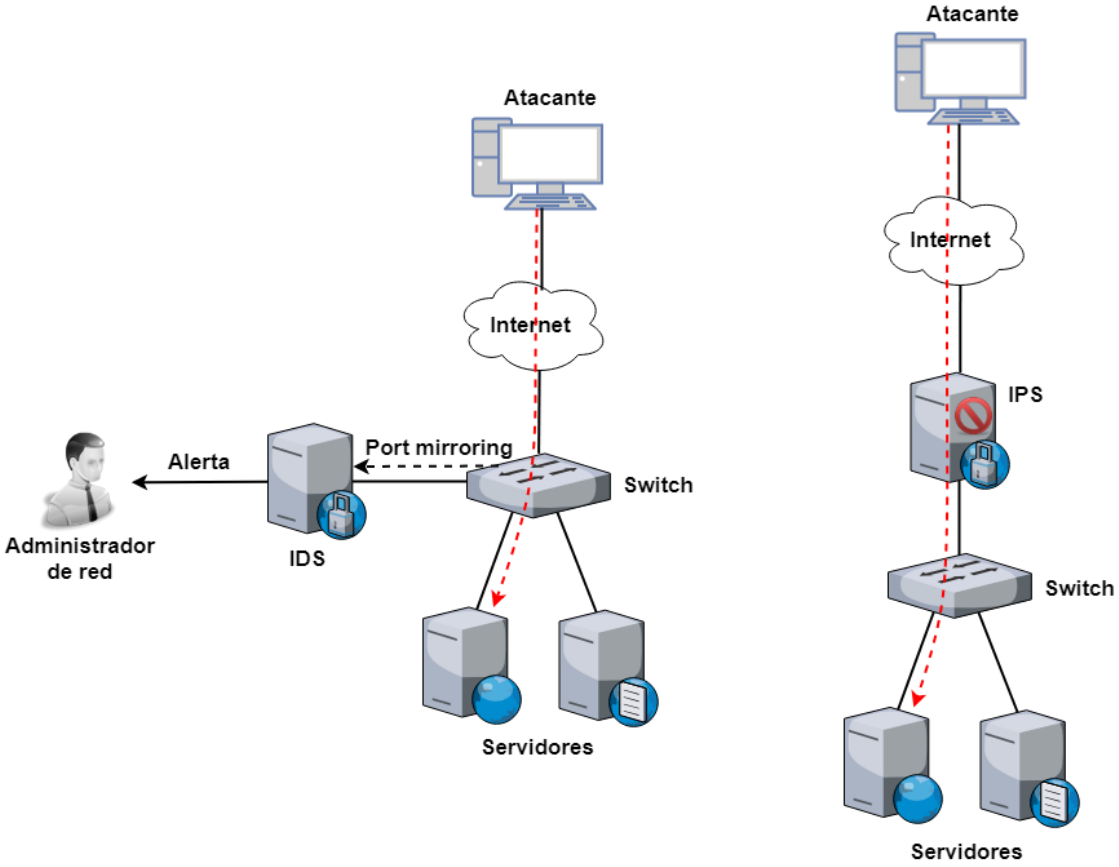

Ilustración 2.- Diferencias entre IDS e IPS

Uno de los beneficios de usar un IDS para identificar tráfico malicioso es que puede detectar ataques conocidos, pero también puede detectar comportamientos inusuales para intentar descubrir nuevos tipos de ataques. En esta línea, la detección basada en firmas se basa en el reconocimiento de patrones, y sirve para localizar ataques ya conocidos por el sistema. En cambio, la detección basada en anomalías es la que intenta encontrar nuevos ataques buscando actividades sospechosas en la red. En la práctica, se utiliza una combinación de ambas técnicas para mejorar la eficacia del IDS. Además, mientras los cortafuegos solo previenen ataques provenientes del exterior, un IDS también es capaz de detectar amenazas originadas dentro del sistema.

2.2 Escenario operacional

La propuesta de integrar un IDS en una red SDN resulta interesante por los beneficios que aporta la combinación de ambas tecnologías [5]. La arquitectura centralizada de las redes SDN otorga un mayor control sobre todos los elementos de la red y el IDS puede aprovechar dicha circunstancia para aumentar su efectividad. Sin embargo, esta misma característica de las redes SDN puede resultar contraproducente en la seguridad del sistema.

Por una parte, la integración de un IDS en redes SDN proporciona las siguientes ventajas:

- Como el controlador tiene acceso directo o indirecto a todo el tráfico de la red, la monitorización de tráfico del IDS es más efectiva.
- Es posible programar los controladores SDN para hacer frente a las amenazas en cuanto éstas son detectadas. Esto permite ir un paso más allá de las funciones tradicionales de un IDS ya que, en lugar de enviar las alertas al administrador de red, éstas llegan al controlador SDN, donde se podría resolver el problema siguiendo los pasos programados de antemano.

Por otra parte, el uso de un IDS en una red SDN puede generar algunos inconvenientes:

- El controlador es un punto único de fallo: si un atacante consigue acceso al controlador, toda la red se ve comprometida.
- La utilización de un interfaz de programación abierto en redes SDN hace que éstas sean más vulnerables frente a amenazas de seguridad.
- La arquitectura por capas de las redes SDN crea vulnerabilidades en más puntos de la red, ya que cada interfaz entre planos es un posible punto de ataque a la red.

A continuación, en la Tabla 1 se muestra un resumen de las ventajas y los inconvenientes que pueden resultar de la implementación de un IDS en redes SDN:

Ventajas	Inconvenientes
Monitorización del tráfico más eficiente	El controlador es punto único de fallo
Capacidad de programar el controlador SDN para hacer frente a amenazas	Más vulnerable a amenazas por utilizar un interfaz de programación abierto
	Cada interfaz de la arquitectura por capas es un posible punto de ataque

Tabla 1.- Ventajas e inconvenientes de la integración de un IDS en una red SDN

3 Objetivos y alcance

El objetivo principal de este proyecto es diseñar e implementar una solución que permita la integración de un Sistema de Detección de Intrusión (IDS) en Redes Definidas por Software (SDN). Para alcanzar este propósito, se han definido algunos objetivos parciales:

- En primer lugar, se ha diseñado un IDS adaptado a las redes SDN, capaz de cumplir con funciones análogas a las que realizaría en una red tradicional. Asimismo, se ha procurado aprovechar las características particulares de las SDNs para mejorar las funcionalidades que ofrecen los IDS convencionales. En esta línea, se busca programar un controlador SDN para responder a las amenazas detectadas por el IDS sin necesidad de intervención por parte de un administrador de red.
- Por otro lado, se ha diseñado y desplegado una maqueta que permita realizar diferentes pruebas a lo largo del proyecto. Esta maqueta ha permitido a su vez analizar la viabilidad del proyecto, en un escenario SDN lo más próximo posible a un entorno real.
- Por último, se ha desarrollado el IDS diseñado y se ha desplegado en la maqueta SDN. Se ha pretendido que el IDS desarrollado tenga las siguientes características:
    - **Modularidad**: el diseño del IDS debe tener una estructura modular para poder realizar ampliaciones futuras de una manera más sencilla. Esta flexibilidad permitirá incorporar nuevas funciones a la aplicación sin necesidad de rediseñar el IDS.
    - **Escalabilidad**: el IDS debe estar diseñado para hacer frente a diferentes escenarios de red SDN. Independientemente de la topología en la que se despliegue el sistema de seguridad, el IDS ha de funcionar de la forma más eficiente posible.
    - **Rendimiento**: el uso del IDS debe afectar lo menos posible al rendimiento general de la red. Se busca que el IDS genere la menor cantidad de tráfico posible para no perjudicar el funcionamiento del sistema.

4 Beneficios

En este apartado se definen los beneficios que aporta este Trabajo Fin de Grado tanto en aspectos técnicos como económicos y sociales.

4.1 Beneficios técnicos

El objetivo de este trabajo es conseguir integrar un IDS en una red SDN. Alcanzar este propósito tendría una serie de beneficios en el ámbito técnico y de investigación:

- En primer lugar, la solución diseñada en este TFG contribuye a incrementar la seguridad en las redes SDN. Las SDNs son una tecnología relativamente reciente, por lo que cualquier investigación al respecto supone una aportación de valor para la comunidad de desarrolladores.
- Por otra parte, en este proyecto se pretende aprovechar las especificidades de las redes SDN para extender las funcionalidades de un IDS tradicional. De esta manera, se plantea un nuevo enfoque en el uso de Sistemas de Detección de Intrusión en redes SDN, abriendo nuevas líneas de investigación en la materia y aportando posibles mejoras en las implementaciones actuales.
- Por último, aunque para la realización del proyecto se utilicen tecnologías concretas, el diseño de la solución se podrá abstraer sin demasiada complejidad a implementaciones con otros controladores SDN u otros IDS.

4.2 Beneficios económicos

El objetivo principal de la mayoría de las empresas es generar beneficios. Para ello, la seguridad de su infraestructura es un pilar fundamental, ya que una brecha de seguridad puede generar grandes pérdidas para la entidad. Además, las redes SDN son un concepto relativamente novedoso, por lo que la seguridad en estas redes supone un reto aún mayor si cabe para las compañías que han optado por hacer uso de ellas.

La solución adoptada en este proyecto puede aportar en este aspecto, estableciendo un modelo a seguir por parte de las corporaciones y proporcionando mayores funcionalidades a los sistemas de seguridad en las redes SDN.

Por otra parte, la solución se ha diseñado y desplegado utilizando herramientas, programas y librerías que son de código abierto. Esto, junto con la separación entre el plano de control y de datos que ofrecen las SDNs, proporciona una independencia total respecto a los fabricantes. De esta manera, la oferta disponible para adquirir equipos de red aumenta, reduciendo los costes de esta inversión.

Asimismo, la transparencia de las soluciones de código abierto permite adaptar las aplicaciones a las necesidades particulares de cada escenario, a diferencia de las soluciones comerciales que suelen ofrecer menos flexibilidad.

4.3 Beneficios sociales

La seguridad de los datos de una entidad es de gran relevancia para la propia compañía, pero también para los clientes de la misma. La seguridad de las corporaciones afecta directamente sobre la confianza de los usuarios, tal y como se ha visto en acontecimientos recientes del panorama internacional. Por consiguiente, la aportación de este proyecto en temas de seguridad puede beneficiar también a la experiencia de los particulares.

Por otro lado, ya se ha mencionado que el software utilizado en la solución es de código abierto. Esto significa que las modificaciones o mejoras realizadas durante el proyecto se convierten en una contribución para la comunidad, de manera que cualquier persona o entidad se pueda beneficiar de ellas.

5 Análisis de alternativas y selección de la solución

A lo largo del desarrollo de cualquier proyecto es necesario tomar decisiones que afectan al desenlace del mismo. En este apartado se detallan las alternativas que se han barajado en diferentes etapas del Trabajo Fin de Grado, los criterios que se han establecido para su evaluación y la alternativa seleccionada en cada caso.

5.1 Escenario de red SDN

Dado que se va a trabajar con una red SDN, es imprescindible desplegar una red de estas características. Para elegir el escenario sobre el que se va a trabajar, se han establecido ciertos criterios:

Criterio	Ponderación
Coste	3
Escalabilidad	2
Rendimiento	1

Tabla 2.- Criterios de selección para "Escenario de red SDN"

Para desplegar una maqueta de red SDN existen dos posibilidades principales: desplegar una maqueta de red con equipos reales u optar por un escenario virtualizado.

5.1.1 Equipamiento real

Utilizar equipamiento de red real para desplegar la maqueta de red SDN permitiría trabajar en un escenario lo más parecido posible a uno real. Esto proporciona ciertas ventajas, sobre todo en el rendimiento de la maqueta: emplear dispositivos de red reales asegura que el funcionamiento de los equipos será el mismo que en un entorno de producción, por lo que los resultados obtenidos en el proyecto serán válidos para cualquier red SDN desplegada.

No obstante, hacer uso de equipamiento real también conlleva algunos inconvenientes. En primer lugar, el coste de utilizar equipos de red SDN reales es bastante elevado actualmente. Este mismo factor condiciona la escalabilidad de la maqueta de red, pues añadir nuevos equipos a la red supone un gasto adicional importante. Por otro lado, al emplear equipos reales no es posible cambiar las características hardware de los equipos una vez se han adquirido.

5.1.2 Entorno virtual

Utilizar un entorno de simulación aporta ciertos beneficios al proyecto. Por un lado, el coste del equipamiento se reduce de forma drástica, pudiendo llegar a no suponer coste alguno si se hace uso de soluciones de simulación gratuitas. Al mismo tiempo, esta reducción del coste de adquisición hace que la escalabilidad de la maqueta desplegada sea mucho mayor que si se usan equipos reales. Por otro lado, un escenario con equipos virtualizados proporciona mayor flexibilidad y facilidad para probar diferentes escenarios según las necesidades en cada etapa del proyecto.

Sin embargo, el comportamiento de la red en un entorno virtual puede ser diferente al de un entorno real, por lo que pueden ser necesarios ciertos cambios para adaptar la solución a un escenario real. Aun así, un escenario virtualizado puede ser el paso previo ideal a pruebas con equipos reales.

5.1.3 Selección de alternativas

Considerando los criterios ponderados y las alternativas de las que se dispone, se ha realizado una valoración para elegir la opción más acorde a las características del proyecto:

Criterio y ponderación	Equipamiento real	Entorno virtual
Coste (3)	3	10
Escalabilidad (2)	5	10
Rendimiento (1)	10	6
TOTAL (sobre 60 puntos)	29	56

Tabla 3.- Selección de alternativas para "Escenario de red SDN"

Teniendo en cuenta los resultados, se ha optado por desplegar la maqueta de red SDN en un entorno virtual o de simulación.

5.2 Entorno de simulación SDN

En el apartado anterior se ha decidido desplegar la maqueta de red en un entorno virtual. Sin embargo, la variedad de entornos de simulación en el mercado es bastante amplia en la actualidad.

Para poder valorar las alternativas disponibles y elegir la más adecuada para el proyecto, se han establecido una serie de criterios y ponderaciones:

Criterio	Ponderación
Coste	2
Compatibilidad con elementos SDN reales	3
Escalabilidad	1
Rendimiento	1

Tabla 4.- Criterios de selección para "Entorno de simulación SDN"

Entre los diferentes entornos de simulación SDN disponibles, se han valorado las siguientes alternativas:

- Mininet
- GNS3
- EstiNet

La elección de estas opciones se debe a que estas herramientas son las más reconocidas y utilizadas a nivel mundial, además de ser referentes en los artículos publicados en el área de las redes SDN.

5.2.1 Mininet

Mininet es un simulador de red de código abierto pensado para apoyar actividades de investigación y educación en el ámbito de las Redes Definidas por Software. Está pensado para crear redes SDN de forma sencilla, utilizando un controlador OpenFlow, varios switches

OpenFlow conectados en una red Ethernet y múltiples hosts conectados a éstos. También tiene funciones para soportar diferentes tipos de switches y controladores, y una API en Python para crear escenarios más complejos.

Por otra parte, las redes de Mininet ejecutan código real de Linux. Gracias a esta característica, el código desarrollado en Mininet (para un controlador, un switch o un host) es fácilmente reproducible en un entorno real efectuando cambios mínimos. Por tanto, un diseño que funcione correctamente en Mininet se puede emplear prácticamente de forma directa para desplegar, probar y evaluar el rendimiento de una red SDN real.

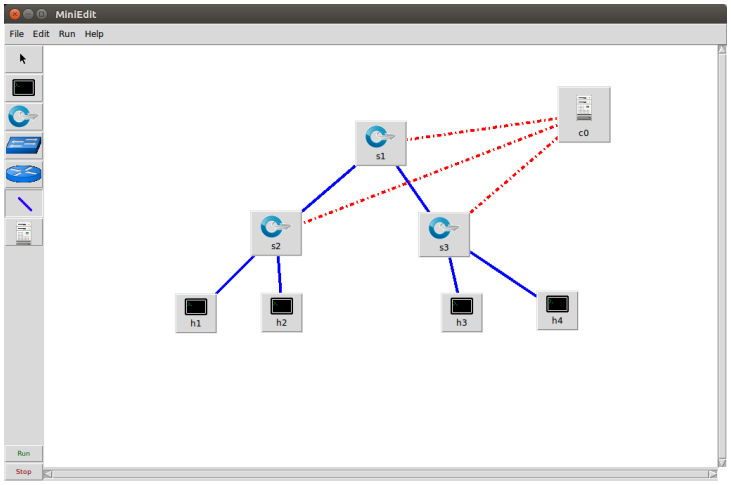

Ilustración 3.- Interfaz gráfica de Mininet - MiniEdit

5.2.2 GNS3

GNS3 (Graphical Network Simulator), tal y como su nombre indica, es un simulador gráfico de red. Este simulador permite ejecutar emuladores de elementos como routers (se soportan varios fabricantes como Cisco, Juniper…) o máquinas virtuales (ya sean de Linux o de Windows), entre otros.

Sin embargo, dado que no tiene soporte nativo de elementos de redes SDN por el momento, es necesario agregar a la maqueta una máquina virtual de Mininet VM para utilizar funcionalidades SDN. Además, el hecho de ejecutar imágenes completas de los equipos hace que a partir de cierto número de elementos de red su rendimiento se ve afectado.

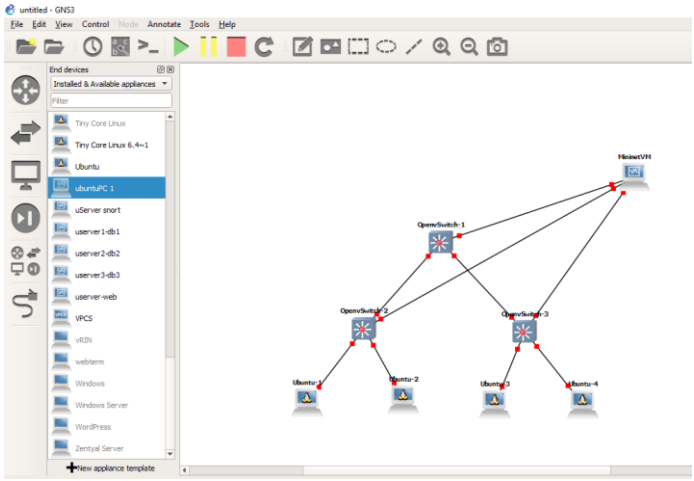

Ilustración 4.- Interfaz gráfica de GNS3

5.2.3 EstiNet

EstiNet es un simulador y emulador de red que implementa todas las capas OSI de los equipos de red. Aunque en sus inicios fue utilizado para investigación, en 2011 pasó a ser un software comercial. La interfaz gráfica de EstiNet hace que sea muy sencillo crear una red simulada y visualizar los resultados de la simulación. Al igual que Mininet, integra el kernel de Linux para proporcionar los servicios de red.

Por otro lado, además de la simulación también incluye funciones de emulación, lo que lo convierte en una herramienta adecuada para diseñar redes que posteriormente van a ser desplegadas en un sistema real.

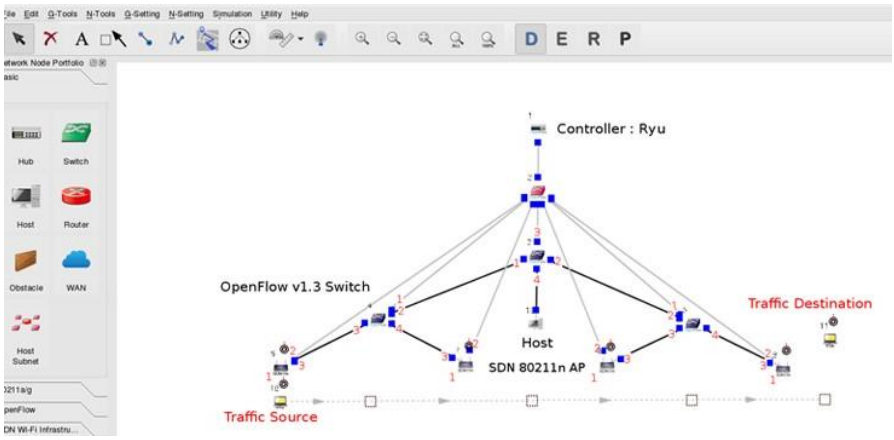

Ilustración 5.- Interfaz gráfica de EstiNet

5.2.4 Selección de alternativas

Teniendo en cuenta los criterios establecidos y las características de cada alternativa, se han evaluado las opciones contempladas para seleccionar la solución final:

Criterio y ponderación	Mininet	GNS3	EstiNet
Coste (2)	10	10	5
Compatibilidad con elementos SDN reales (3)	10	0	10
Escalabilidad (1)	7	4	8
Rendimiento (1)	7	6	8
TOTAL (sobre 70 puntos)	64	30	56

Tabla 5.- Selección de alternativas para "Entorno de simulación SDN"

Finalmente, se ha decidido hacer uso de Mininet para desplegar la maqueta de red SDN que se va a utilizar durante el proyecto.

5.3 Controlador SDN

Teniendo en cuenta los elementos de una red SDN, es necesario contar con uno o varios controladores SDN en la red. Para optar por uno de ellos, se han establecido ciertos criterios:

Criterio	Ponderación
Versiones de OpenFlow soportadas	2
Soporte y documentación	1
Rendimiento	2

Tabla 6.- Criterios de selección para "Controlador SDN"

Durante los últimos años, se han diseñado nuevos controladores SDN y también variaciones de otros ya existentes, por lo que son muchas las opciones entre las que elegir. De todos los controladores disponibles en el mercado, se han elegido algunas por su buena acogida entre los usuarios y las características y el rendimiento que ofrecen. Entre los elegidos se encuentran el controlador ONOS, OpenDaylight, NOX, POX, Ryu y Floodlight.

Para poder valorar todos los controladores contemplados, se han representado mediante una tabla comparativa las características de las que disponen las diferentes alternativas:

Características	ONOS	OpenDaylight	NOX	POX	Ryu	Floodlight
Lenguaje de programación	Java	Java	C++	Python	Python	Java
Versiones OpenFlow soportadas	OF 1.0, 1.3	OF 1.0, 1.3, 1.4	OF 1.0	OF 1.0	OF 1.0, 1.2, 1.3, 1.4, 1.5	OF 1.0, 1.3, 1.4
Documentación	Buena	Muy buena	Pobre	Pobre	Buena	Buena
Rendimiento	Medio	Medio	Medio	Alto	Alto	Medio

Tabla 7.- Comparativa de controladores SDN

En base a estas características y los criterios establecidos, se han valorado las diferentes opciones:

Criterio y ponderación	ONOS	OpenDaylight	NOX	POX	Ryu	Floodlight
Versión de OpenFlow soportada (2)	6	8	4	4	10	8
Soporte y documentación (1)	7	9	5	5	7	7
Rendimiento (2)	5	5	5	8	8	5
TOTAL (sobre 50 puntos)	29	35	23	29	43	33

Tabla 8.- Selección de alternativas para "Controlador SDN"

Por tanto, se ha elegido el controlador Ryu para el proyecto porque se cree que es el más completo de todos: soporta el mayor número de versiones OpenFlow, dispone de buena documentación y su rendimiento es alto.

5.4 IDS

Si el objetivo del trabajo es la integración de un IDS en una red SDN, es preciso elegir un Sistema de Detección de Intrusión acorde al alcance del proyecto. Para ello, se han definido los siguientes criterios:

Criterio	Ponderación
Uso de recursos	2
Soporte y documentación	1
Rendimiento	2
Complejidad	1

Tabla 9.- Criterios de selección para "IDS"

Considerando la necesidad de adaptar el IDS a las particularidades de las redes SDN, se cree más conveniente hacer uso de un IDS de software libre. Entre las opciones disponibles en el mercado, se han elegido Snort, Suricata y Bro, pues éstos son los IDS de software libre más reconocidos por la comunidad por las características que ofrecen.

5.4.1 Snort

Snort es el IDS líder en la comunidad *opensource*. Aunque no disponga de ninguna interfaz de gestión sencilla, su efectividad como NIDS en un gran abanico de escenarios ha hecho que consiga una gran aceptación por parte de la comunidad. Snort utiliza tanto métodos de detección de firmas como de detección de anomalías para la detección de amenazas. El formato claro y sencillo de las denominadas *Snort Rules* y la amplia comunidad de usuarios permiten crear reglas para las amenazas emergentes en tiempo récord. No obstante, cuando Snort fue creado hace 20 años fue diseñado para la infraestructura de esa época, por lo que adaptarse a los nuevos requerimientos de las redes (amenazas más complejas, enlaces de alta capacidad, mayor velocidad de procesamiento…) está resultando en todo un reto.

5.4.2 Suricata

Suricata fue creado hace casi una década con el propósito de cumplir las demandas de las infraestructuras modernas. Al igual que Snort, Suricata utiliza un formato basado en reglas; de hecho, ofrece compatibilidad con las *Snort Rules*. Sin embargo, Suricata incorpora el lenguaje de scripting Lua para dotar de más flexibilidad a las reglas de Snort. Asimismo, Suricata introdujo la opción de emplear técnicas de multihilo, proporcionando la capacidad (teórica) de procesar más reglas en redes de más capacidad y tráfico utilizando el mismo hardware. Por tanto, es habitual encontrar comparaciones que definen Suricata como una extensión de Snort para redes a mayor escala.

5.4.3 Bro

Bro es un IDS con un enfoque completamente distinto a Snort y Suricata. En lugar de emplear una estructura basada en reglas, Bro utiliza un lenguaje de scripting propio denominado Bro Script (con cierto parecido a C++) para definir las condiciones para los incidentes de red. Aunque puede ser utilizado como IDS, Bro suele utilizarse más a menudo para grabar el comportamiento de la red de manera detallada por su inspección minuciosa del tráfico. El inconveniente de esta flexibilidad es que Bro consume recursos de forma intensiva.

5.4.4 Selección de alternativas

Considerando las alternativas propuestas y los criterios definidos se ha realizado la siguiente valoración:

Criterio y ponderación	Snort	Suricata	Bro
Uso de recursos (2)	10	7	5
Soporte y documentación (1)	10	7	6
Rendimiento (2)	6	8	10
Complejidad (1)	9	8	6
TOTAL (sobre 60 puntos)	51	45	42

Tabla 10.- Selección de alternativas para "IDS"

Después de realizar la valoración, se considera que el IDS más apropiado para el proyecto es Snort.

6 Diseño de la solución

Durante la fase de diseño, se han definido dos escenarios de desarrollo para llevar a cabo el proyecto: un primer escenario para la realización de pruebas de funcionalidades individuales y un segundo escenario para la integración.

6.1 Escenario 1

En primer lugar, se ha diseñado un escenario que permita desarrollar pruebas de forma sencilla y ágil. Con ese propósito, se ha definido una maqueta de red suficientemente sencilla como para que se puedan analizar y estudiar las funcionalidades deseadas, sin que la interacción entre componentes o la complejidad del escenario influyan en los resultados correspondientes.

En concreto, las funcionalidades que se quieren probar son las siguientes:

- **Switch básico de nivel 2:** se pretende configurar los switches de la red de manera que reenvíen los paquetes por el puerto correspondiente al destino.
- **Reenvío del tráfico al IDS:** es necesario que los switches de la red copien todos los paquetes recibidos y los envíen al IDS para que éste tenga acceso a todo el tráfico de la red y pueda analizarlo.
- **Recepción de alarmas del IDS:** para que el controlador pueda responder a las amenazas detectadas en la red, es preciso que reciba las alarmas del IDS.
- **Respuesta a alarmas:** una vez que el controlador recibe la alarma del IDS, debe tener una respuesta programada para responder a la amenaza detectada.

Para ello, el escenario incluye los siguientes elementos:

- 3 equipos finales
- 1 switch
- 1 IDS
- 1 controlador SDN

A continuación, se muestra un esquema del escenario definido:

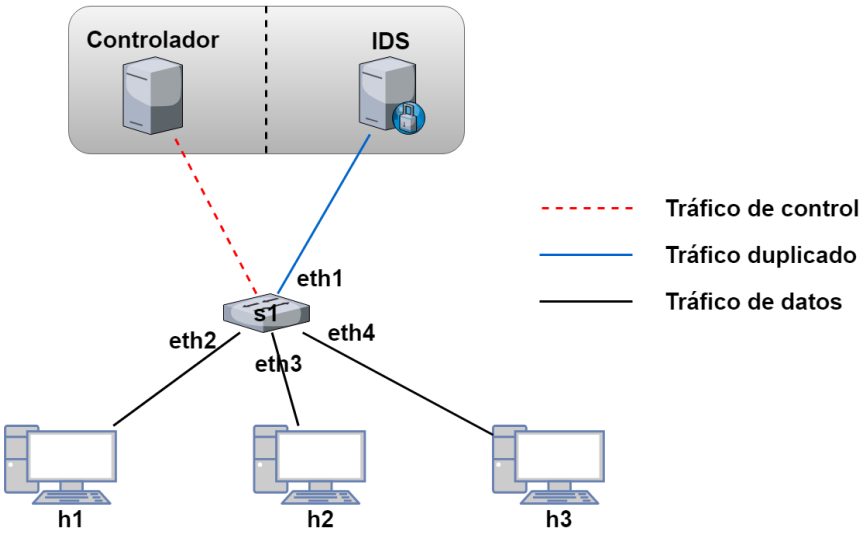

Ilustración 6.- Diseño escenario 1

6.1.1 Configuración del escenario

En este apartado se define la configuración que necesitará cada elemento del escenario de desarrollo, teniendo en cuenta las funciones que debe cumplir cada uno de ellos. La forma de llevar a cabo esta configuración (junto con la del escenario 2) se detallará más tarde en el apartado Descripción de la solución.

Equipos finales

El propósito de los equipos finales es simular a los usuarios finales de la red. Es decir, su función será generar tráfico de usuario para poder probar así las funcionalidades del resto de elementos del sistema.

Respecto a la configuración de los equipos, lo único que hace falta para que sean capaces de comunicarse a través de la red es asignar una dirección IP.

Switch

El switch tiene dos cometidos principales que debe cumplir: por un lado, reenviar los paquetes recibidos hacia su destino y, por otro lado, enviar copias de dichos paquetes al IDS.

Por tanto, la configuración que tendrá el switch también constará de dos partes claramente diferenciadas:

- Por una parte, para reenviar los paquetes al destino será necesario llevar un registro de las direcciones MAC de los equipos y los puertos del switch correspondientes por los que hay que enviar los paquetes. Para ello, el funcionamiento será el mismo que en los switches tradicionales: cuando se reciba un paquete con una dirección MAC destino sin registrar, el paquete se reenviará mediante inundación por todos los puertos del switch. Al mismo tiempo, se registrará la dirección MAC origen junto con el puerto de entrada para ir registrando las MAC conocidas y evitar en la medida de lo posible la inundación. Cuando el paquete recibido tenga una MAC destino conocida, simplemente se reenviará por el puerto correspondiente. Sin embargo, hay un detalle importante que difiere respecto a las redes convencionales: el comportamiento explicado arriba no está programado en el switch, sino en el controlador. Cuando el switch recibe un paquete y no tiene una entrada configurada que coincida, envía el paquete al controlador para que éste le indique las acciones que debe realizar. Esta manera de proceder se profundizará en el apartado correspondiente a la configuración del controlador.
- Por otra parte, para enviar copias de los paquetes recibidos al IDS se utilizará una técnica conocida como *port mirroring*. Este método es utilizado para que un switch envíe copias de los paquetes recibidos por un puerto hacia un enlace habitualmente conectado a un equipo de monitorización. En este escenario, el switch enviará al IDS copias de todo el tráfico, por lo que tendrá que emplear esta técnica en todos los puertos excepto el que está conectado al IDS.

No obstante, al tratarse de una red SDN donde el plano de control depende exclusivamente del controlador, la configuración mencionada arriba se realizará a través del mismo en lugar de hacerlo directamente sobre el switch. Para ello, se hará uso del protocolo OpenFlow, cuyas características se han descrito en el apartado OpenFlow. Esto significa que en vez de añadir entradas manualmente a las tablas de flujo del switch para las dos funciones que deben realizar, se programará el controlador para que añada las entradas de manera automática cada vez que

un switch se incorpora a la red. De esta manera, la configuración es también válida si se añaden más switches a la red y permite, además, cambiar la configuración de los switches de forma más dinámica y rápida, pues es posible hacer cambios de manera global en lugar de hacerlo individualmente para cada switch.

IDS

El objetivo del IDS es detectar las amenazas de la red y avisar al controlador de la situación. Para ello, debe recibir todo el tráfico de la red y analizarlo, detectar las amenazas, generar las alarmas correspondientes y enviarlas al controlador.

La configuración necesaria para conseguir implementar estas funcionalidades se realizará en tres pasos:

- En primer lugar, habrá que configurar el IDS para que reciba todo el tráfico de la red. Como ya se ha configurado el switch para enviar copias de los paquetes recibidos al IDS, éste solo tiene que escuchar en el puerto conectado al switch. Por tanto, se indicará al IDS que tiene que capturar tráfico en la interfaz conectada al switch.
- En segundo lugar, es necesario configurar las condiciones según las cuales el IDS detecta las amenazas y genera alarmas. Es decir, el IDS contrasta el tráfico recibido con una serie de reglas, y si encuentra coincidencias es cuando genera las alarmas correspondientes. Para este escenario, se definirá una regla sencilla pero efectiva a la vez: si se detecta un paquete ICMP en la red, el IDS generará una alarma. Aunque esta regla puede parecer trivial, existen muchas técnicas de ciberataques basados en paquetes ICMP: barridos por ICMP, inundaciones ICMP (denegación de servicio), *fingerprinting* del sistema operativo, etc.
- Por último, el IDS debe ser capaz de enviar las alarmas al controlador. Para ello, existen dos posibilidades: si el IDS y el controlador se encuentran en equipos diferentes, es necesario enviar las alarmas a través de la red; en cambio, si el IDS y el controlador se ejecutan en el mismo equipo, es posible enviar las alarmas a través de los sockets de dominio Unix. La primera opción permite que el IDS tenga todos los recursos de un equipo dedicados para el análisis de tráfico, lo que aumenta su rendimiento. Sin embargo, también genera mucho tráfico adicional al tener que enviar las alarmas a través de la red. Al contrario, la segunda alternativa tiene características opuestas a la primera. Considerando el alcance de las pruebas que se quieren realizar, se ha decidido que es más conveniente que tanto el IDS como el controlador estén en el mismo equipo. Con la regla de detección configurada, los recursos que consumirá el IDS no serán excesivos, por lo que es preferible que no se agregue más tráfico y que la red funcione de la manera más eficiente posible.

Controlador

El controlador es el elemento más relevante en una red SDN, pues es el encargado de realizar todas las funciones de control. Estas funcionalidades incluyen la gestión de paquetes entrantes (cuando el switch no sabe qué hacer con un paquete y se lo envía al controlador) y la adición de entradas en las tablas de flujo del switch, entre otros. Asimismo, en este escenario también debe responder a las alarmas recibidas del IDS.

A continuación, se explicará cómo se ha diseñado el controlador para configurar cada una de las funciones que debe realizar:

- Respecto a la gestión de paquetes entrantes, el controlador tiene que añadir entradas a las tablas de flujo del switch para que éste realice el reenvío de paquetes tal y como se ha indicado en el apartado de configuración del switch, es decir, enviando los paquetes por inundación cuando la MAC destino no está registrada y por el puerto correspondiente en el caso de que esté guardada. Para ello, el controlador debe llevar un registro de las direcciones MAC y los puertos de salida correspondientes. Para cada paquete recibido, comprobará si la MAC está registrada o no y actuará en consecuencia:
    - En caso de que la MAC esté registrada, añadirá al switch una entrada para indicar el puerto de salida y evitar así el paquete entrante para paquetes posteriores. Al mismo tiempo, indicará al switch por dónde tiene que enviar el paquete actual para que éste no es pierda, ya que la entrada se ha añadido después de que el switch contraste sus entradas con el paquete.
    - En caso de que la MAC no esté registrada, el controlador indicará al switch que debe enviar el paquete mediante inundación por todos sus puertos.
    - En ambos casos, el controlador registrará la dirección MAC origen y el puerto de llegada para ir aprendiendo las direcciones MAC de los equipos de la red.
- En el punto anterior no se ha tenido en cuenta que todos los paquetes deben llegar también al IDS. Por ello, tanto en las entradas que se añadan al switch como en las instrucciones que dé el controlador, se indicará que el paquete también debe enviarse por el puerto conectado al IDS.
- En lo que se refiere a la recepción de alarmas, ya se ha mencionado que la comunicación entre el controlador y el IDS (ambos en el mismo equipo) se realizará mediante sockets de dominio Unix. Para comprobar que las alarmas se reciben correctamente, se configurará el controlador de manera que muestre en consola la información correspondiente a las alarmas recibidas.
- Por último, hay que configurar el controlador para que responda frente a las amenazas detectadas por el IDS. Para ello, se ha definido que la acción que debe realizar cuando se detecte un paquete malicioso es bloquear la dirección IP origen del paquete. Para hacer efectiva esta medida en la red, se realizarán los siguientes pasos:
    - Cuando se recibe una alarma, además de informar en consola sobre ella, se extraen los datos correspondientes a la IP origen del paquete que ha originado la alarma.
    - Con la IP extraída, se genera una entrada para el switch indicando que un paquete con esa IP origen debe ser descartado.
    - Se añade la entrada al switch, de manera que a partir de ese momento el equipo con esa dirección IP queda bloqueado en la red.

6.1.2 Plan de pruebas

En este apartado se definirán las pruebas diseñadas para comprobar que las funciones deseadas en cada elemento de la red se han configurado de manera adecuada. Después, en el apartado de Análisis de resultados de las pruebas se describirán los resultados de las mismas.

Switch básico de nivel 2

El propósito de esta prueba es comprobar que el switch realiza correctamente las funciones de reenvío de paquetes y aprendizaje de direcciones MAC.

Para llevar a cabo la prueba, se realizarán 3 pings consecutivos entre los equipos h1 y h2, capturando mediante Wireshark las interfaces del switch conectadas a esos equipos (s1-eth2 y s1-eth3 respectivamente). Si los pings se muestran en ambas interfaces, se podrá afirmar que el reenvío de paquetes ha sido satisfactorio. Al mismo tiempo, se capturará en la interfaz s1-eth4, conectada a h3, para probar el aprendizaje MAC. Si en dicha captura no aparecen los pings en cuestión, significará que se ha registrado la MAC correctamente y se ha evitado reenviar los paquetes por inundación.

Reenvío del tráfico al IDS

El objetivo de la prueba es confirmar que el IDS recibe el tráfico de la red del switch y puede generar alarmas con los paquetes recibidos.

Para ello, se generará el mismo tráfico que en la prueba anterior: 3 pings consecutivos entre h1 y h2. En este caso, la captura de Wireshark se realizará en la interfaz del switch conectada al IDS (s1-eth1). Si se captura el mismo tráfico observado en s1-eth2 en la prueba previa a ésta, significará que la copia de los paquetes se ha realizado correctamente. Por otro lado, se pondrá en marcha el IDS con la regla de detección de paquetes ICMP mencionada en la configuración del IDS, indicando que muestre las alarmas por consola. De esta manera, se podrán observar las alarmas generadas por el IDS correspondientes a los paquetes ICMP enviados en la red.

Recepción de alarmas del IDS

La finalidad de esta prueba es verificar que el controlador SDN recibe las alarmas generadas por el IDS.

Con ese fin, se llevarán a cabo los mismos pasos que para la prueba anterior, con la excepción de que, en lugar de indicar al IDS que muestre las alarmas por consola, se indicará al mismo que envíe las alarmas al controlador mediante los sockets de dominio Unix. Como el controlador se ha configurado para mostrar en consola las alarmas recibidas del IDS, se podrá comprobar si la recepción de alarmas funciona adecuadamente.

Respuesta a alarmas

Esta última prueba pretende demostrar que el controlador es capaz de responder a las alarmas generadas por el IDS.

Para asegurarse de ello, se seguirá la estructura de las pruebas anteriores y se realizarán 3 pings consecutivos entre los equipos h1 y h2. En este caso, será interesante capturar en las interfaces conectadas a h1 y h2 (s1-eth2 y s1-eth3), ya que la respuesta del controlador que se ha configurado es bloquear la IP origen del paquete malicioso. Por tanto, una vez se genere la respuesta del controlador, el switch debería descartar los paquetes provenientes de h1, en lugar de reenviarlos hacia h2, y esto se plasmaría en dichas capturas. Asimismo, el controlador informará en consola de las alarmas recibidas y las acciones llevadas a cabo para solucionar la situación.

6.2 Escenario 2

Después de diseñar el primer escenario, destinado a pruebas sencillas, se ha diseñado un segundo escenario más complejo para realizar pruebas de funcionalidades más avanzadas, así como las pruebas de integración. Con esa finalidad, se ha definido una maqueta de red más sofisticada que la primera, donde existe interacción entre los componentes, para estudiar el comportamiento de la red en situaciones más reales.

En concreto, las funcionalidades que se quieren probar son las siguientes:

- **Protocolo STP:** como este escenario es más complejo y dispone de enlaces redundantes entre los switches, es necesario un protocolo como STP (Spanning Tree Protocol) para gestionarlos.
- **Reenvío del tráfico al IDS:** este escenario consta de varios switches en lugar de uno solo, por lo que, a diferencia del primer escenario, habrá que configurar el IDS para capturar el tráfico proveniente de todos ellos.
- **Respuesta a alarmas:** igual que para el punto anterior, el hecho de tener varios switches afecta también a la respuesta que genera el controlador frente a las alarmas recibidas, pues debe solucionar el problema en toda la red y no solo en un punto del mismo.
- **Integración:** se realizará una comprobación final donde se implementarán todas las funcionalidades probadas hasta la fecha.

Para ello, el escenario incluye los siguientes elementos:

- 4 equipos finales
- 6 switches
- 1 IDS
- 1 controlador SDN

A continuación, se muestra un esquema del escenario definido:

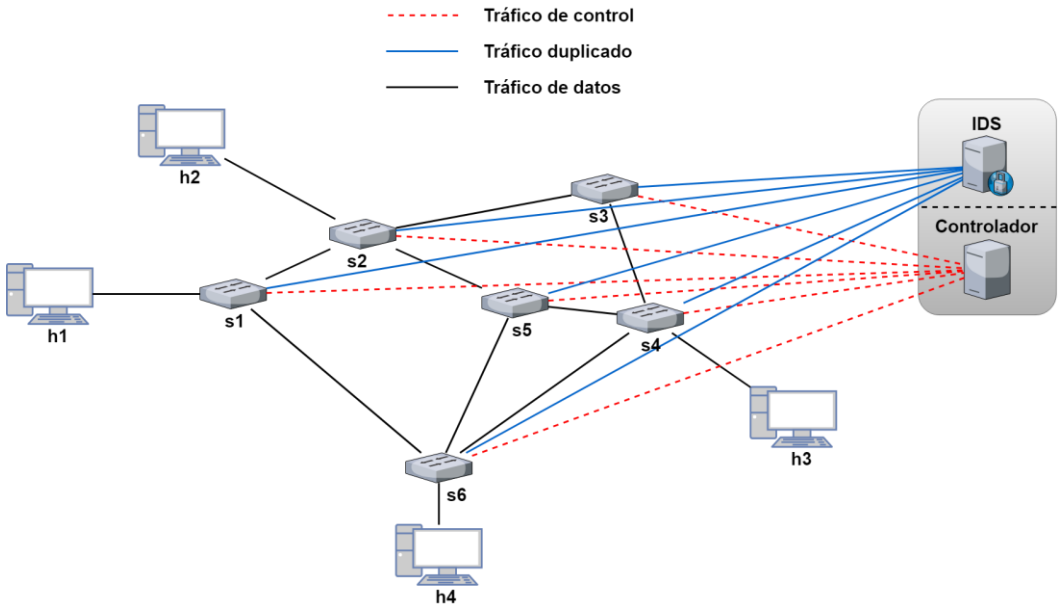

Ilustración 7.- Diseño escenario 2

6.2.1 Configuración del escenario

En este apartado se define la configuración a realizar en cada componente de la red para obtener las funciones deseadas. La forma de llevar a cabo esta configuración (junto con la del escenario 1) se detallará más tarde en el apartado Descripción de la solución.

Equipos finales

La función de los equipos finales es la misma que en el escenario 1, es decir, simular usuarios de la red generando tráfico de todo tipo para verificar así las funcionalidades del resto de elementos.

Igual que en el primer escenario, la única configuración necesaria en los equipos es la asignación de una dirección IP.

Switches

En el escenario 1 el switch tenía dos funciones: reenviar los paquetes y copiar el tráfico hacia el IDS. En el escenario actual, además de dichas funciones, los switches deben gestionar el protocolo STP para el manejo de enlaces redundantes.

Respecto a la configuración de los switches, la única diferencia respecto al primer escenario es también debida al protocolo STP. El protocolo STP funciona de la misma manera en redes SDN y en las redes tradicionales: elección de un puente raíz, determinación de los puentes designados y puertos raíz y designados, y bloqueo del resto de puertos. Para ello, los switches intercambian mensajes de configuración llamados BPDUs (Bridge Protocol Data Units). Por tanto, habrá que configurar los switches para que puedan enviar y recibir dichos mensajes. No obstante, al igual que en el escenario 1, se pretende que las tareas de control estén centralizadas, por lo que se intentará configurar el protocolo STP a través del controlador SDN.

IDS

La única diferencia en el IDS respecto al primer escenario es que debe recibir el tráfico proveniente de varios switches. El resto de funciones permanecen inalteradas: analizar el tráfico, detectar amenazas y enviar alarmas al controlador.

Para poder recibir los paquetes en varias interfaces de forma simultánea, habrá que configurar el IDS para capturar en todos y cada uno de los enlaces que se encuentran conectados a los switches. Para ello, se van a lanzar varias instancias del IDS, todas ellas con la misma configuración, pero cada una de ellas capturando en una interfaz distinta. De esta manera, es posible capturar tráfico desde múltiples interfaces. Asimismo, esta estrategia puede ser de utilidad en un futuro si se quiere dar prioridad a analizar el tráfico de algún enlace en concreto o si se quieren emplear diferentes opciones de configuración en cada interfaz.

Controlador SDN

El controlador es el centro de la red SDN, por lo que agrupa muchas de las funciones necesarias en la red. Además de todas las funcionalidades mencionadas para el escenario 1, en el escenario actual debe gestionar el protocolo STP. Por otra parte, al disponer ahora de varios switches en la red, el controlador debe responder a las alarmas protegiendo toda la red, no solo el punto donde se ha detectado la amenaza.

En cuanto a la configuración necesaria para llevar a cabo estas funciones, también se ha definido por partes:

- En primer lugar, toda la configuración expuesta para el primer escenario es imprescindible también para el segundo.
- Respecto al protocolo STP, el controlador debe gestionar los paquetes BPDU, llevar un registro de los estados de los puertos y añadir, modificar o eliminar las entradas correspondientes en las tablas de flujo de los switches. Teniendo en cuenta que existen ya implementaciones de esta funcionalidad y que se aleja en cierto modo del objetivo final del proyecto, se ha decidido hacer uso de una librería ya existente para gestionar el protocolo STP. Por tanto, será preciso importar dicha librería y adaptarla y configurarla a las necesidades del escenario de desarrollo. Se procurará que la librería empiece a funcionar nada más arrancar la maqueta y el controlador, y que proporcione información en consola sobre el estado de los puertos para poder comprobar las rutas activas y bloqueadas.
- En cuanto a los cambios necesarios en las respuestas a las alarmas, ya se ha mencionado que la finalidad es arreglar el problema en toda la red. Para ello, el cambio realizado respecto al primer escenario será el siguiente: una vez identificada la IP origen a bloquear y creado el flujo para la tabla de flujo de los switches, en lugar de añadir la entrada solo en el switch que ha recibido el paquete malicioso, se añadirá ese mismo flujo en todos los switches a cargo del controlador. De esta forma, la IP origen del paquete que ha provocado la alarma queda bloqueada al instante en toda la red.

6.2.2 Plan de pruebas

En este apartado se detallará el plan de pruebas diseñado para comprobar el correcto funcionamiento de la solución desarrollada. Después, en el apartado de Análisis de resultados de las pruebas se describirán los resultados de las mismas.

Protocolo STP

El propósito de esta prueba es comprobar el correcto funcionamiento de la librería importada para gestionar el protocolo STP.

Para llevar a cabo la prueba, se pondrá en marcha la maqueta de red y se arrancará el controlador. Si se ha configurado la librería según lo diseñado, el protocolo STP debería empezar a funcionar en ese momento. Para observar si realmente funciona, se tendrán en cuenta dos condiciones: por un lado, el estado de los puertos de los switches, mostrado en la consola del controlador SDN, y, por otro lado, las tablas de flujo, donde se configuran las entradas para indicar a los switches qué puertos deben estar activos y cuales deben estar bloqueados.

Reenvío del tráfico al IDS

El objetivo de esta prueba es verificar que el IDS recibe el tráfico de todos los switches de la red, y que genera las alarmas correspondientes.

Para ello, siguiendo la misma estrategia que con el escenario 1, se hará uso de paquetes ICMP. Se realizarán 3 pings consecutivos desde h1 hacia h4, por lo que los paquetes pasarán previsiblemente por los switches s1 y s6. Por tanto, se capturará mediante Wireshark el tráfico en las interfaces de esos switches conectados al IDS, para mostrar que efectivamente el *port mirroring* funciona adecuadamente. Por otra parte, se observarán las consolas de las dos instancias del IDS que escuchan en esos enlaces, mostrando las alarmas generadas en cada una de ellas.

Respuesta a alarmas

La finalidad de esta prueba es confirmar que la respuesta a las alarmas se ha adaptado de manera adecuada al escenario 2.

Con ese fin, se generará el mismo tráfico que para la prueba anterior. En esta prueba se observarán las tablas de flujo de los switches para demostrar que las entradas correspondientes a la respuesta del controlador se han añadido de forma satisfactoria. Asimismo, se tendrá en cuenta la salida en la consola del controlador, donde se indican las medidas tomadas para resolver la situación.

Integración

Esta última prueba tiene como objetivo comprobar que todas las funcionalidades pueden emplearse de manera simultánea, es decir, verificar que se ha integrado toda la solución de forma adecuada.

Para llevar a cabo esta prueba, se realizarán todas las comprobaciones de las pruebas anteriores sobre un escenario completamente funcional, con todos los componentes desempeñando todas las funciones que les corresponden. Por tanto, una vez arrancado la maqueta con todos los elementos, se realizarán 3 pings entre h1 y h4, donde se deberá observar lo siguiente:

- El primer ping llegando con éxito y los otros dos siendo descartados por s1. Para ello se observará la consola donde se realizan los pings y se capturará mediante Wireshark en las interfaces del switch s1 conectadas a h1 y s6, para poder comprobar el proceso de descarte.
- Las alarmas generadas por el IDS y las respuestas correspondientes por parte del controlador. Las alarmas serán expuestas en la consola del controlador, al igual que las medidas tomadas. Asimismo, las medidas se plasmarán en las tablas de flujo de los switches y también en la captura de Wireshark mencionada en el punto anterior.

Si estas condiciones se cumplen, se podrá asegurar que el resto de funcionalidades se están cumpliendo según lo esperado: si el primer ping llega significa que las tareas básicas de un switch de nivel 2 y el protocolo STP están funcionando adecuadamente, mientras que el bloqueo de los pings posteriores implica que el IDS recibe el tráfico de la red, genera las alarmas correspondientes y las hace llegar al controlador, que toma cartas en el asunto.

7 Descripción de la solución

7.1 Introducción

En este apartado se describe la solución desarrollada para integrar un IDS en una red SDN, siguiendo los diseños realizados en el apartado Diseño de la solución. Para ello, la solución propuesta consiste en desplegar en Mininet una red SDN gestionada por el controlador Ryu. En dicha red habrá un equipo con Snort funcionando como IDS que alertará de las amenazas detectadas al controlador, para que éste responda de la forma más adecuada a la situación.

Por tanto, tenemos tres elementos clave a tener en cuenta en la solución:

- La maqueta de red desplegada en Mininet
- El IDS Snort ejecutado en uno de los equipos de la red
- El controlador Ryu

Cada uno de los tres elementos realiza funciones concretas en el sistema e interactúa con el resto de elementos:

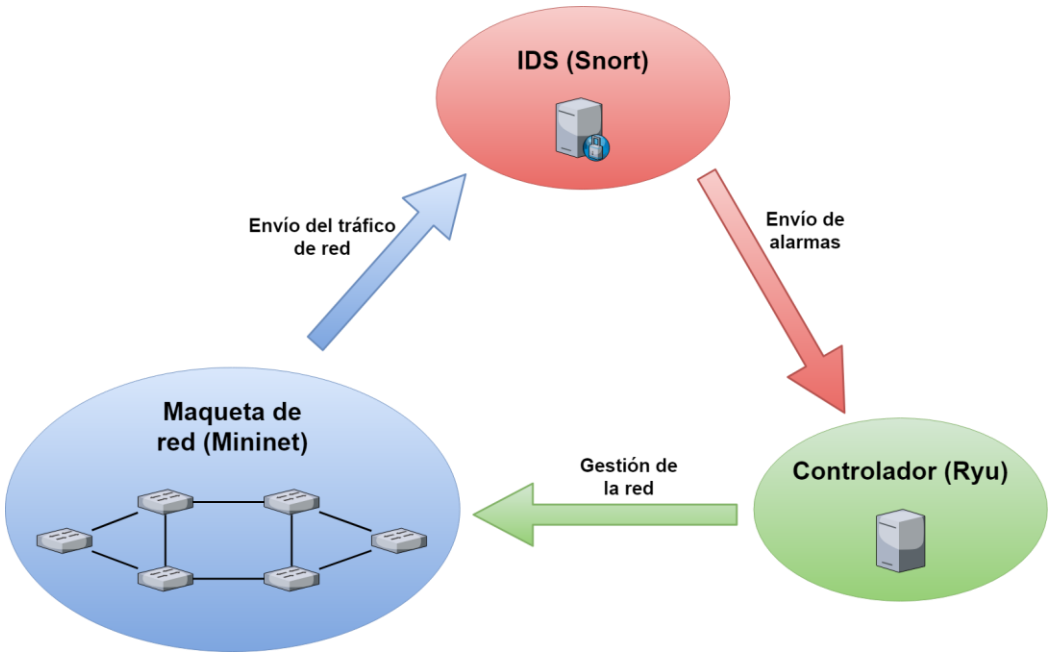

Ilustración 8.- Esquema de la solución propuesta

En este escenario, la maqueta de red de Mininet realiza las funciones de reenvío de paquetes del plano de control, mientras que el controlador Ryu es el encargado del plano de control. Sobre esta arquitectura habitual de las redes SDN se ha integrado el IDS Snort, encargado de dotar de seguridad al sistema.

Como se puede observar en la Ilustración 8, la interacción de los elementos se da de manera cíclica. En primer lugar, la red de Mininet envía todo el tráfico generado al IDS para que Snort analice los paquetes. Con la información recibida, Snort detecta los paquetes maliciosos y genera alarmas para enviarlos al controlador. Cuando el controlador Ryu recibe las alarmas del IDS, genera una respuesta al problema detectado, habitualmente añadiendo entradas a las tablas de flujo de los switches del plano de datos. Este ciclo se repite para cada paquete de tráfico generado.

7.2 Despliegue del escenario de desarrollo

En este apartado se detalla el despliegue de los dos escenarios de desarrollo definidos en la fase de diseño. Cada uno de ellos se ha utilizado con un objetivo concreto: por una parte, se ha utilizado un primer escenario sencillo para realizar pruebas de diversa índole y, por otra parte, se ha hecho uso de un escenario más complejo para integrar la solución final. Ambas maquetas de red se han definido utilizando Mininet [6] [7].

7.2.1 Escenario 1

Con la finalidad de probar ciertas funcionalidades del controlador y del IDS antes de incorporarlos al escenario final, se ha empleado una maqueta sencilla que permite experimentar fácilmente con distintas funcionalidades individuales del controlador que se desean agregar a la solución final: reenvío de paquetes al IDS, recepción de alertas, etc. Para ello, se ha desplegado una topología de red con un switch, tres equipos finales y un IDS conectados al mismo y un controlador SDN.

El escenario 1 se puede observar en la Ilustración 9 tal y como se ve en la interfaz gráfica de Mininet, MiniEdit [8]. La herramienta MiniEdit es muy útil para crear y editar topologías SDN para Mininet, sobre todo al principio del proyecto. Una vez avanzado el trabajo, la API de Python de Mininet ofrece muchas más posibilidades que la GUI, aunque MiniEdit sigue siendo de utilidad para visualizar la red de manera gráfica.

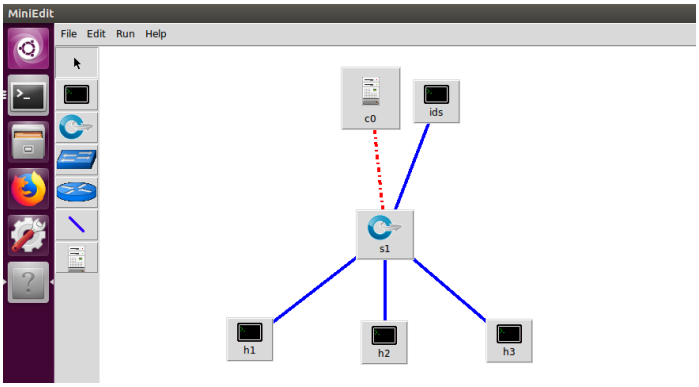

Ilustración 9.- Escenario de pruebas

7.2.2 Escenario 2

El escenario 2 es el que se ha utilizado para implementar la solución final del proyecto. Este escenario es más complejo que el primero y permite estudiar el comportamiento de la red en situaciones más reales. Los elementos que componen la red son los siguientes:

- 6 switches interconectados mediante enlaces redundantes
- 4 equipos finales conectados a los switches
- 1 equipo realizando las funciones de IDS, conectado a todos los switches
- 1 controlador SDN conectado a los switches

Cada uno de estos elementos tiene funciones concretas en la maqueta:

- **Switches:** como en las redes SDN el plano de control queda a cargo del controlador, los switches se encargan de las funcionalidades del plano de datos, es decir, todo lo relacionado con la transmisión de paquetes. Los switches SDN funcionan en base a unas tablas denominadas *flow tables* o tablas de flujo. Estas tablas establecen ciertas

condiciones para los paquetes recibidos, y las acciones a realizar en caso de que un paquete coincida con dicho criterio. Si no se da ninguna coincidencia, todos los switches tienen un *table miss*, que indica la acción a ejecutar en estos casos. Lo más habitual suele ser enviar el paquete al controlador para que éste tome una decisión al respecto.

- **Equipos finales:** son los equipos equivalentes a los de usuarios finales. La tarea principal de los hosts es simular el tráfico que se podría dar en situaciones reales: mensajes ICMP, consultas HTTP… Con esto se pretende comprobar que la respuesta de la red a diferentes tipos de paquetes es la adecuada en cada caso.
- **IDS:** es el equipo encargado de detectar posibles amenazas en la red. Para ello, todos los switches reenvían el tráfico recibido a este equipo, de forma que el IDS puede analizar el tráfico de la red al completo. En redes tradicionales, un IDS suele alertar de las amenazas identificadas al administrador de red. Sin embargo, las redes SDN permiten enfocar la situación de otra manera: si el controlador SDN recibe las alarmas del IDS, puede ser capaz de resolver el problema sin necesidad de intervención humana. El funcionamiento del IDS se expondrá de forma más extensa en el apartado Instalación y configuración del IDS.
- **Controlador SDN:** es el elemento más importante de una red SDN, pues toda la inteligencia de red esta centralizada en este dispositivo. El controlador mantiene una visión global de la red y gestiona todos los switches a su cargo. Funciones habituales del controlador incluyen la modificación de las *flow tables*, recepción de paquetes que han llegado al *table miss*, configuración de nuevos switches, etc. En este proyecto, además de todas esas funciones, el controlador realizara tareas más específicas como la gestión del protocolo STP, la recepción de alertas del IDS o la respuesta a amenazas detectadas por el IDS. El controlador se explicará con todo detalle en el apartado Instalación y configuración del controlador.

Todos estos elementos conforman el escenario final utilizado en el proyecto. La red se ha diseñado para aproximarse lo máximo posible a una situación real, con equipos en diferentes localizaciones, una red *core* redundante, un IDS como sistema de seguridad y un controlador para gestionar toda la red.

Para crear esta maqueta de red se ha hecho uso de la API de Python de Mininet. El código generado se ha adjuntado en el Anexo II: Código desarrollado de este documento. Por otra parte, en la Ilustración 10 se puede observar la topología visualizada en MiniEdit:

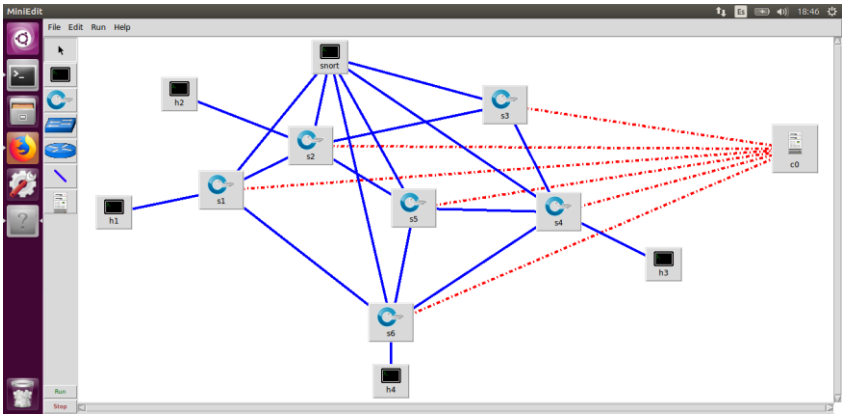

Ilustración 10.- Escenario final

7.3 Instalación y configuración del IDS

El Sistema de Detección de Intrusión es el elemento de la red que realiza las funciones de seguridad. El IDS monitoriza el tráfico de la red, y genera alarmas en base a una serie de reglas establecidas previamente. El IDS que se ha utilizado en este proyecto se denomina Snort [9]. A continuación, se hará más hincapié en la instalación y la configuración de este IDS.

7.3.1 Instalación y puesta en marcha

Para la instalación de Snort, se han seguido los pasos indicados en la guía oficial de la web de Snort [10]. Han sido de especial interés los apartados 7-12 de dicha guía, pues están orientados a instalar y probar una configuración básica de Snort. Estos capítulos incluyen la instalación de librerías y dependencias, la instalación de Snort, una configuración mínima del IDS y una prueba de funcionamiento.

Si se procede con los pasos establecidos en la guía con esmero, se llega a instalar Snort sin ningún problema. Se puede ver si la instalación ha tenido éxito ejecutando `snort -V` en consola:

```
iban@iban-VirtualBox:~$ snort -V

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.11.1 GRE (Build 268)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2017 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.7.4
           Using PCRE version: 8.38 2015-11-23
           Using ZLIB version: 1.2.8
```

Ilustración 11.- Comprobación de Snort

7.3.2 Configuración

En cuanto a la configuración de Snort, la variedad de opciones que permite utilizar es muy amplia. Sin embargo, para la prueba de concepto llevada a cabo en este proyecto se ha hecho uso de una configuración relativamente básica.

Tal y como se ha mencionado anteriormente, Snort funciona en base a una serie de reglas establecidas. Estas reglas son uno de los aspectos más importantes a la hora de configurar Snort. Las reglas de Snort suelen tener la siguiente estructura:

[action] [protocol] [sourceIP] [sourceport] -> [destIP] [destport] ([Rule options])

La parte destacada en negrita se denomina *rule header*, y aparece en todas las reglas de Snort. Los elementos que contiene son los siguientes:

- **action**: indica la acción que debe realizar Snort en el caso de que se encuentre un paquete que cumple con los criterios de la regla.
- **protocol**: indica el protocolo que se analiza para buscar actividad sospechosa. Actualmente Snort analiza cuatro protocolos: TCP, UDP, ICMP e IP.
- **sourceIP**: dirección de origen. Puede ser literal o variable.
- **sourceport**: puerto de origen. Puede ser literal o variable.
- **"->"**: operador de dirección. Indica en qué dirección debe ir el tráfico de paquetes para que se aplique la regla.
- **destIP**: dirección de destino. Puede ser literal o variable.
- **destport**: puerto de destino. Puede ser literal o variable.

Además de los campos que incluye el *rule header*, las reglas de Snort pueden tener más opciones, que se indican al final de la regla entre paréntesis. Si se indican varias opciones, éstas se separan mediante punto y coma. Algunas de las opciones más habituales son las que se definen a continuación:

- **message (msg)**: es una simple cadena de texto que indica a Snort que debe mostrar a la salida cuando la regla coincida. Normalmente este mensaje indica qué está detectando la regla.
- **flow**: indica la dirección en la que tiene que ir el tráfico de la red para que la regla se dispare.
- **reference**: gracias a esta palabra clave es posible incluir referencias a fuentes externas de información.
- **classtype**: indica mediante una palabra clave el efecto que tendría el ataque que intenta detectar la regla si fuera exitoso.
- **sid/rev**: el Snort id (sid) es un identificador único que tiene cada regla, lo que permite identificar la regla fácilmente. Normalmente se utiliza junto con el número de revisión (rev).
- **detection**: además de los parámetros indicados en el *rule header*, permite indicar otros criterios más concretos para realizar la búsqueda de coincidencia (contenido, *offset*, tamaño en bytes…).

Es muy importante definir correctamente las reglas que se vayan a configurar, ya que depende de ello que el sistema de detección sea capaz de detectar las intrusiones. En este caso, para comprobar el correcto funcionamiento del IDS se ha configurado una regla de prueba que genera una alarma en caso de detectar un paquete ICMP. La regla en cuestión es la siguiente:

```
alert icmp any any -> $HOME_NET any (msg:“ICMP test detected”; sid:10000001; rev:001; classtype:icmp-event;)
```

Teniendo en cuenta el formato de las reglas detallado más arriba en este apartado, esta regla se puede interpretar de la siguiente manera: el IDS alertará de los paquetes ICMP cuya dirección destino sea una dirección de la red interna, independientemente de la dirección origen y de los puertos utilizados. Cuando un paquete cumpla con estas condiciones saltará un mensaje informativo indicado en la opción *msg*. El resto de parámetros utilizados sirven para llevar una clasificación de las reglas, aunque su utilidad aumenta según vaya incrementando el número de reglas aplicadas.

De esta manera, cualquier paquete ICMP que se envíe por la red hará saltar la alarma. Por tanto, solo hace falta realizar un ping entre dos equipos para ver si el IDS funciona de forma adecuada.

Por otra parte, el fichero de configuración de Snort (por defecto en */etc/snort/snort.conf*) también permite personalizar el funcionamiento del IDS. Si se han seguido todos los pasos de la guía de instalación, el único parámetro que habrá que ajustar es el *&HOME_NET* que se ha utilizado en la regla de prueba. Esta variable permite indicar a Snort el rango de direcciones IP que se quiere proteger mediante el IDS. Si se quiere una explicación detallada de las distintas posibilidades de configuración que incluye Snort, lo más sencillo es recurrir al manual de usuario de la página web oficial [11].

Una vez se ha configurado correctamente el Snort, es necesario ponerlo en marcha para que comience a analizar el tráfico de la red. Para ello, solo hace falta ejecutar un comando:

```
sudo /usr/local/bin/snort -A unsock -q -i snort-eth0 -c /etc/snort/snort.conf
-l /tmp
```

A continuación, se explica la función de los diferentes parámetros que recoge este comando:

- -A unsock: envía las alertas utilizando sockets de Unix
- -q: modo silencio (no muestra el informe de estadísticas)
- -i snort-eth0: indica la interfaz en la que debe escuchar Snort
- -c /etc/snort/snort.conf: señala el directorio donde se encuentra el fichero de configuración de Snort
- -l /tmp: indica el directorio donde se quieren guardar los logs

7.4 Instalación y configuración del controlador

El controlador SDN es el elemento más significativo de cualquier red SDN. La inteligencia de red está centralizada en el controlador, y éste es el encargado de gestionar el plano de control. En este escenario en concreto sus funcionalidades son varias:

- **Funciones básicas:** son las funcionalidades necesarias en cualquier red SDN, como la gestión de tablas de flujo de los switches, la recepción de paquetes que no coinciden con las entradas de las tablas…
- **Protocolo STP:** tratándose de una red con enlaces redundantes es necesario gestionar el protocolo del árbol de expansión para activar o bloquear rutas, asignar puentes raíz y designados, etc.
- **Recepción de alarmas:** es necesario habilitar una comunicación entre el IDS y el controlador para que el segundo reciba las alertas generadas por el primero.
- **Programación de respuestas:** se debe programar el controlador para establecer los protocolos de respuesta frente a las alarmas recibidas del IDS.

A continuación, se detallará la configuración de todas estas funcionalidades en el controlador elegido para este proyecto: el controlador Ryu [12]. El código completo del controlador se puede ver en el Anexo II: Código desarrollado.

7.4.1 Funciones básicas

Todas las aplicaciones de Ryu suelen tener una estructura bastante parecida en cuanto a los módulos que incluyen [13]. Habitualmente, las aplicaciones del controlador se componen de tres segmentos principales:

- **Inicialización de la aplicación:** esta parte se ejecuta antes de que el controlador empiece a atender a los switches. En esta primera fase se inicializan los datos que se comparten para toda la red. Un ejemplo puede ser la tabla MAC a puerto que se utiliza para el aprendizaje de direcciones MAC.
- **Inicialización de los switches:** este fragmento del código se emplea para definir configuraciones estáticas en los switches que se conectan a la red. Por ejemplo, la utilidad más básica de esta etapa puede ser añadir la entrada del *table-miss* en los switches, de manera que los paquetes que no coincidan con ninguna entrada sean enviados al controlador.
- **Gestión de paquetes entrantes:** esta función está relacionada con el ejemplo del punto anterior. Si un switch no sabe qué hacer con un paquete, lo habitual es enviarlo al controlador. En esta parte del código se indica al controlador qué debe hacer con los paquetes sin identificar que recibe de los switches.

Inicialización de la aplicación

Esta primera sección crea la aplicación del controlador (línea 40), indica la versión del protocolo OpenFlow a utilizar (línea 41) e inicializa las variables que se emplearán en el resto de módulos de la aplicación (líneas 42-64).

```
35: ### FUNCIONES BÁSICAS ###
36:
37: ### Inicialización de la aplicación ###
38:
39: # Definir el controlador
```

```
40: class switchStpSnortFix(simple_switch_13.SimpleSwitch13):
41:     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
42:       _CONTEXTS = {'stplib': stplib.Stp, 'snortlib': snortlib.SnortLib,
'dpset': dpset.DPSet,}
43:
44: # Definir variables
45:     def __init__(self, *args, **kwargs):
46:         super(switchStpSnortFix, self).__init__(*args, **kwargs)
47:         self.mac_to_port = {}
48:         self.stp = kwargs['stplib']
49:         self.snort = kwargs['snortlib']
50:         self.snort_port = 1
51:         self.dpset = kwargs['dpset']
52:
53:         socket_config = {'unixsock': True}
54:
55:         self.snort.set_config(socket_config)
56:         self.snort.start_socket_server()
57:
58:         config = {dpid_lib.str_to_dpid('0000000000000001'):
59:                   {'bridge': {'priority': 0x8000}},
60:                   dpid_lib.str_to_dpid('0000000000000002'):
61:                   {'bridge': {'priority': 0x9000}},
62:                   dpid_lib.str_to_dpid('0000000000000003'):
63:                   {'bridge': {'priority': 0xa000}}}
64:         self.stp.set_config(config)
```

Mediante estas líneas de código se establece que la versión de OpenFlow utilizada será la 1.3. Por otro lado, se definen las variables necesarias en las demás secciones: variables para guardar la tabla MAC a puerto, para gestionar el protocolo STP, para recibir alarmas del Snort…

Inicialización de los switches

Se ha decidido que la configuración inicial de los switches conste de una entrada *table-miss*, de manera que si el paquete entrante no coincide con ninguna entrada de la tabla de flujo del switch ese paquete se hace llegar al controlador.

```
66: ### Inicialización de los switches ###
67:
68: # Evento configuración nuevo switch
69:     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
70:     def switch_features_handler(self, ev):
71:         datapath = ev.msg.datapath
72:         ofproto = datapath.ofproto
73:         parser = datapath.ofproto_parser
74:
75:         # Instalar entrada table-miss en el switch
76:         match = parser.OFPMatch()
77:               actions  =  [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)]
78:         self.add_flow(datapath, 0, match, actions)
```

Para configurar esta entrada en el switch, primero se extraen los datos del switch que ha desencadenado el evento de nuevo switch (líneas 69-73). Una vez que se tiene conocimiento del nuevo switch, se añade una entrada sin condiciones (para que coincida con todos los paquetes) y con la menor prioridad posible, cuya acción es enviar el paquete al controlador

(líneas 76-78). Con esta entrada se consigue configurar el *table-miss* para el switch que se ha conectado a la red.

Gestión de paquetes entrantes

El propósito de este segmento es que el controlador se haga cargo de los paquetes que los switches no saben manejar (porque no coinciden con ninguna entrada de las tablas de flujo). Para ello, el controlador debe crear nuevos flujos y entradas para las tablas de flujo, de manera que la próxima vez el switch sepa lo que debe hacer con dicho flujo. Las entradas que creará esta aplicación para el controlador son equivalentes a métodos habituales en las redes tradicionales, como la transmisión por inundación y el aprendizaje de direcciones MAC.

```
080: ### Gestión de paquetes entrantes ###
081:
082: # Evento paquete entrante al controlador
083:     @set_ev_cls(stplib.EventPacketIn, MAIN_DISPATCHER)
084:     def _packet_in_handler(self, ev):
085:         msg = ev.msg
086:         datapath = msg.datapath
087:         ofproto = datapath.ofproto
088:         parser = datapath.ofproto_parser
089:         in_port = msg.match['in_port']
090:
091:         pkt = packet.Packet(msg.data)
092:         eth = pkt.get_protocols(ethernet.ethernet)[0]
093:         dst = eth.dst
094:         src = eth.src
095:
096:         dpid = datapath.id
097:         self.mac_to_port.setdefault(dpid, {})
098:
099:         # Aprender la dirección MAC para evitar inundación
100:         self.mac_to_port[dpid][src] = in_port
101:
102:         if dst in self.mac_to_port[dpid]:
103:             out_port = self.mac_to_port[dpid][dst]
104:         else:
105:             out_port = ofproto.OFPP_FLOOD
106:
107:                             actions   =   [parser.OFPActionOutput(out_port),
parser.OFPActionOutput(self.snort_port)]
108:
109:         # Instalar flow para evitar paquete entrante
110:         if out_port != ofproto.OFPP_FLOOD:
111:             match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
112:             self.add_flow(datapath, 1, match, actions)
113:
114:         data = None
115:         if msg.buffer_id == ofproto.OFP_NO_BUFFER:
116:             data = msg.data
117:
118:                             out   =   parser.OFPPacketOut(datapath=datapath,
buffer_id=msg.buffer_id, in_port=in_port, actions=actions, data=data)
119:         datapath.send_msg(out)
```

El código de esta sección se puede dividir a su vez en funcionalidades más simples. Primero, se extraen todos los datos correspondientes al paquete recibido del switch (líneas 83-97). Después,

se ejecuta el aprendizaje de las direcciones MAC (líneas 100-107). Para ello, se consulta la tabla MAC-puerto que se guarda en el controlador, para comprobar si existe una entrada para esa dirección MAC o no. En caso afirmativo, significa que la MAC es conocida pero el switch todavía no tiene añadida la entrada para dicha dirección. En caso negativo, será necesario hacer llegar el paquete por inundación. Al mismo tiempo, se aprovecha la dirección MAC origen del paquete para añadir una nueva entrada a la tabla MAC-puerto. Por último, se añade una entrada al switch que ha desencadenado el evento, para que la próxima vez pueda manejar el paquete sin ayuda del controlador (líneas 110-112). Asimismo, se le dan instrucciones al switch para que transmita el paquete que ha enviado al controlador al destinatario original, de forma que el paquete en cuestión no se pierda durante el proceso de aprendizaje (líneas 114-119).

Por otra parte, en esta sección de código hay un detalle muy relevante relacionado con la recepción de alarmas que se explicará más adelante. En la línea 107, a la hora de indicar la acción que debe realizar el switch con los paquetes que coincidan con la entrada que se está configurando, se establece que tiene que enviar el paquete por el puerto correspondiente al destino, pero también por el puerto conectado al IDS. Esto hace que todo el tráfico que pase por el switch se duplique y se envíe también a Snort, para que éste pueda monitorizar el tráfico de red.

7.4.2 Protocolo STP

El protocolo del árbol de expansión es un protocolo que permite gestionar redes con enlaces redundantes bloqueando y activando los diferentes puertos de los switches. Para emplear el protocolo STP en esta aplicación se ha utilizado la librería *stplib* que incluye el controlador Ryu. La librería se encarga de realizar las funciones relacionadas con el protocolo, por lo que la aplicación solo tiene que atender los eventos que genera. En este caso, se deben atender los eventos de cambio de topología (líneas 124-133) y de cambio de estado de puerto (líneas 136-145).

```
121: ### PROTOCOLO STP ###
122:
123: # Evento cambio de topología
124:     @set_ev_cls(stplib.EventTopologyChange, MAIN_DISPATCHER)
125:     def _topology_change_handler(self, ev):
126:         dp = ev.dp
127:         dpid_str = dpid_lib.dpid_to_str(dp.id)
128:         msg = 'Receive topology change event. Flush MAC table.'
129:         self.logger.debug("[dpid=%s] %s", dpid_str, msg)
130:
131:         if dp.id in self.mac_to_port:
132:             self.delete_flow(dp)
133:             del self.mac_to_port[dp.id]
134:
135: # Evento cambio de estado de puerto
136:     @set_ev_cls(stplib.EventPortStateChange, MAIN_DISPATCHER)
137:     def _port_state_change_handler(self, ev):
138:         dpid_str = dpid_lib.dpid_to_str(ev.dp.id)
139:         of_state = {stplib.PORT_STATE_DISABLE: 'DISABLE',
140:                     stplib.PORT_STATE_BLOCK: 'BLOCK',
141:                     stplib.PORT_STATE_LISTEN: 'LISTEN',
142:                     stplib.PORT_STATE_LEARN: 'LEARN',
143:                     stplib.PORT_STATE_FORWARD: 'FORWARD'}
144:         self.logger.debug("[dpid=%s][port=%d] state=%s",
```

```
145:                                 dpid_str, ev.port_no, of_state[ev.port_state])
```

En el caso de cambio de topología, el controlador resetea las entradas de la tabla de flujo del switch, ya que en caso de no hacerlo se corre el riesgo de que las entradas antiguas interfieran con las nuevas. En caso de cambio de estado de puerto, se muestran en consola los estados de todos los puertos según vayan ocurriendo cambios en los mismos.

Para mayor información sobre el funcionamiento del protocolo STP o la librería *stplib* de Ryu, se puede consultar la página web indicada en [14].

7.4.3 Recepción de alarmas

Para la recepción de las alarmas procedentes del IDS, se ha empleado la librería *snortlib* incluida en el controlador Ryu. La librería se encarga de enviar las alertas generadas al controlador, por tanto, la aplicación debe indicar qué hacer con las alertas recibidas. Para configurar el Snort de manera que la librería pueda funcionar de forma adecuada, se han seguido los pasos expuestos en [15]. En la implementación final se ha planteado el caso 1 de dicha referencia, en el que Snort y Ryu se ejecutan en el mismo equipo, aunque la opción 2 también se ha puesto a prueba durante el proyecto.

```
147: ### RECEPCIÓN DE ALARMAS ###
148:
149: # Evento alerta de Snort
150:     @set_ev_cls(snortlib.EventAlert, MAIN_DISPATCHER)
151:     def _dump_alert(self, ev):
152:         msg = ev.msg
153:         pkt = packet.Packet(array.array('B', msg.pkt))
154:         _ipv4 = pkt.get_protocol(ipv4.ipv4)
155:
156:         if _ipv4:
157:               print('alertmsg: %s' % ''.join(msg.alertmsg))
158:               self.packet_print(msg.pkt)
159:
160:             # Llamada a respuesta frente a alarma
161:               self.fix_alert(ev)
```

Lo primero que se hace al recibir la alerta es obtener información sobre la misma (líneas 150-154). Después, para cada alerta recibida por el controlador se realizan dos acciones: por un lado, se muestra en consola la alerta generada y su información correspondiente (líneas 157-158) y, por otro lado, se llama a otro módulo para intentar responder a la amenaza detectada (línea 161).

7.4.4 Programación de respuestas

Una de las novedades que aporta este Trabajo Fin de Grado es la respuesta programada del controlador frente a las amenazas detectadas por el IDS. Se cree que el objetivo principal es estudiar la implementación de esta función más que considerar todos los ataques y respuestas posibles, por ello, se ha programado una respuesta para la regla de prueba definida en el IDS, que genera alarmas al detectar paquetes ICMP. De esta manera, además, será muy sencillo realizar una prueba de la respuesta del controlador haciendo uso de pings.

```
163: ### PROGRAMACIÓN DE RESPUESTAS ###
164:
165: # Programar respuesta frente a alerta del Snort
166:     def fix_alert(self, ev):
167:         msg = ev.msg
```

```
168:         pkt = msg.pkt
169:         pkt = packet.Packet(array.array('B', pkt))
170:         _ipv4 = str(pkt.get_protocol(ipv4.ipv4))
171:
172:         srcIP = _ipv4.split("'")[3]
173:
174:         print('IP_origen= %s' %srcIP)
175:         print('Bloqueando la IP %s...' %srcIP)
176:
177:         dp_set = self.dpset.get_all()
178:
179:         i = 0
180:         for dp in dp_set:
181:               i += 1
182:               datapath = dp[1]
183:
184:               parser = datapath.ofproto_parser
185:               match = parser.OFPMatch(eth_type=0x800, ipv4_src=srcIP)
186:               actions = []
187:               self.add_flow(datapath, 2, match, actions)
188:
189:               print('Flow anadido a s-%d' %i)
190:
191:         print('Problema arreglado')
```

Respecto a la respuesta del controlador, se ha decidido que Ryu identifique el equipo que origina los paquetes maliciosos y bloquee la dirección IP de dicho equipo. Para ello, extrae información sobre el paquete de la alerta recibida (líneas 166-172), obteniendo la dirección IP del equipo origen. Después, añade una entrada a las tablas de flujo de los switches (líneas 177-187), definiendo como condición la dirección IP en cuestión y estableciendo como acción a realizar descartar el paquete (una acción nula como la indicada en el código es equivalente al descarte del paquete). De esta forma, a partir de ese momento el tráfico generado por el equipo con esa dirección IP queda bloqueado en la red.

Aunque en este caso se ha decidido filtrar por la dirección IP, es posible poner reglas más estrictas para la respuesta del controlador. Lo único que cambia del código es la extracción de los datos necesarios para definir la condición y la condición en sí, establecida en la línea 185. Los campos que se pueden utilizar para indicar la condición se pueden consultar en las especificaciones de OpenFlow 1.3 [3].

7.5 Funcionamiento

Una vez se han detallado todos los elementos que componen la implementación final, se procederá a ilustrar el funcionamiento del escenario final. Para ello, se utilizarán ilustraciones que representan instantáneas tomadas en diferentes etapas del proceso. Mencionar aquí que el IDS y el controlador se han separado para mayor claridad en las ilustraciones, pero en la solución diseñada y desarrollada se encuentran en el mismo equipo físico.

1.- Situación inicial

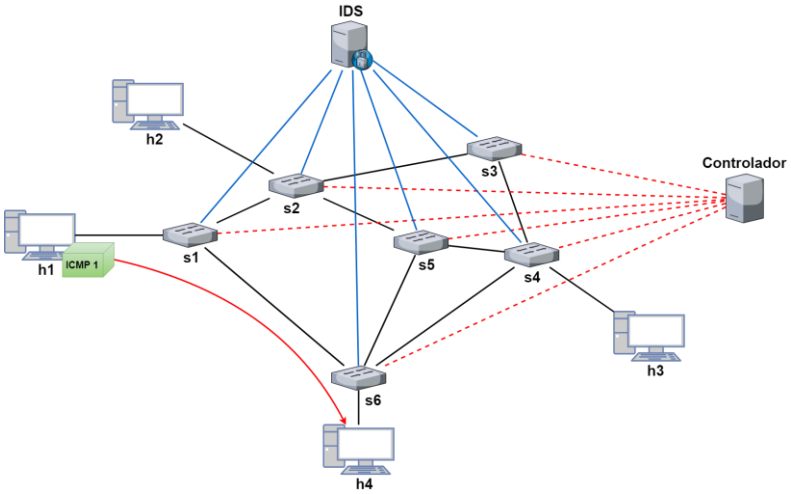

Ilustración 12.- Funcionamiento - situación inicial

En esta primera imagen se muestra la situación inicial: el equipo final h1 desea enviar varios paquetes ICMP al equipo final h4. Para ello, los paquetes en cuestión deberán pasar por los switches s1 y s6 respectivamente[1]. Se ha elegido un paquete ICMP para la prueba de manera que genere la alerta configurada en el IDS.

2.- H1 envía ICMP 1

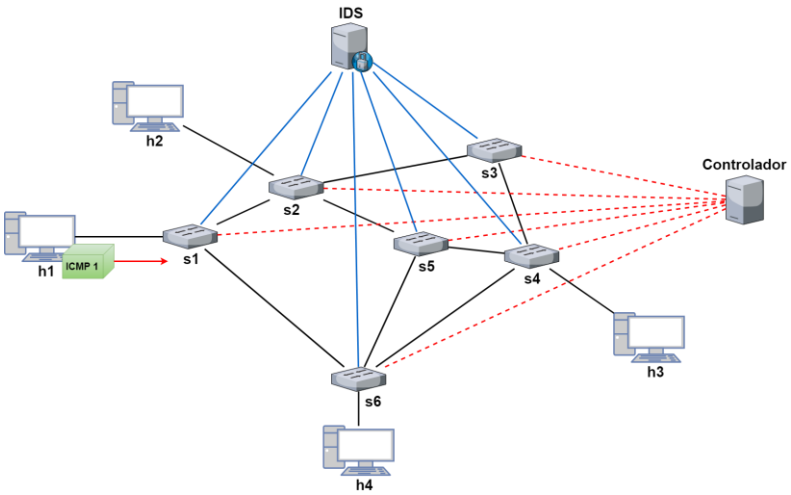

Ilustración 13.- Funcionamiento - h1 envía ICMP 1

[1] Teniendo en cuenta que todos los enlaces tienen la misma capacidad, el protocolo STP establecerá la ruta más corta, es decir, la que pasa por s1 y s6.

El host h1 envía el paquete ICMP 1 al switch s1 al que está conectado. Considerando que es el primer paquete que recibe s1 desde que se ha arrancado la maqueta, las tablas de flujo del switch estarán vacías, a excepción del *table-miss* con el que se inicializan todos los switches. Por tanto, el paquete acabará coincidiendo con la entrada del *table-miss*, y el switch enviará el paquete al controlador para que éste le indique las acciones a realizar.

3.- *Table-miss* y adición de nuevo flujo

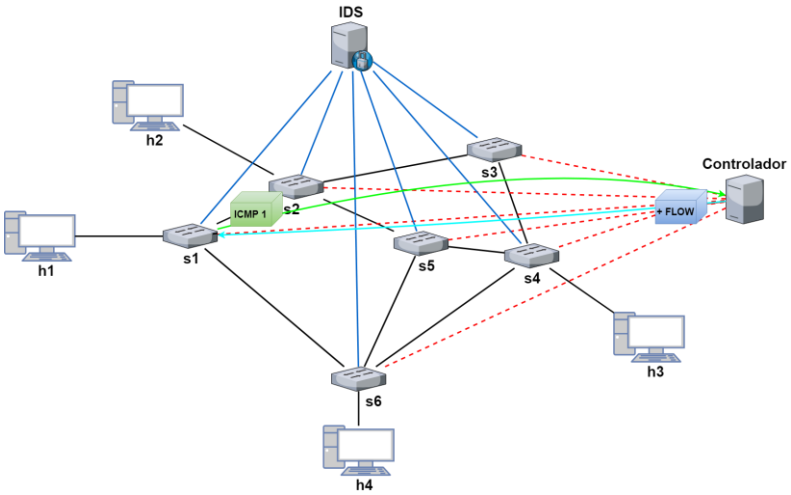

Ilustración 14.- Funcionamiento - *table-miss y adición de nuevo flujo*

El controlador recibirá el paquete y, después de analizarlo, configurará una entrada en las tablas de flujo del switch para que la próxima vez sepa gestionar el paquete. A su vez, dará instrucciones al switch para que ejecute de inmediato las acciones indicadas en la entrada, evitando así la pérdida del paquete original.

4.- Duplicación del paquete

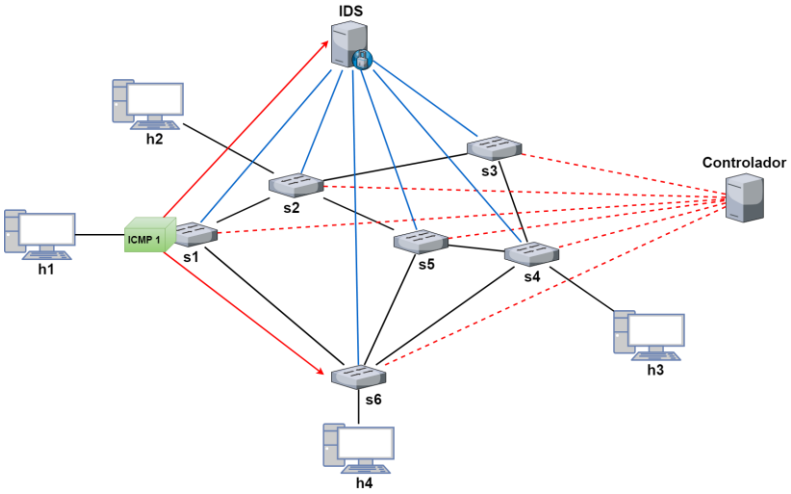

Ilustración 15.- Funcionamiento - duplicación del paquete

En este caso, para cada paquete se indican dos acciones: por un lado, enviar el paquete hacia el destino y, por otro lado, duplicar el paquete para enviarlo al IDS. Para mayor claridad en la ilustración, se ha supuesto que el controlador ya tiene una tabla MAC a puerto para indicar al switch el puerto por el que debe enviar el paquete. En caso contrario, el switch enviaría el paquete en cuestión mediante inundación a todos los dispositivos de red adyacentes.

5.- Generación y envío de alarma

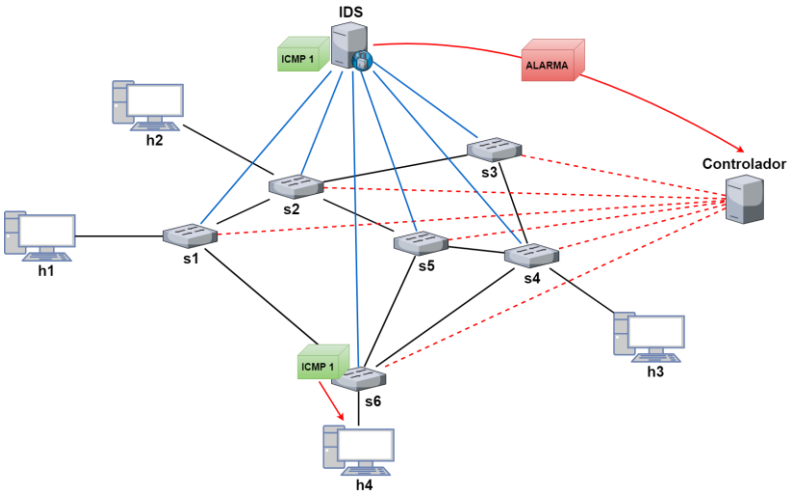

Ilustración 16.- Funcionamiento - generación y envío de alarma

Cuando el paquete llega al switch s6, éste seguirá el mismo procedimiento que se ha explicado para s1. Por tanto, aunque en la imagen se han omitido algunos pasos para no ser redundantes, el switch s6 tendría que enviar el paquete al controlador, recibir instrucciones y enviar el paquete al IDS y al equipo destino h4.

Al mismo tiempo, el IDS ha recibido el paquete duplicado enviado por s1 y se dispone a analizarlo. Como se trata de un paquete ICMP para el cual hay una regla definida, Snort generará una alarma y se la enviará al controlador.

6.- Respuesta a alarma

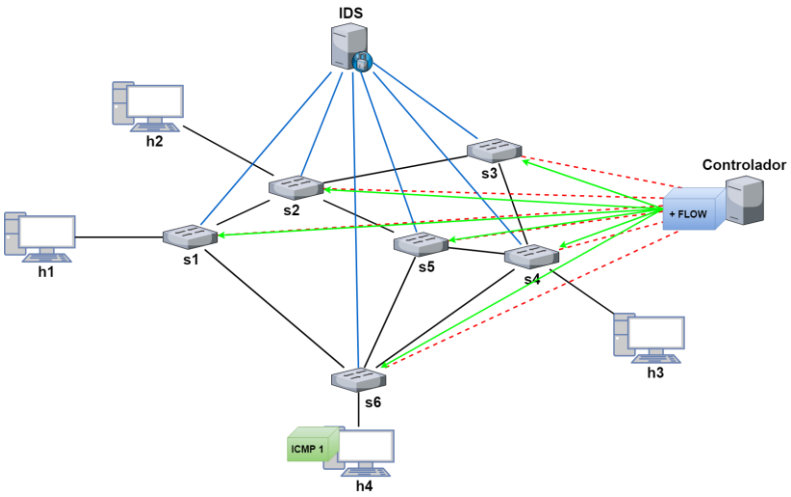

Ilustración 17.- Funcionamiento - respuesta a alarma

Cuando el controlador recibe la alarma del IDS, toma medidas para intentar solucionar el problema lo antes posible. Como se ha mencionado anteriormente, se ha diseñado la aplicación del controlador para bloquear la IP origen del paquete que ha desencadenado la alarma. Por tanto, el controlador añade una entrada en todas las tablas de flujo de los switches de la red indicando que, si se recibe un paquete con la dirección IP de h1 como origen, el paquete debe ser descartado.

Mientras tanto, el paquete ICMP 1 ha conseguido llegar a su destino. Esto era inevitable, ya que el tiempo que tarda un solo paquete en llegar al destino es menor que lo que tarda el controlador en añadir la regla que lo bloquee (el paquete debe llegar al IDS, el IDS generar la alarma y enviarla al controlador, y éste definir la respuesta y añadir las entradas de las tablas de flujo). A pesar de ello, se ha logrado evitar que un paquete con las mismas características sea enviado por la red en el futuro.

7.- Situación después de la respuesta

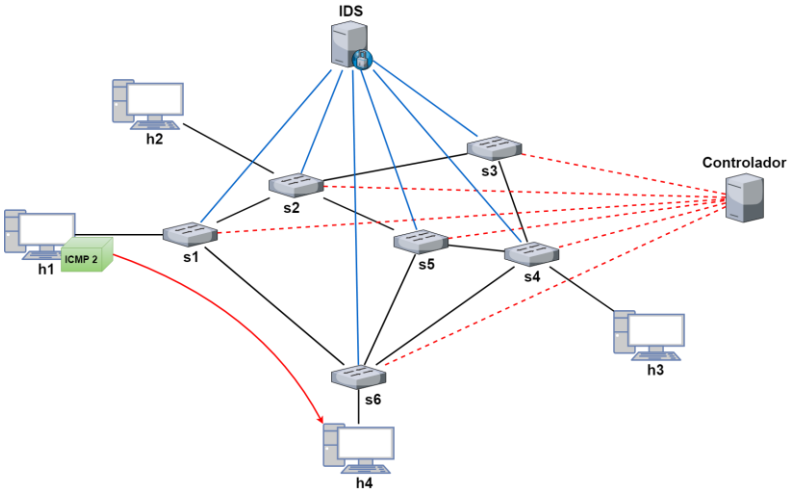

Ilustración 18.- Funcionamiento - situación después de la respuesta

Después de todo el proceso realizado para el paquete ICMP 1, el equipo h1 intenta enviar un segundo paquete (ICMP 2) con las mismas características. Sin embargo, la situación actual es distinta a la inicial, ya que se ha detectado que estos paquetes son maliciosos y se han tomado medidas para evitar su difusión por la red.

8.- H1 envía ICMP 2

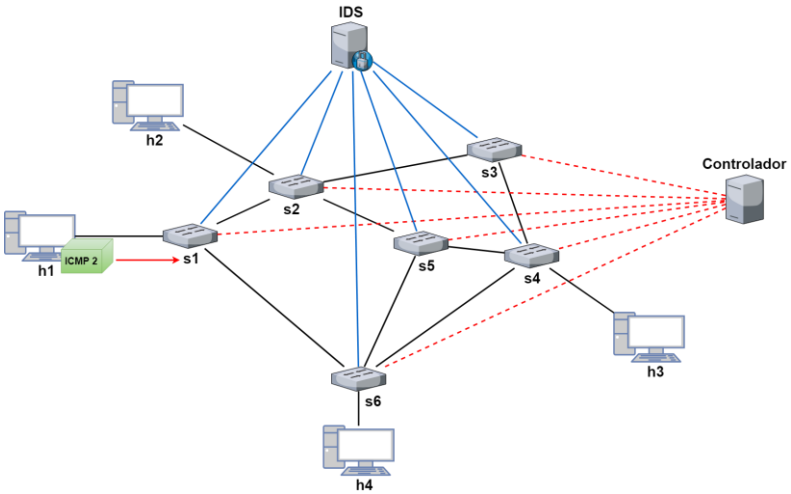

Ilustración 19.- Funcionamiento - h1 envía ICMP 2

Igual que con ICMP 1, h1 enviará el paquete al switch s1. No obstante, en este caso el switch dispone de una entrada en sus tablas de flujo que coincide con el paquete recibido (y no es el *table-miss*). Dicha entrada indica al switch que la acción con la que debe proceder es descartar el paquete y no realizar ninguna acción posterior.

9.- Descarte del paquete

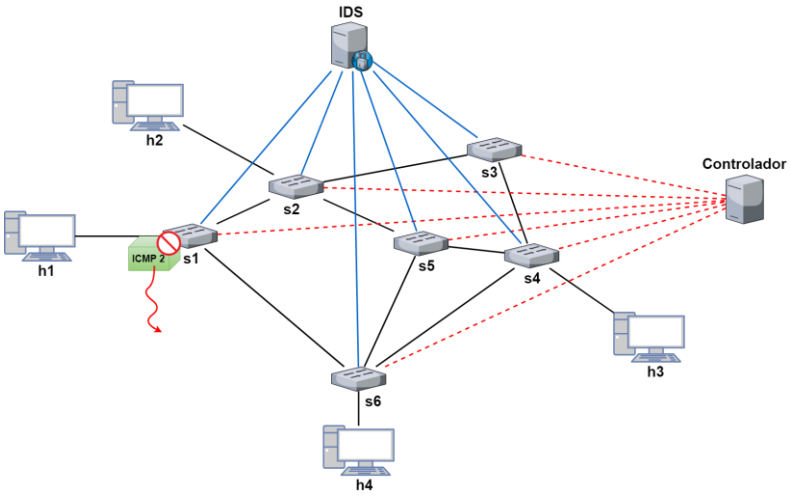

Ilustración 20.- Funcionamiento - descarte del paquete

Por tanto, el switch s1 descarta el paquete ICMP 2, al igual que cualquier paquete que provenga de h1 en el futuro (la entrada de la tabla de flujo del switch tiene la IP origen como condición). De esta manera, se ha conseguido solventar la situación sin intervención humana, solamente con la respuesta programada del controlador SDN.

8 Análisis de resultados de las pruebas

En este apartado se demostrará de forma práctica que el funcionamiento de la solución es el que se ha detallado en el apartado anterior, siguiendo para ello el plan de pruebas diseñado en el apartado de Diseño de la solución.

8.1 Escenario 1

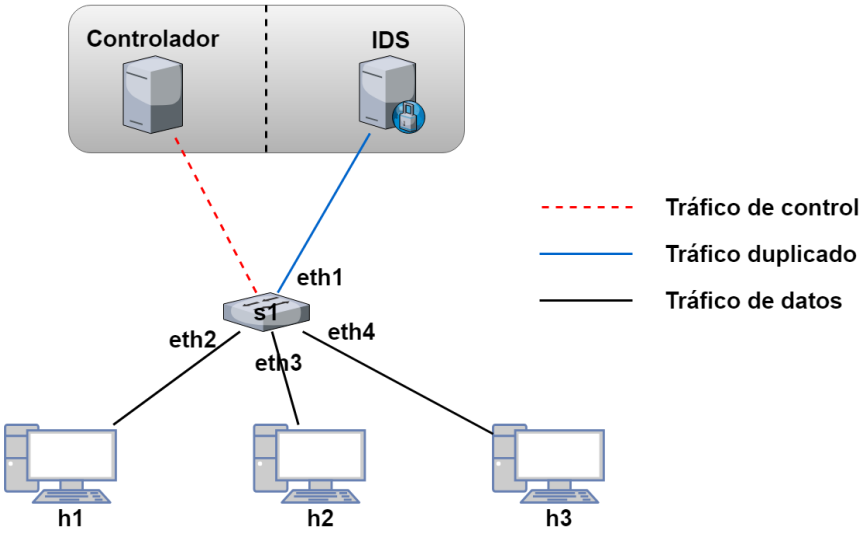

Ilustración 21.- Escenario de pruebas 1

8.1.1 Prueba 1 – Switch básico de nivel 2

Se han realizado desde la consola de Mininet 3 pings entre h1 y h2[2], capturando con Wireshark varias interfaces del switch s1. Se puede observar que el ping llega al switch por la interfaz eth2 del mismo y sale por eth3, por lo que el reenvío de paquetes es correcto. En la interfaz eth4 no se ha capturado nada, lo que significa que el aprendizaje MAC también se realiza correctamente.

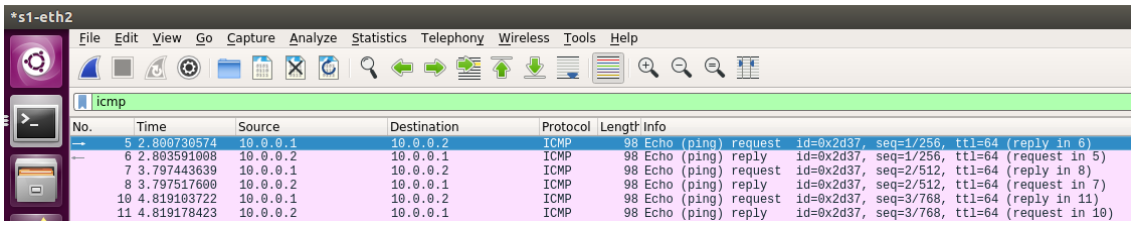

Ilustración 22.- Prueba 1 - captura s1_eth2

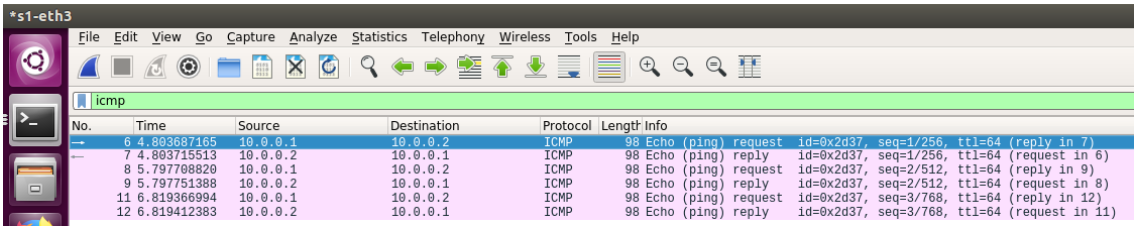

Ilustración 23.- Prueba 1 - captura s1_eth3

[2] El comando para realizar los pings desde Mininet es: `h1 ping -c 3 h2`

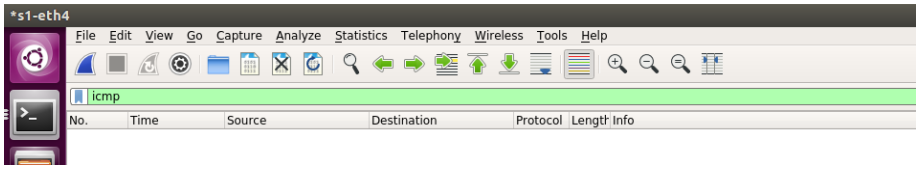

Ilustración 24.- Prueba 1 - captura s1_eth4

8.1.2 Prueba 2 – Reenvío del tráfico al IDS

Se han realizado 3 pings desde h1 a h2, capturando en la interfaz s1-eth1, conectada al IDS. Se puede observar en la captura que el switch ha copiado los paquetes recibidos y los ha enviado al IDS tal y como debería. Asimismo, se han presentado en la consola del IDS las alarmas generadas por los pings enviados (tanto las solicitudes como las respuestas son detectadas, de ahí que sean 6 alarmas y no 3), lo que confirma que el IDS ha recibido los paquetes enviados por el switch.

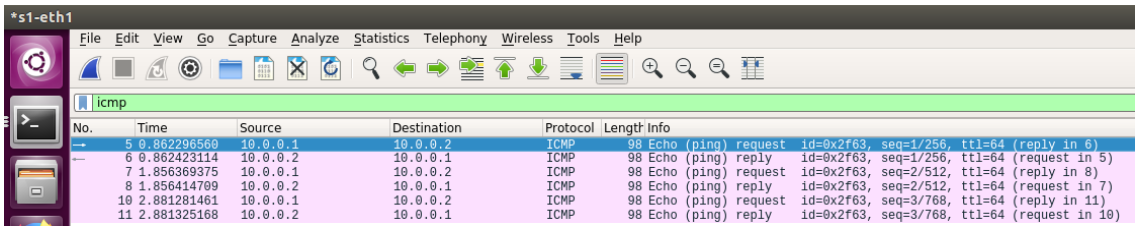

Ilustración 25.- Prueba 2 - captura s1_eth1

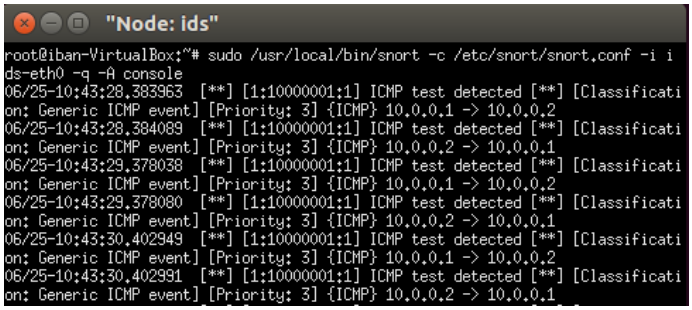

Ilustración 26.- Prueba 2 - consola IDS Snort

8.1.3 Prueba 3 – Recepción de alarmas del IDS

En esta prueba se ha seguido la misma metodología que la anterior, pero indicando al IDS que en lugar de mostrar las alarmas por consola las envíe al controlador. Las alarmas recibidas por Ryu se exponen en la consola del controlador, con información separada para cada protocolo utilizado en las diferentes capas que componen el paquete ICMP.

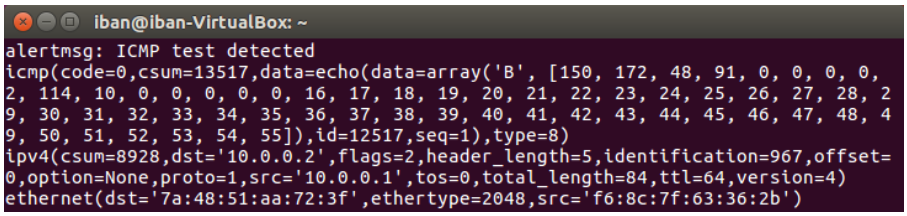

Ilustración 27.- Prueba 3 - consola controlador Ryu

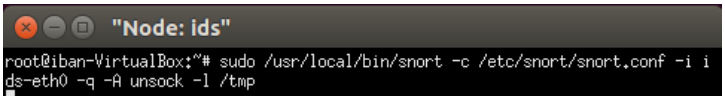

Ilustración 28.- Prueba 3 - consola IDS Snort

8.1.4 Prueba 4 – Respuesta a alarmas

En esta última prueba también se ha empleado la misma estrategia que en las anteriores, generando 3 pings entre h1 y h2. En este caso, se quiere observar cómo el controlador bloquea la IP de los equipos h1 y h2 después de detectar los paquetes ICMP intercambiados entre ellos. Para ello, se ha capturado en las interfaces s1-eth2 y s1-eth3, para exhibir como el switch recibe los paquetes por la primera, pero, en lugar de enviarlos por la segunda, los descarta. También se presenta la consola de Ryu donde se muestra la alarma originada y las medidas tomadas como respuesta a ella.

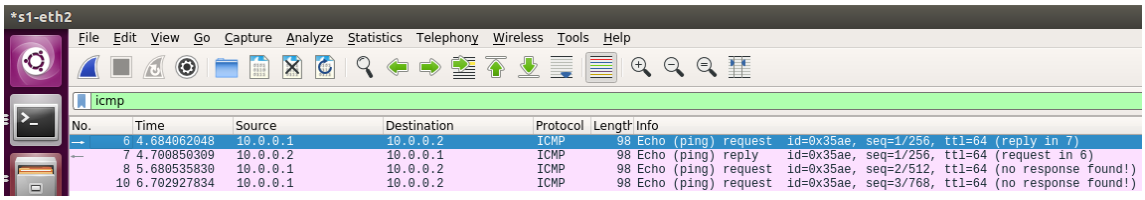

Ilustración 29.- Prueba 4 - captura s1_eth2

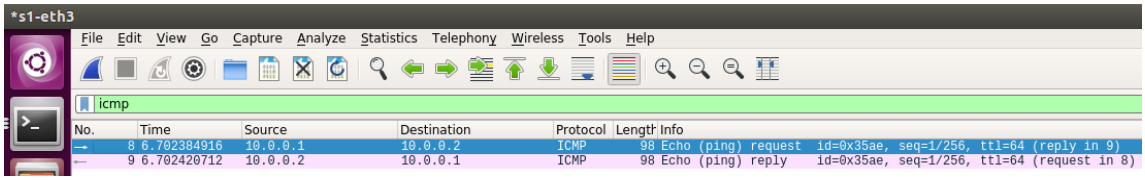

Ilustración 30.- Prueba 4 - captura s1_eth3

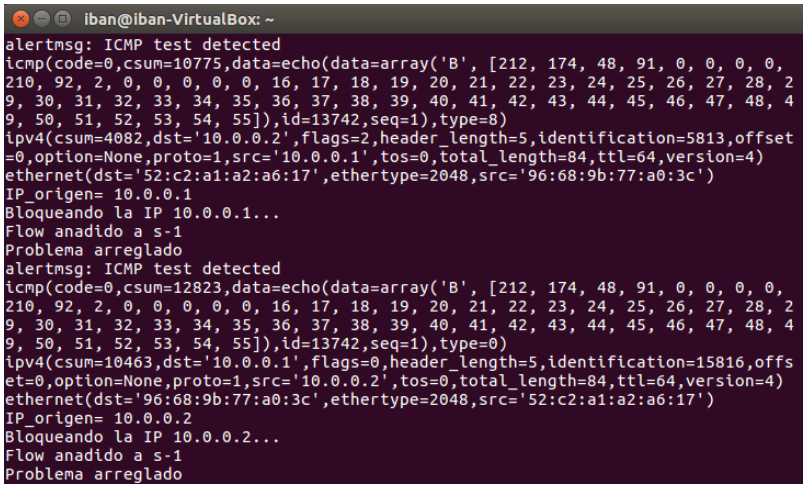

Ilustración 31.- Prueba 4 - consola controlador Ryu

8.2 Escenario 2

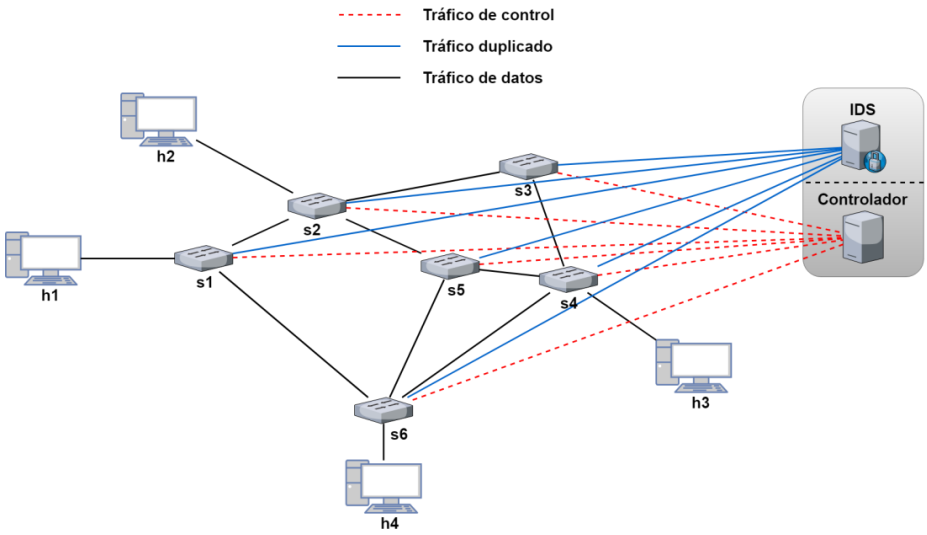

Ilustración 32.- Escenario de pruebas 2

8.2.1 Prueba 5 – Protocolo STP

Después de poner en marcha la maqueta de red y el controlador Ryu, se ha esperado cierto tiempo para que el protocolo STP llegue a un estado de convergencia. Una vez llegado a esa situación, se han observado la salida por consola del controlador y las tablas de flujo del switch s1. En la consola de Ryu se puede observar el estado final de los puertos (FORWARD para puertos designados o raíz y BLOCK para puertos no designados), mientras que en las tablas de s1 se muestran las entradas añadidas para configurar las rutas según el protocolo STP.

```
iban@iban-VirtualBox: ~
[STP][INFO] dpid=0000000000000004: [port=4] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000004: [port=5] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000006: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000006: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000006: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000006: [port=4] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000006: [port=5] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000003: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000001: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=2] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000001: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000001: [port=4] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000005: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000005: [port=3] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000005: [port=4] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000005: [port=2] NON_DESIGNATED_PORT / BLOCK
[STP][INFO] dpid=0000000000000002: [port=1] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=2] NON_DESIGNATED_PORT / BLOCK
[STP][INFO] dpid=0000000000000002: [port=3] DESIGNATED_PORT     / FORWARD
[STP][INFO] dpid=0000000000000002: [port=5] ROOT_PORT           / FORWARD
[STP][INFO] dpid=0000000000000002: [port=4] NON_DESIGNATED_PORT / BLOCK
```

Ilustración 33.- Prueba 5 - consola controlador Ryu

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=111.585s, table=0, n_packets=63, n_bytes=3780, idle_age=0,
 priority=65535,dl_dst=01:80:c2:00:00:0e actions=CONTROLLER:65535
 cookie=0x0, duration=3.114s, table=0, n_packets=8, n_bytes=672, idle_age=2, pri
ority=1,in_port=3,dl_dst=aa:67:40:31:e2:04 actions=output:2,output:1
 cookie=0x0, duration=3.112s, table=0, n_packets=1, n_bytes=98, idle_age=3, prio
rity=1,in_port=2,dl_dst=2e:57:0c:29:08:a3 actions=output:3,output:1
 cookie=0x0, duration=3.093s, table=0, n_packets=3, n_bytes=294, idle_age=2, pri
ority=1,in_port=1,dl_dst=aa:67:40:31:e2:04 actions=output:2,output:1
 cookie=0x0, duration=3.090s, table=0, n_packets=1, n_bytes=98, idle_age=2, prio
rity=1,in_port=2,dl_dst=4a:c3:86:7b:59:45 actions=output:1,output:1
 cookie=0x0, duration=3.020s, table=0, n_packets=1, n_bytes=98, idle_age=2, prio
rity=1,in_port=2,dl_dst=6e:f9:31:e1:a3:b2 actions=output:3,output:1
 cookie=0x0, duration=2.998s, table=0, n_packets=3, n_bytes=294, idle_age=2, pri
ority=1,in_port=1,dl_dst=2e:57:0c:29:08:a3 actions=output:3,output:1
 cookie=0x0, duration=2.991s, table=0, n_packets=7, n_bytes=574, idle_age=2, pri
ority=1,in_port=3,dl_dst=4a:c3:86:7b:59:45 actions=output:1,output:1
 cookie=0x0, duration=2.879s, table=0, n_packets=2, n_bytes=196, idle_age=2, pri
ority=1,in_port=1,dl_dst=6e:f9:31:e1:a3:b2 actions=output:3,output:1
```

Ilustración 34.- Prueba 5 - tablas de flujo de s1

8.2.2 Prueba 6 – Reenvío del tráfico al IDS

En esta prueba se han realizado 3 pings entre h1 y h4. En las capturas realizadas en las interfaces eth1 de los switches s1 y s6 (las interfaces conectadas al IDS) se demuestra que los paquetes recibidos se han copiado y enviado según lo esperado. Por otra parte, en las consolas de las instancias del IDS encargadas de escuchar en esas interfaces se observan las alarmas generadas, que coinciden con el tráfico enviado por cada switch.

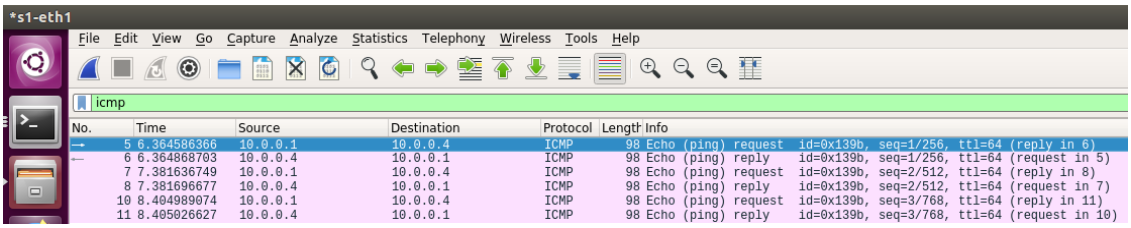

Ilustración 35.- Prueba 6 - captura s1_eth1

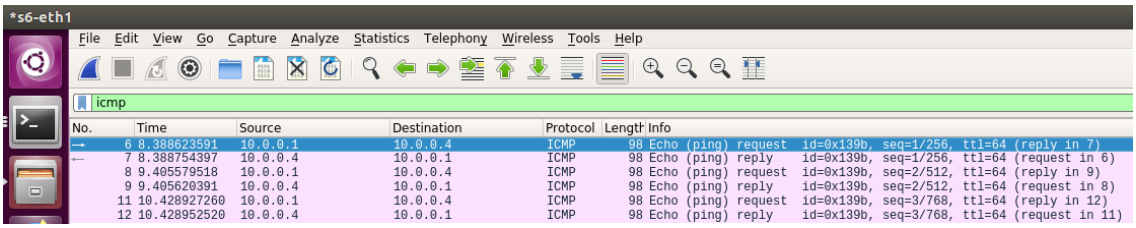

Ilustración 36.- Prueba 6 - captura s6_eth1

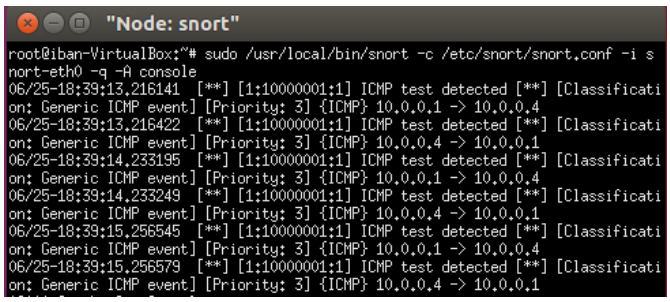

Ilustración 37.- Prueba 6 - consola IDS escuchando a s1

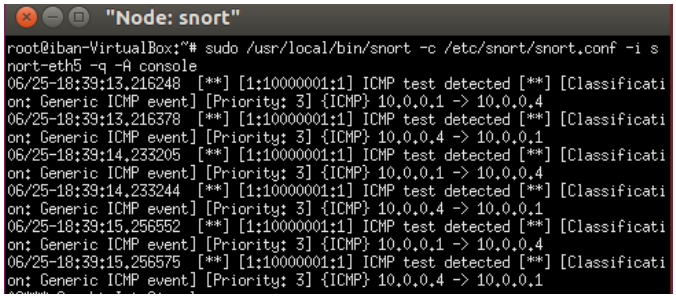

Ilustración 38.- Prueba 6 - consola IDS escuchando a s6

8.2.3 Prueba 7 – Respuesta a alarmas

Para esta prueba también se han realizado 3 pings entre h1 y h4. En este caso se utilizan las tablas de flujo del switch s1 y la consola de Ryu para demostrar que la respuesta del controlador ha sido adecuada. En la consola se muestran las acciones tomadas por el controlador en respuesta a la alarma: la extracción de la IP origen, la creación del flujo para bloquear dicha IP y la adición de dicha entrada en las tablas de los switches. En las tablas de flujo de s1 se comprueba

cómo se ha añadido el flujo cuyo objetivo es bloquear la IP: la entrada indica como condición la dirección IP en cuestión y el descarte como la acción a realizar.

```
alertmsg: ICMP test detected
icmp(code=0,csum=27672,data=echo(data=array('B', [110, 27, 49, 91, 0, 0, 0, 0, 2
0, 95, 5, 0, 0, 0, 0, 0, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
 50, 51, 52, 53, 54, 55]),id=5182,seq=1),type=8)
ipv4(csum=7058,dst='10.0.0.4',flags=2,header_length=5,identification=2835,offset
=0,option=None,proto=1,src='10.0.0.1',tos=0,total_length=84,ttl=64,version=4)
ethernet(dst='6e:f9:31:e1:a3:b2',ethertype=2048,src='aa:67:40:31:e2:04')
IP_origen= 10.0.0.1
Bloqueando la IP 10.0.0.1...
Flow anadido a s-1
Flow anadido a s-2
Flow anadido a s-3
Flow anadido a s-4
Flow anadido a s-5
Flow anadido a s-6
Problema arreglado
```

Ilustración 39.- Prueba 7 - consola controlador Ryu

```
mininet> sh ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=884.209s, table=0, n_packets=448, n_bytes=26880, idle_age=
1, priority=65535,dl_dst=01:80:c2:00:00:0e actions=CONTROLLER:65535
 cookie=0x0, duration=62.263s, table=0, n_packets=2, n_bytes=196, idle_age=60, p
riority=2,ip,nw_src=10.0.0.1 actions=drop
 cookie=0x0, duration=62.261s, table=0, n_packets=0, n_bytes=0, idle_age=62, pri
ority=2,ip,nw_src=10.0.0.4 actions=drop
 cookie=0x0, duration=775.738s, table=0, n_packets=29, n_bytes=2338, idle_age=62
, priority=1,in_port=3,dl_dst=aa:67:40:31:e2:04 actions=output:2,output:1
 cookie=0x0, duration=775.736s, table=0, n_packets=2, n_bytes=140, idle_age=770,
```

Ilustración 40.- Prueba 7 - tablas de flujo de s1

8.2.4 Prueba final – Integración

Esta prueba engloba todas las anteriores y verifica que la solución desarrollada funciona tal y como se definió en la fase de diseño. Como se trata de la prueba final donde se integra todo lo anterior, se realizará un análisis más exhaustivo de los resultados.

El punto de partida es el mismo que en la mayoría de las pruebas: se realizan 3 pings consecutivos entre los equipos h1 y h4, utilizando la consola de Mininet para ello. La salida del comando en cuestión es el siguiente:

```
mininet> h1 ping -c 3 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=0.343 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 1 received, 66% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.343/0.343/0.343/0.000 ms
```

Ilustración 41.- Prueba de integración - CLI Mininet

En esta ilustración se puede ver que el primer ping ha llegado con éxito, mientras que los dos siguientes no han llegado a su destino. De ello se pueden extraer varias conclusiones:

- Por un lado, el protocolo STP y las funciones de reenvío y aprendizaje MAC de los switches funcionan correctamente, pues en caso contrario el primer paquete ICMP nunca habría llegado a su destino.
- Por otro lado, se puede suponer que el *port mirroring* hacia el IDS, la generación de alarmas, su envío al controlador y la respuesta del mismo también han funcionado según lo esperado, considerando que el primer ping ha sido detectado y los posteriores han sido descartados. No obstante, la pérdida de paquetes en una red puede deberse a causas muy variadas, por lo que se han realizado más pruebas para demostrar que efectivamente la pérdida del paquete ICMP se debe a un bloqueo intencionado.

En primer lugar, se ha utilizado el analizador de protocolos Wireshark para capturar el tráfico en dos puntos de la red: en la interfaz s1-eth2 que conecta el equipo h1 con el switch s1 y en la interfaz s1-eth3 que conecta el switch s1 con el switch s6. Las capturas realizadas se muestran en las siguientes ilustraciones:

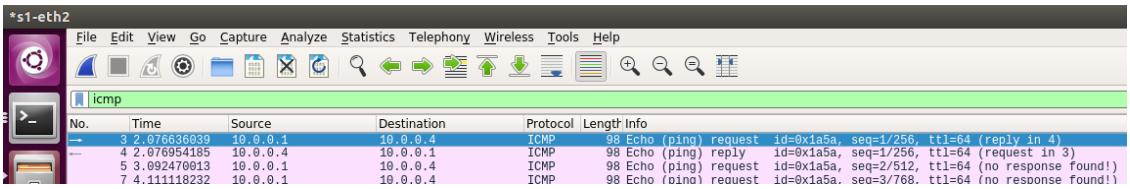

Ilustración 42.- Prueba de integración - captura s1_eth2

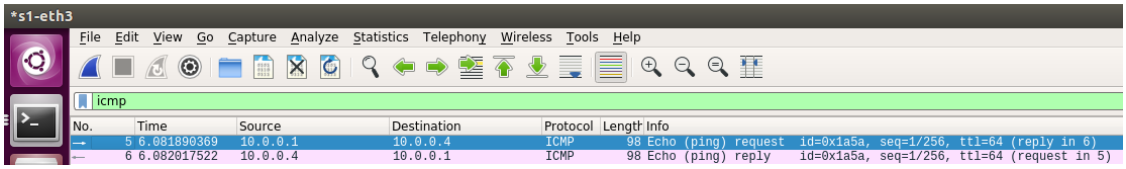

Ilustración 43.- Prueba de integración - captura s1_eth3

Estas capturas aportan más claridad a la causa por la cual se han perdido los paquetes ICMP: se observa claramente como los tres pings han llegado al switch s1, pero solo se ha reenviado hacia su destino el primero de ellos. Por tanto, se puede afirmar en un principio que el switch ha descartado intencionadamente los paquetes ICMP que se han perdido. Sin embargo, si se adopta una actitud crítica es posible pensar que los paquetes pueden haber sido descartados por otras razones: por ejemplo, que la cola de salida del switch estuviera saturada. Por tanto, se han analizado más parámetros para verificar que el comportamiento del switch ha sido programado y no se ha debido a problemas de capacidad de la red.

En esa línea, se ha estudiado la salida de la consola del controlador Ryu, que proporciona información sobre las acciones que realiza en cada momento. Observando la parte correspondiente a las alarmas del IDS y la respuesta adoptada, uno se encuentra con lo siguiente:

```
alertmsg: ICMP test detected
icmp(code=0,csum=44597,data=echo(data=array('B', [151, 40, 49, 91, 0, 0, 0, 0, 1
63, 24, 5, 0, 0, 0, 0, 0, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49
, 50, 51, 52, 53, 54, 55]),id=6746,seq=1),type=8)
ipv4(csum=4273,dst='10.0.0.4',flags=2,header_length=5,identification=5620,offset
=0,option=None,proto=1,src='10.0.0.1',tos=0,total_length=84,ttl=64,version=4)
ethernet(dst='fa:3a:43:be:ee:7a',ethertype=2048,src='7a:dd:75:33:64:a0')
IP_origen= 10.0.0.1
Bloqueando la IP 10.0.0.1...
Flow anadido a s-1
Flow anadido a s-2
Flow anadido a s-3
Flow anadido a s-4
Flow anadido a s-5
Flow anadido a s-6
Problema arreglado
alertmsg: ICMP test detected
icmp(code=0,csum=46645,data=echo(data=array('B', [151, 40, 49, 91, 0, 0, 0, 0, 1
63, 24, 5, 0, 0, 0, 0, 0, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49
, 50, 51, 52, 53, 54, 55]),id=6746,seq=1),type=0)
```

Ilustración 44.- Prueba de integración - consola del controlador Ryu

El texto resaltado se corresponde con una de las alarmas recibidas del IDS, aunque el total de alarmas atendidas asciende a cuatro. Esto se debe a que el primer ping realizado, en su camino de ida hacia h4 y vuelta hacia h1, pasa dos veces por el switch s1 y otras dos veces por el switch

s6, por lo que el IDS recibe cuatro paquetes ICMP (aunque la mitad sean duplicados), dando como resultado las cuatro alarmas mencionadas.

La estructura de la salida es la misma para todas las alarmas: información sobre el paquete que origina la alarma (separado según las capas ICMP, IP y Ethernet), extracción de la dirección IP origen, creación del flujo para bloquear dicha IP, y la adición de la entrada a las tablas de flujo de los switches. En la Ilustración 44 se muestra como la dirección IP 10.0.0.1, correspondiente al equipo h1, es bloqueada. Para la respuesta del ping se realiza el proceso inverso, y la IP 10.0.0.4, correspondiente al equipo h4, también acaba siendo bloqueada.

Por otra parte, es posible consultar las tablas de flujo de los switches para confirmar definitivamente que se ha añadido una entrada para bloquear las direcciones IP de los equipos h1 y h4. A continuación, se exhiben las tablas de flujo del switch s1:

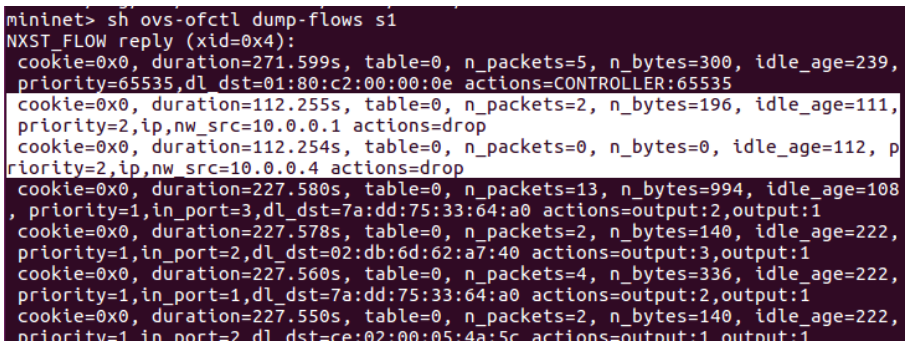

Ilustración 45.- Prueba de integración - tablas de flujo del switch s1

La parte resaltada de las tablas se corresponde a los flujos añadidos por el controlador para hacer frente a las amenazas detectadas. En ellas, se puede ver cómo el criterio para que haya una coincidencia con el paquete es la dirección IP y la acción a realizar por el switch es el descarte del paquete en cuestión. Llegado a este punto, se puede afirmar con seguridad que la solución desarrollada cumple con lo establecido en el diseño.

9 Metodología

9.1 Descripción de tareas

A continuación, se indican las etapas que se han completado para llevar a cabo el proyecto:

- **Definición del proyecto:** en esta primera fase se definen los objetivos del proyecto, así como las especificaciones y los beneficios del mismo. En esta etapa también se definen las tareas a realizar durante el proyecto.
- **Análisis de alternativas:** en esta segunda etapa se estudian las diferentes herramientas y tecnologías adecuadas para llevar a cabo el proyecto. Después de valorar las alternativas disponibles, se realiza una selección para determinar los recursos a utilizar en el trabajo.
- **Diseño de la solución:** este tercer período se dedica a diseñar los módulos o los elementos que componen la solución propuesta. Para este trabajo en concreto se han tenido que diseñar el escenario de desarrollo (la maqueta de red) y el sistema IDS.
- **Desarrollo de la solución:** esta cuarta fase consiste en desarrollar el proyecto teniendo en cuenta el trabajo previo realizado en las etapas anteriores. En este trabajo en particular esto incluye el despliegue de la maqueta de red SDN, todo lo relacionado con la puesta en marcha del IDS y la configuración y programación del controlador SDN.
- **Gestión y documentación:** este último apartado se lleva a cabo durante todo el proyecto e incluye todas las actividades relacionadas con la documentación del proyecto.

Considerando estas etapas del proyecto, se ha utilizado una representación gráfica conocida como estructura de descomposición de trabajo (en inglés *Work Breakdown Structure*, WBS) para descomponer de forma jerárquica las actividades a desarrollar durante el proyecto:

Gráfico 1.- Estructura de descomposición de trabajo (WBS)

9.2 Planificación

A continuación, se muestran en una tabla las tareas a realizar durante el proyecto, detallando la duración y las fechas de inicio y final de cada una de ellas:

Código	Nombre de tarea	Duración	Comienzo	Fin
1	**Definición del proyecto**	**7 días**	**lun 02/10/17**	**mar 10/10/17**
T.1.1	Definición de objetivos y alcance	2 días	lun 02/10/17	mar 03/10/17
T.1.2	Definición de especificaciones	2 días	mie 04/10/17	jue 05/10/17
T.1.3	Definición de tareas	3 días	vie 06/10/17	mar 10/10/17
H.1.4	Definición del proyecto terminada	0 días	mie 11/10/17	mie 11/10/17
2	**Análisis de alternativas**	**16 días**	**mie 11/10/17**	**mie 01/11/17**
T.2.1	Estudio de tecnologías disponibles	10 días	mie 11/10/17	mar 24/10/17
T.2.2	**Análisis de alternativas**	**6 días**	**mie 25/10/17**	**mie 01/11/17**
T.2.2.1	Alternativas de escenario de desarrollo	2 días	mie 25/10/17	jue 26/10/17
T.2.2.2	Alternativas de controladores SDN	2 días	vie 27/10/17	lun 30/10/17
T.2.2.3	Alternativas de IDS	2 días	mar 31/10/17	mie 01/11/17
H.2.3	Alternativas seleccionadas	0 días	jue 02/11/17	jue 02/11/17
3	**Diseño de la solución**	**20 días**	**jue 02/11/17**	**mie 06/12/17**
T.3.1	Diseño de la maqueta de red	5 días	jue 02/11/17	mie 15/11/17
T.3.2	Diseño del IDS	15 días	jue 16/11/17	mie 06/12/17
H.3.3	Diseño de la solución completo	0 días	jue 07/12/17	jue 07/12/17
4	**Desarrollo de la solución**	**85 días**	**lun 29/01/18**	**vie 25/05/18**
T.4.1	**Entorno de desarrollo**	**20 días**	**lun 29/01/18**	**vie 23/02/18**
T.4.1.1	Configuración del entorno	10 días	lun 29/01/18	vie 09/02/18
T.4.1.2	Despliegue de la maqueta de red	10 días	lun 12/02/18	vie 23/02/18
H.4.1.3	La maqueta de red funciona	0 días	lun 26/02/18	lun 26/02/18

T.4.2	**IDS**	**25 días**	**lun 26/02/18**	**vie 30/03/18**
T.4.2.1	Instalación del IDS	5 días	lun 26/02/18	vie 02/03/18
T.4.2.2	Configuración del IDS	5 días	lun 05/03/18	vie 09/03/18
T.4.2.3	Integración del IDS en la red	15 días	lun 12/03/18	vie 30/03/18
H.4.2.4	El IDS genera alertas	0 días	lun 02/04/18	lun 02/04/18
T.4.3	**Controlador SDN**	**40 días**	**lun 02/04/18**	**vie 25/05/18**
T.4.3.1	Configuración básica	5 días	lun 02/04/18	vie 06/04/18
T.4.3.2	Recepción de alarmas del IDS	15 días	lun 09/04/18	vie 27/04/18
T.4.3.3	Programación de respuestas	20 días	lun 30/04/18	vie 25/05/18
H.4.3.4	El controlador responde de forma adecuada	0 días	lun 28/05/18	lun 28/05/18
H.4.4	Solución desarrollada	0 días	lun 28/05/18	lun 28/05/18
5	**Gestión y documentación**	**55 días**	**lun 09/04/18**	**vie 22/06/18**
T.5.1	Memoria TFG	55 días	lun 09/04/18	vie 22/06/18
H.5.2	Documentación completada	0 días	lun 25/06/18	lun 25/06/18

Tabla 11.- Listado de tareas

Se puede observar en la tabla que hay un intervalo temporal sin utilizar entre diciembre y enero, correspondiente a los exámenes del primer cuatrimestre. Asimismo, se distingue entre las tareas a realizar (códigos empezados por T) y los hitos que marcan el término de cada etapa (códigos empezados por H). Es posible señalar también que se ha utilizado el primer cuatrimestre para llevar a cabo todas las tareas previas al desarrollo, mientras que el segundo cuatrimestre se ha dedicado exclusivamente al desarrollo y la documentación del proyecto.

Considerando todos estos aspectos, se ha elaborado un diagrama de Gantt, mostrado en los gráficos Gráfico 2 y Gráfico 3 de las páginas siguientes.

9.3 Diagrama de Gantt

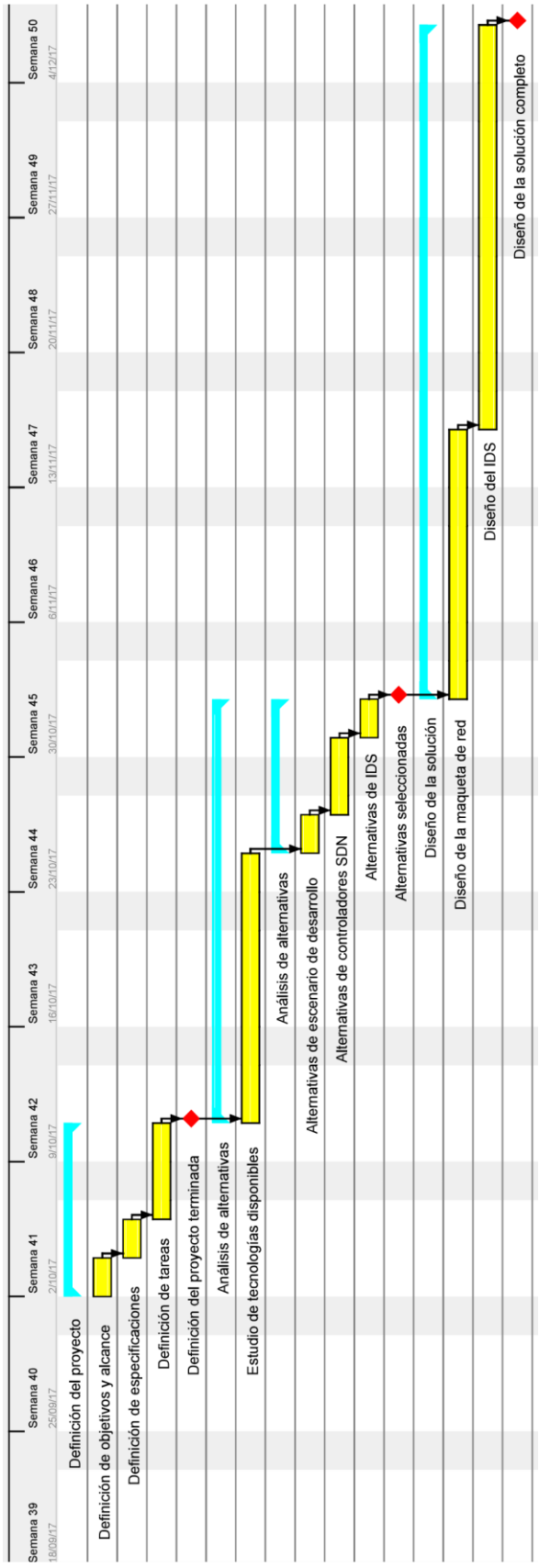

Gráfico 2.- Diagrama de Gantt (1)

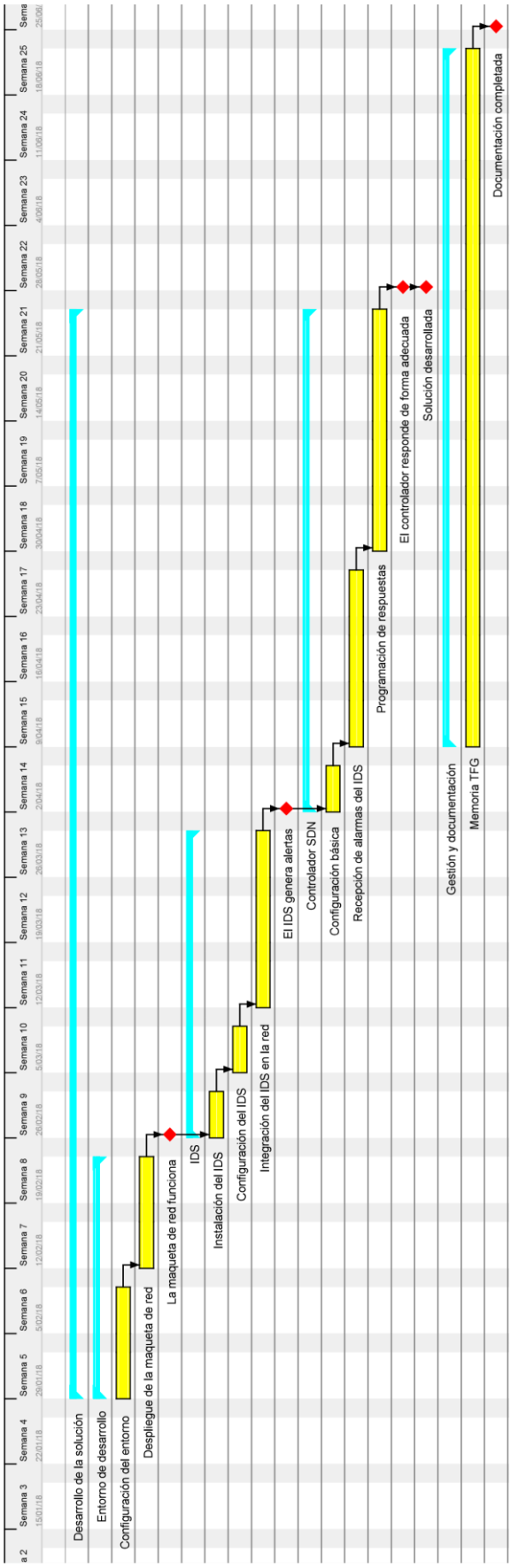

Gráfico 3.- Diagrama de Gantt (2)

10 Costes del proyecto

Es evidente que para una empresa el factor más importante a la hora de llevar a cabo un proyecto es obtener beneficios. Para conocer la inversión necesaria para este proyecto, se ha elaborado un informe sobre los costes del proyecto.

10.1 Horas internas

Concepto	Nº horas	Coste horario (€/h)	Coste total (€)
Ingeniero técnico	300	20	6 000
Ingeniero sénior	60	50	3 000
SUBTOTAL HORAS INTERNAS			9 000

Tabla 12.- Partida de horas internas

10.2 Amortizaciones

Concepto	Coste adquisición (€)	Vida útil[3]	Uso en el proyecto (h)	Coste total (€)
Ordenador	400	5 años	300	13,62
Licencia Microsoft Office	69	1 año	100	3,92
SUBTOTAL AMORTIZACIONES				17,54

Tabla 13.- Partida de amortizaciones

10.3 Gastos

Concepto	Coste (€)
Conexión a Internet	250
Material de oficina	30
SUBTOTAL GASTOS	280

Tabla 14.- Partida de gastos

[3] A la hora de calcular el uso en el proyecto frente a la vida útil se ha considerado que un año tiene 1760 horas laborables

10.4 Resumen de costes

A la hora de calcular el coste completo, se ha añadido una partida de costes indirectos e imprevistos (10 % del coste directo), con la intención de considerar gastos no atribuibles a un solo proyecto (como el agua o la electricidad), así como posibles gastos de problemas que hayan podido surgir a lo largo del proyecto (fallos técnicos en los equipos, etc.).

Por otra parte, se ha esbozado un gráfico para comparar la importancia de cada partida dentro de los costes del proyecto:

Gráfico 4.- Comparativa de partidas del coste del proyecto

Se puede observar que la mayor parte del coste se corresponde con las horas internas. Esto es de esperar teniendo en cuenta que el proyecto requiere muchas horas de trabajo correspondientes al estudio de las tecnologías y el diseño y desarrollo de la solución. Por otra parte, utilizar software libre y no tener que comprar licencias de uso hace que la partida de amortizaciones resulte en un coste despreciable frente al resto.

Concepto	Coste (€)
Horas internas	9 000
Amortizaciones	17,54
Gastos	280
SUBTOTAL	9 297,54
Costes indirectos e imprevistos (10 %)	929,75
TOTAL	10 227,29

Tabla 15.- Resumen de costes

Por tanto, el coste total del proyecto (sin incluir el IVA) asciende a 10 227,29 €.

11 Análisis de riesgos

El propósito de este apartado es identificar los riesgos que pueden afectar de manera negativa al proyecto. Asimismo, se pretende planificar una respuesta para intentar reducir la probabilidad de que ocurran y minimizar la amenaza que suponen para lograr los objetivos del proyecto.

11.1 Identificación de riesgos

A continuación, se recogen los riesgos identificados durante la fase de planificación del proyecto. Estos riesgos se clasifican como externos si tienen su origen fuera del equipo de proyecto e internos si dependen de los integrantes del proyecto.

11.1.1 Riesgos externos

En primer lugar, para el desarrollo de este Trabajo Fin de Grado se han utilizado tecnologías concretas como Snort y Ryu. Aunque en el momento de realizar el proyecto se hayan elegido como las alternativas más adecuadas, esta situación puede cambiar en un futuro. Es posible que el controlador Ryu caiga en desuso porque surja un nuevo controlador SDN o uno ya existente consiga más aceptación en el mercado. De la misma manera, Snort puede ser sustituido debido a un nuevo ataque dirigido específicamente a esta herramienta o a causa de sistemas IDS más actuales y con más funcionalidades.

Por otra parte, es muy probable que tanto Ryu como Snort sufran modificaciones en sus versiones futuras. Esto podría suponer un problema si los cambios realizados afectan de manera significativa a la arquitectura del sistema o a las funcionalidades proporcionadas. Por ejemplo, la aplicación del controlador necesitaría ser adaptada si las librerías de Ryu utilizadas para la misma experimentan cambios. Del mismo modo, se puede dar el caso de que Snort realice modificaciones en la estructura de sus reglas, por lo que sería necesario revisar la configuración del IDS.

11.1.2 Riesgos internos

En cuanto a los riesgos internos, los fallos en el equipamiento suelen ser uno de los problemas habituales. Aunque para este proyecto se ha utilizado un entorno virtual y no equipamiento real, el equipo sobre el que se ha desplegado la maqueta de red simulada puede sufrir problemas técnicos, provocando una perdida parcial o total del proyecto.

11.2 Análisis de riesgos

Para realizar un análisis cualitativo-cuantitativo de los riesgos identificados y clasificarlos según su prioridad, se han valorado dos aspectos importantes de cada riesgo: la probabilidad de que suceda y el impacto que puede tener en caso de que ocurra. A continuación, se presenta el análisis realizado:

ID	Riesgo	Causa	Probabilidad	Impacto	Prioridad
R1	**Nuevas soluciones**	Alternativas más aceptadas o mejores	Posible (0,5)	Alto (0,4)	Alta (0,2)
R2	**Incompatibilidad parcial o completa**	Modificaciones en versiones futuras	Probable (0,7)	Moderado (0,2)	Moderada (0,14)
R3	**Pérdida de información**	Problemas técnicos	Difícil (0,3)	Moderado (0,2)	Moderada (0,06)

Tabla 16.- Análisis de riesgos

Para representar estos resultados de una manera más gráfica, se ha utilizado una herramienta denominada matriz de probabilidad-impacto. Como su nombre indica, esta matriz contrasta la probabilidad y el impacto de cada riesgo para determinar su prioridad.

		IMPACTO				
		Muy bajo (0,05)	Bajo (0,1)	Moderado (0,2)	Alto (0,4)	Muy alto (0,8)
PROBABILIDAD	Raro (0,1)	Baja 0,005	Baja 0,01	Baja 0,02	Moderada 0,04	Moderada 0,08
	Difícil (0,3)	Baja 0,015	Baja 0,03	**R3** Moderada 0,06	Moderada 0,12	Alta 0,24
	Posible (0,5)	Baja 0,025	Moderada 0,05	Moderada 0,1	**R1** Alta 0,2	Alta 0,4
	Probable (0,7)	Baja 0,035	Moderada 0,07	**R2** Moderada 0,14	Alta 0,28	Alta 0,56
	Casi seguro (0,9)	Moderada 0,045	Moderada 0,09	Alta 0,18	Alta 0,36	Alta 0,72

Gráfico 5.- Matriz de probabilidad-impacto

11.3 Planificación de la respuesta

Una vez identificados y evaluados los riesgos, es necesario planificar una respuesta para intentar reducir la probabilidad de que sucedan y minimizar su impacto sobre el proyecto.

El primer riesgo (R1) es el más complicado de afrontar, puesto que es imposible conocer el estado del mercado en un futuro. Por tanto, no queda más remedio que aceptar el riesgo y planificar una respuesta para hacerle frente en caso de que suceda. Para ello, se ha realizado un diseño de la solución lo más independiente posible respecto a las tecnologías utilizadas. De esta manera, aunque sea necesario rehacer parte del desarrollo, el diseño sigue siendo válido para otras tecnologías que se puedan emplear en el futuro. Por otra parte, se llevará un seguimiento de las tecnologías más relevantes en cada momento para poder anticiparse a una situación de este tipo.

Respecto al segundo riesgo (R2), también se ha decidido asumirlo. Es bastante probable que las versiones futuras de Ryu o Snort sean susceptibles a cambios y por tanto sea necesario adaptar la solución desarrollada. Con la intención de minimizar el impacto, se ha buscado realizar un diseño modular de la solución, de forma que en caso de ocurrir el riesgo la mayor parte sea reutilizable y haya que modificar lo menos posible. Otra opción interesante puede ser el de transferir el riesgo: si se consigue que un grupo de desarrolladores se interesen en darle continuidad al proyecto y mantenerlo, se garantizaría la compatibilidad de la solución con versiones futuras de las tecnologías empleadas.

En lo que se refiere al riesgo R3, se ha buscado minimizar dicho riesgo. Por un lado, se ha reducido la probabilidad de que ocurra el riesgo llevando un mantenimiento adecuado del equipo utilizado para simular la maqueta de red SDN. También se han mejorado las especificaciones del equipo añadiendo algunos componentes adicionales para evitar una situación de sobrecarga. Por otro lado, para evitar que la pérdida de información sea devastadora en caso de suceder el riesgo, se ha guardado en todo momento una copia de seguridad tanto en la nube como en un dispositivo de almacenamiento externo.

12 Conclusiones

La conclusión más significativa de este Trabajo Fin de Grado es que se han alcanzado los objetivos planteados al inicio del proyecto, en su totalidad y de forma satisfactoria.

Las Redes Definidas por Software ofrecen características superiores en muchos aspectos a las redes tradicionales, lo que permite satisfacer los requisitos de las aplicaciones tecnológicas actuales. No obstante, el aumento de los ataques cibernéticos en los últimos años y la lógica centralizada de las redes SDN hace que éstas sean vulnerables. Los Sistemas de Detección de Intrusión permiten convertir esa debilidad en virtud, aprovechando la centralización para tener un acceso más sencillo al tráfico de la red y poder detectar de forma más efectiva las actividades maliciosas en el sistema.

Este Trabajo Fin de Grado detalla el proceso de diseño e implementación de un sistema de seguridad de estas características. Mediante este proyecto se demuestra la viabilidad de los IDS como herramienta de seguridad en redes SDN, además de mostrar las ventajas que supone dicha solución.

Por otra parte, el estudio propone una nueva forma de utilizar el IDS gracias a las capacidades que ofrecen las redes SDN. En la solución diseñada se plantea programar el controlador SDN para que, además de realizar las funciones de control de la red SDN, pueda tomar medidas frente a las amenazas detectadas para intentar solventarlas. De esta manera, se evita en cierta medida la necesidad de intervención humana que presentan los IDS tradicionales. Asimismo, las respuestas previamente programadas permiten contrarrestar los ataques identificados inmediatamente, pues el controlador interviene en cuanto el IDS genera la alarma.

En resumen, este Trabajo Fin de Grado aborda uno de los problemas pendientes de las redes SDN, la seguridad, siendo ésta además una de las preocupaciones más actuales en el sector de las telecomunicaciones. Con este objetivo, presenta una solución capaz de integrar un sistema de seguridad utilizado en redes tradicionales como es un IDS en una red SDN de nueva generación. Por otro lado, se abre una nueva vía de investigación al proponer un controlador SDN capaz de responder a las amenazas detectadas de manera automática.

13 Bibliografía

[1] L. Hardesty, «Google brings SDN to the Public Internet - SDxCentral,» [En línea]. Available: https://www.sdxcentral.com/articles/news/google-brings-sdn-public-internet/2017/04/. [Último acceso: 13 Junio 2018].

[2] Open Networking Foundation, «Software-Defined Networking (SDN) Definition,» [En línea]. Available: https://www.opennetworking.org/sdn-definition/. [Último acceso: 13 Junio 2018].

[3] Open Networking Foundation, «OpenFlow Switch Specification,» [En línea]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.1.pdf. [Último acceso: 13 Junio 2018].

[4] A. Lazarevic, V. Kumar y J. Srivastava, «Intrusion detection: A survey,» de *Managing Cyber Threats*, Springer, 2005, pp. 19-78.

[5] J. Ibrahim y S. Gajin, «SDN-Based Intrusion Detection System,» de *Infoteh*, Jahorina, 2017.

[6] «Mininet: An Instant Virtual Network on your Laptop,» [En línea]. Available: http://mininet.org/. [Último acceso: 13 Junio 2018].

[7] «mininet/mininet: Emulator for rapid prototyping of Software Defined Networks,» [En línea]. Available: https://github.com/mininet/mininet. [Último acceso: 13 Junio 2018].

[8] B. Linkletter, «How to use MiniEdit, Mininet's graphical user interface | Open-Source Routing and Network Simulation,» [En línea]. Available: http://www.brianlinkletter.com/how-to-use-miniedit-mininets-graphical-user-interface/. [Último acceso: 13 Junio 2018].

[9] «Snort - Network Intrusion Detection & Prevention System,» [En línea]. Available: https://www.snort.org/. [Último acceso: 13 Junio 2018].

[10] N. Dietrich, «Snort 2.9.9.x on Ubuntu 14 and 16,» [En línea]. Available: https://www.snort.org/documents/snort-2-9-9-x-on-ubuntu-14-16. [Último acceso: 13 Junio 2018].

[11] The Snort Project, «Snort Users Manual,» [En línea]. Available: https://www.snort.org/documents/snort-users-manual. [Último acceso: 13 Junio 2018].

[12] «Welcome to RYU the Network Operating System(NOS),» [En línea]. Available: http://ryu.readthedocs.io/en/latest/index.html. [Último acceso: 13 Junio 2018].

[13] «Writing Your Ryu Application - Ryu 4.25 documentation,» [En línea]. Available: http://ryu.readthedocs.io/en/latest/developing.html. [Último acceso: 13 Junio 2018].

[14] «Spanning Tree - Ryubook 1.0 documentation,» [En línea]. Available: https://osrg.github.io/ryu-book/en/html/spanning_tree.html. [Último acceso: 13 Junio 2018].

[15] «Snort Integration - Ryu 4.25 documentation,» [En línea]. Available: http://ryu.readthedocs.io/en/latest/snort_integrate.html. [Último acceso: 13 Junio 2018].

Anexo I: Normativa aplicable

Para el desarrollo del proyecto no ha sido necesario cumplir ninguna normativa, pero las contribuciones del código desarrollado sí deben cumplir con ciertas licencias de distribución aplicadas al software de código libre.

Licencia BSD

La licencia BSD es una licencia de software libre permisiva, que impone restricciones mínimas al uso y distribución al software cubierto por esta licencia. Al contrario que las licencias *copyleft*, no tiene requerimientos de igual compartición, es decir, no es necesario que las copias o adaptaciones del código se distribuyan bajo la misma licencia que el código original.

El autor, bajo la licencia BSD, mantiene la protección de los derechos de autor únicamente para la renuncia de garantía y para requerir la adecuada atribución de la autoría en trabajos derivados. No obstante, como ya se ha mencionado se permite la libre distribución y modificación sin la obligación de mantener el tipo de licencia. Esto incluye la posibilidad de que el software modificado pueda convertirse en software privativo y se redistribuya como no libre, a diferencia de otras licencias de software libre como GPL.

El entorno de simulación SDN utilizado en este proyecto, Mininet, se distribuye precisamente bajo la licencia BSD.

GNU General Public License

La Licencia Pública General de GNU (más conocida por su nombre en inglés GNU General Public License o, directamente, GNU GPL) es una licencia utilizada ampliamente en la comunidad de software libre y código abierto. Esta licencia garantiza a los usuarios finales la libertad de usar, modificar y distribuir el software. La licencia tiene un doble propósito: por un lado, declarar que el software cubierto por esta licencia es libre y, por otro lado, proteger el software de intentos de apropiación mediante la práctica conocida como *copyleft*.

La GPL hace hincapié en la práctica del *copyleft*, lo que implica que solo se permite que modificaciones de un software bajo la GPL se distribuyan bajo la misma licencia. Por tanto, se da a los autores la seguridad de que el software creado a partir del original siempre será libre y no podrá ser explotado de manera comercial.

El software cubierto por esta licencia utilizado en el proyecto ha sido Snort, que se distribuye con una licencia GNU GPL versión 2.

Apache License

La licencia Apache, al igual que la licencia BSD, es una licencia de software libre permisiva. La licencia Apache requiere la conservación del aviso de derecho de autor y el descargo de responsabilidad, pero no es una licencia *copyleft*, por lo que es posible distribuir modificaciones del software sin mantener la licencia original.

No obstante, es necesario que se aplique la misma licencia a todas las partes no modificadas del software, así como los derechos de autor u cualquier otra atribución al autor original del mismo. La licencia Apache solo exige que se mantenga un aviso para informar al usuario que en la

distribución se ha utilizado código bajo esa licencia. En caso de cambiar la licencia original, se debe añadir una notificación para indicar que se han realizado cambios al software en cuestión.

El controlador Ryu empleado para el proyecto se ha distribuido bajo la licencia Apache 2.0.

Anexo II: Código desarrollado

Código de la maqueta de red de Mininet

```
01: #!/usr/bin/python
02:
03: # Importar librerías
04: from mininet.net import Mininet
05: from mininet.node import Controller, RemoteController, OVSController
06: from mininet.node import CPULimitedHost, Host, Node
07: from mininet.node import OVSKernelSwitch, UserSwitch
08: from mininet.node import IVSSwitch
09: from mininet.cli import CLI
10: from mininet.log import setLogLevel, info
11: from mininet.link import TCLink, Intf
12: from subprocess import call
13:
14: # Definir la topología
15: def myNetwork():
16:
17:     net = Mininet( topo=None,
18:                    build=False,
19:                    ipBase='10.0.0.0/8')
20:
21: # Añadir controlador
22:     info( '*** Adding controller\n' )
23:     c0=net.addController(name='c0',
24:                          controller=RemoteController,
25:                          ip='127.0.0.1',
26:                          protocol='tcp',
27:                          port=6633)
28:
29: # Añadir switches
30:     info( '*** Add switches\n')
31:     s6 = net.addSwitch('s6', cls=OVSKernelSwitch)
32:     s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
33:     s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
34:     s5 = net.addSwitch('s5', cls=OVSKernelSwitch)
35:     s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
36:     s4 = net.addSwitch('s4', cls=OVSKernelSwitch)
37:
38: # Añadir hosts
39:     info( '*** Add hosts\n')
40:     h1 = net.addHost('h1', cls=Host, ip='10.0.0.1', defaultRoute=None)
41:     h3 = net.addHost('h3', cls=Host, ip='10.0.0.3', defaultRoute=None)
42:     snort = net.addHost('snort', cls=Host, ip='10.0.0.5',
defaultRoute=None)
43:     h2 = net.addHost('h2', cls=Host, ip='10.0.0.2', defaultRoute=None)
44:     h4 = net.addHost('h4', cls=Host, ip='10.0.0.4', defaultRoute=None)
45:
46: # Añadir enlaces
47:     info( '*** Add links\n')
48:     net.addLink(s1, snort)
49:     net.addLink(s2, snort)
50:     net.addLink(s3, snort)
51:     net.addLink(s4, snort)
52:     net.addLink(s5, snort)
```

```
53:     net.addLink(s6, snort)
54:     net.addLink(s1, h1)
55:     net.addLink(s1, s6)
56:     net.addLink(s1, s2)
57:     net.addLink(s2, h2)
58:     net.addLink(s2, s3)
59:     net.addLink(s3, s4)
60:     net.addLink(s6, s4)
61:     net.addLink(s5, s6)
62:     net.addLink(s5, s4)
63:     net.addLink(s2, s5)
64:     net.addLink(s6, h4)
65:     net.addLink(s4, h3)
66:
67: # Desplegar la red
68:     info( '*** Starting network\n')
69:     net.build()
70:     info( '*** Starting controllers\n')
71:     for controller in net.controllers:
72:         controller.start()
73:
74:     info( '*** Starting switches\n')
75:     net.get('s6').start([c0])
76:     net.get('s3').start([c0])
77:     net.get('s1').start([c0])
78:     net.get('s5').start([c0])
79:     net.get('s2').start([c0])
80:     net.get('s4').start([c0])
81:
82:     info( '*** Post configure switches and hosts\n')
83:
84: # Mostrar CLI después de desplegar la red
85:     CLI(net)
86:     net.stop()
87:
88: # Función principal
89: if __name__ == '__main__':
90:     setLogLevel( 'info' )
91:     myNetwork()
```

Código del controlador Ryu

```
001: # Copyright (C) 2016 Nippon Telegraph and Telephone Corporation.
002: #
003: # Licensed under the Apache License, Version 2.0 (the "License");
004: # you may not use this file except in compliance with the License.
005: # You may obtain a copy of the License at
006: #
007: #    http://www.apache.org/licenses/LICENSE-2.0
008: #
009: # Unless required by applicable law or agreed to in writing, software
010: # distributed under the License is distributed on an "AS IS" BASIS,
011: # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
012: # implied.
013: # See the License for the specific language governing permissions and
014: # limitations under the License.
015:
016: ### IMPORTAR LIBRERÍAS ###
017: from __future__ import print_function
018: import array
019:
020: from ryu.base import app_manager
021: from ryu.controller import ofp_event
022: from ryu.controller import dpset
023: from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
024: from ryu.controller.handler import set_ev_cls
025: from ryu.ofproto import ofproto_v1_3
026: from ryu.lib import dpid as dpid_lib
027: from ryu.lib import stplib
028: from ryu.lib.packet import packet
029: from ryu.lib.packet import ethernet
030: from ryu.lib.packet import ipv4
031: from ryu.lib.packet import icmp
032: from ryu.app import simple_switch_13
033: from ryu.lib import snortlib
034:
035: ### FUNCIONES BÁSICAS ###
036:
037: ### Inicialización de la aplicación ###
038:
039: # Definir el controlador
040: class switchStpSnortFix(simple_switch_13.SimpleSwitch13):
041:     OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
042:     _CONTEXTS = {'stplib': stplib.Stp, 'snortlib': snortlib.SnortLib,
'dpset': dpset.DPSet,}
043:
044: # Definir variables
045:     def __init__(self, *args, **kwargs):
046:         super(switchStpSnortFix, self).__init__(*args, **kwargs)
047:         self.mac_to_port = {}
048:         self.stp = kwargs['stplib']
049:         self.snort = kwargs['snortlib']
050:         self.snort_port = 1
051:         self.dpset = kwargs['dpset']
052:
053:         socket_config = {'unixsock': True}
054:
055:         self.snort.set_config(socket_config)
```

```
056:         self.snort.start_socket_server()
057:
058:         config = {dpid_lib.str_to_dpid('0000000000000001'):
059:                   {'bridge': {'priority': 0x8000}},
060:                   dpid_lib.str_to_dpid('0000000000000002'):
061:                   {'bridge': {'priority': 0x9000}},
062:                   dpid_lib.str_to_dpid('0000000000000003'):
063:                   {'bridge': {'priority': 0xa000}}}
064:         self.stp.set_config(config)
065:
066: ### Inicialización de los switches ###
067:
068: # Evento configuración nuevo switch
069:     @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
070:     def switch_features_handler(self, ev):
071:         datapath = ev.msg.datapath
072:         ofproto = datapath.ofproto
073:         parser = datapath.ofproto_parser
074:
075:         # Instalar entrada table-miss en el switch
076:         match = parser.OFPMatch()
077:         actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
ofproto.OFPCML_NO_BUFFER)]
078:         self.add_flow(datapath, 0, match, actions)
079:
080: ### Gestión de paquetes entrantes ###
081:
082: # Evento paquete entrante al controlador
083:     @set_ev_cls(stplib.EventPacketIn, MAIN_DISPATCHER)
084:     def _packet_in_handler(self, ev):
085:         msg = ev.msg
086:         datapath = msg.datapath
087:         ofproto = datapath.ofproto
088:         parser = datapath.ofproto_parser
089:         in_port = msg.match['in_port']
090:
091:         pkt = packet.Packet(msg.data)
092:         eth = pkt.get_protocols(ethernet.ethernet)[0]
093:         dst = eth.dst
094:         src = eth.src
095:
096:         dpid = datapath.id
097:         self.mac_to_port.setdefault(dpid, {})
098:
099:         # Aprender la dirección MAC para evitar inundación
100:         self.mac_to_port[dpid][src] = in_port
101:
102:         if dst in self.mac_to_port[dpid]:
103:             out_port = self.mac_to_port[dpid][dst]
104:         else:
105:             out_port = ofproto.OFPP_FLOOD
106:
107:         actions = [parser.OFPActionOutput(out_port),
parser.OFPActionOutput(self.snort_port)]
108:
109:         # Instalar flow para evitar paquete entrante
110:         if out_port != ofproto.OFPP_FLOOD:
111:             match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
```

```
112:             self.add_flow(datapath, 1, match, actions)
113:
114:         data = None
115:         if msg.buffer_id == ofproto.OFP_NO_BUFFER:
116:             data = msg.data
117:
118:         out = parser.OFPPacketOut(datapath=datapath,
buffer_id=msg.buffer_id, in_port=in_port, actions=actions, data=data)
119:         datapath.send_msg(out)
120:
121: ### PROTOCOLO STP ###
122:
123: # Evento cambio de topología
124:     @set_ev_cls(stplib.EventTopologyChange, MAIN_DISPATCHER)
125:     def _topology_change_handler(self, ev):
126:         dp = ev.dp
127:         dpid_str = dpid_lib.dpid_to_str(dp.id)
128:         msg = 'Receive topology change event. Flush MAC table.'
129:         self.logger.debug("[dpid=%s] %s", dpid_str, msg)
130:
131:         if dp.id in self.mac_to_port:
132:             self.delete_flow(dp)
133:             del self.mac_to_port[dp.id]
134:
135: # Evento cambio de estado de puerto
136:     @set_ev_cls(stplib.EventPortStateChange, MAIN_DISPATCHER)
137:     def _port_state_change_handler(self, ev):
138:         dpid_str = dpid_lib.dpid_to_str(ev.dp.id)
139:         of_state = {stplib.PORT_STATE_DISABLE: 'DISABLE',
140:                     stplib.PORT_STATE_BLOCK: 'BLOCK',
141:                     stplib.PORT_STATE_LISTEN: 'LISTEN',
142:                     stplib.PORT_STATE_LEARN: 'LEARN',
143:                     stplib.PORT_STATE_FORWARD: 'FORWARD'}
144:         self.logger.debug("[dpid=%s][port=%d] state=%s",
145:                           dpid_str, ev.port_no, of_state[ev.port_state])
146:
147: ### RECEPCIÓN DE ALARMAS ###
148:
149: # Evento alerta de Snort
150:     @set_ev_cls(snortlib.EventAlert, MAIN_DISPATCHER)
151:     def _dump_alert(self, ev):
152:         msg = ev.msg
153:         pkt = packet.Packet(array.array('B', msg.pkt))
154:         _ipv4 = pkt.get_protocol(ipv4.ipv4)
155:
156:         if _ipv4:
157:               print('alertmsg: %s' % ''.join(msg.alertmsg))
158:               self.packet_print(msg.pkt)
159:
160:             # Llamada a respuesta frente a alarma
161:               self.fix_alert(ev)
162:
163: ### PROGRAMACIÓN DE RESPUESTAS ###
164:
165: # Programar respuesta frente a alerta del Snort
166:     def fix_alert(self, ev):
167:         msg = ev.msg
168:         pkt = msg.pkt
```

```
169:         pkt = packet.Packet(array.array('B', pkt))
170:         _ipv4 = str(pkt.get_protocol(ipv4.ipv4))
171:
172:         srcIP = _ipv4.split("'")[3]
173:
174:         print('IP_origen= %s' %srcIP)
175:         print('Bloqueando la IP %s...' %srcIP)
176:
177:         dp_set = self.dpset.get_all()
178:
179:         i = 0
180:         for dp in dp_set:
181:               i += 1
182:               datapath = dp[1]
183:
184:               parser = datapath.ofproto_parser
185:               match = parser.OFPMatch(eth_type=0x800, ipv4_src=srcIP)
186:               actions = []
187:               self.add_flow(datapath, 2, match, actions)
188:
189:               print('Flow anadido a s-%d' %i)
190:
191:         print('Problema arreglado')
192:
193: ### FUNCIONES AUXILIARES ###
194:
195: # Función auxiliar para añadir flow
196:     def add_flow(self, datapath, priority, match, actions):
197:         ofproto = datapath.ofproto
198:         parser = datapath.ofproto_parser
199:
200:         inst =
[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
201:
202:         mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
match=match, instructions=inst)
203:         datapath.send_msg(mod)
204:
205: # Función auxiliar para eliminar flow
206:     def delete_flow(self, datapath):
207:         ofproto = datapath.ofproto
208:         parser = datapath.ofproto_parser
209:
210:         for dst in self.mac_to_port[datapath.id].keys():
211:             match = parser.OFPMatch(eth_dst=dst)
212:             mod = parser.OFPFlowMod(datapath,
command=ofproto.OFPFC_DELETE, out_port=ofproto.OFPP_ANY,
out_group=ofproto.OFPG_ANY, priority=1, match=match)
213:             datapath.send_msg(mod)
214:
215: # Función auxiliar para imprimir paquete en pantalla
216:     def packet_print(self, pkt):
217:         pkt = packet.Packet(array.array('B', pkt))
218:         eth = pkt.get_protocol(ethernet.ethernet)
219:         _ipv4 = pkt.get_protocol(ipv4.ipv4)
220:         _icmp = pkt.get_protocol(icmp.icmp)
221:
222:         if _icmp:
```

```
223:             self.logger.info("%r", _icmp)
224:         if _ipv4:
225:             self.logger.info("%r", _ipv4)
226:         if eth:
227:             self.logger.info("%r", eth)
```