



# Amenazas de seguridad en redes SDN

---

Universidad Nacional Española a Distancia (UNED)

Facultad de Informática

Proyecto de Fin de Máster



Autor: Luis Ildefonso Gómez Solana

Tutor: Miguel Romero Hortelano

Co-Tutor: Elio San Cristobal Cruz

Fecha: Septiembre 2021



*Dedicado a mi pareja Marketa, mi hija Karolina y mi hijo pequeño  
Daniel, ¡Gracias por vuestro apoyo!*

*El hogar está detrás, el mundo por delante  
Y hay muchos caminos por andar  
A través de la sombra, al filo de la noche  
Hasta las estrellas han bajado  
Niebla y sombra  
Nube y sombra  
Todo se desvanecerá  
Todo se desvanecerá*



# Índice general

<b>1. INTRODUCCIÓN</b>	<b>1</b>
<b>2. FACTORES DEL PROYECTO</b>	<b>3</b>
2.1. Objetivos . . . . .	3
2.2. Motivación . . . . .	3
2.3. Estructura del documento . . . . .	4
<b>3. ESTADO DEL ARTE</b>	<b>5</b>
3.1. Internet . . . . .	5
3.1.1. Pila de protocolo de Internet . . . . .	7
3.2. Computación en la Nube . . . . .	8
3.2.1. Modelo de servicio . . . . .	9
3.2.2. Tipos de despliegues . . . . .	11
3.3. Protocolos de red . . . . .	13
3.3.1. Spanning Tree Protocol . . . . .	14
3.3.2. Protocolo Simple de Administración de Red . . . . .	14
3.3.3. Seguridad en la capa de Transporte . . . . .	15
3.3.4. Protocolo de resolución de direcciones . . . . .	16
3.4. Software-Defined Networking . . . . .	19
3.4.1. Protocolos . . . . .	20
3.4.2. Controladora SDN . . . . .	23
3.4.3. Mininet . . . . .	25
3.4.4. RestAPI . . . . .	26
3.5. Python . . . . .	27
3.6. Diseño de dispositivos de red . . . . .	28
3.6.1. Sistemas operativo red Aruba OS . . . . .	28
3.7. Centro de procesamiento de datos . . . . .	29
3.7.1. Network Functions Virtualization . . . . .	30
<b>4. Amenazas de seguridad en redes SDN</b>	<b>31</b>
4.1. ARP Spoofing en SDN . . . . .	31
4.1.1. Escenario ARP Spoofing para SDN . . . . .	32
4.1.2. Mitigar ARP Spoofing para SDN . . . . .	34
4.1.3. Escenario Proxy ARP Spoofing para SDN . . . . .	37
4.1.4. Mitigar Proxy ARP Spoofing para SDN . . . . .	40
4.2. Denial of Service - DoS . . . . .	42

4.2.1.	Escenario DoS para SDN . . . . .	42
4.2.2.	Mitigar DoS para SDN . . . . .	45
4.3.	Entorno de ejecución . . . . .	46
4.3.1.	Controladora . . . . .	46
4.3.2.	Lenguaje de programación . . . . .	47
4.3.3.	Construcción del entorno . . . . .	48
4.3.4.	Entorno de desarrollo . . . . .	48
4.3.5.	Entorno de ejecución . . . . .	49
4.3.6.	Despliegue de la aplicación . . . . .	49
4.3.7.	Control de versiones . . . . .	49
4.3.8.	Código fuente . . . . .	50
<b>5.</b>	<b>CONCLUSIONES</b>	<b>51</b>
5.1.	Planes futuros para SDN . . . . .	51
5.2.	SDN y seguridad . . . . .	52
5.3.	Conclusiones . . . . .	53

# Capítulo 1

## INTRODUCCIÓN

El presente documento ahonda en el análisis y estudio del modelo arquitectónico para las redes de computadores conocido como *SDN* (del inglés: “*Software Defined Networking*”)

En concreto, analizará y estudiará cómo este nuevo modelo afronta los ataques de seguridad más comunes en redes de computadores.

Esta nueva tecnología, enfocada en la mejora de la Computación en la Nube y más en concreto en la optimización de la Infraestructura como Servicio, de sus siglas en inglés *IaaS*, busca agilizar la administración, gestión, automatización de las redes y ofrecer mayor flexibilidad, escalabilidad y una capa de abstracción adicional a los usuarios y clientes.

El proyecto busca crear un entorno de estudio donde se pueda estudiar la respuesta de este modelo a las amenazas de seguridad que las redes actuales deben combatir y estudiar la respuesta que dicho modelo propone para ello.





# Capítulo 2

## FACTORES DEL PROYECTO

### 2.1. Objetivos

El presente documento tiene como objetivo estudiar las amenazas de seguridad en redes de computadores que usan la arquitectura *SDN* (del inglés: “*Software Defined Networking*”) tienen que afrontar.

El proyecto investigará si estas nuevas redes solucionan este tipo de amenazas y de ser así cómo lo consiguen. Se realizará una comparación con el modelo actual y se diseñará las pruebas necesarias para poder estudiar dicho comportamiento en un ambiente controlado y desde un punto de vista práctico.

Dentro de las posibles amenazas de seguridad que se estudiaran podemos citar ARP Spoofing [1] [2] y DoS [1] [2].

Es por tanto que el proyecto intentará dar a conocer las posibles mejoras de dicha nueva arquitectura contra estas amenazas y creará un entorno para su investigación.

### 2.2. Motivación

Han pasado muchos años desde que Internet se creará y se globalizara. Durante este nacimiento hemos asistido a diferentes tipos de arquitecturas de redes que se han ido sucediendo y ampliando para manejar la infraestructura sobre la que los centros de datos y por tanto la misma Internet se sustenta.

Un ejemplo llamativo serían las Redes Activas [1] [2] [3] que plantearon ya en los 90 controlar la red a través de una interfaz de programación (API). Este modelo que se extinguió debido a la carente fuerza computacional de por aquel entonces. Otros ejemplos relevantes fueron *ANTS* [4] o *SwitchWare* [5].

Hoy en día y con un Internet que no para de crecer [6] se están creando cada vez más redes y centros de datos. Todos ellos con una mayor cantidad y complejidad en cuanto a infraestructura se refiere. La gestión y seguridad de las redes a gran escala es un reto

si queremos seguir ofreciendo flexibilidad, fiabilidad y mayor control a cada uno de los clientes y dispositivos conectados a la misma. Esto se traduce en que la gestión de los centros de datos se hace especialmente tediosa y sensible para las empresas y por tanto para los administradores.

Esta mayor agilidad, flexibilidad y escalabilidad de las redes se ha convertido en unas de las principales razones para traer *nuevos* conceptos de redes o renovar ideas . Es ahora que disponemos de una mayor capacidad computacional y protocolos, cuando podemos llevarlo a cabo. De ahí que desde diversos centros de investigación, tanto públicos como privados, y empresas hayan desarrollado modelos de gestión de redes con nuevos protocolos y técnicas capaces de gestionarlas y administrarlas.

Sin embargo, puesto que el acceso y uso de Internet crece junto con la infraestructura que la mantiene y que ésta está dispersándose por el varias regiones para ofrecer disponibilidad, nos estamos exponiendo a mayor número de ataques [7]. Los ataques a los que nuestras redes se exponen tienen que ser resueltos con agilidad y rapidez evitando caídas de servicio en la medida de lo posible y este documento tratara de investigar si SDN consigue tales objetivos, estudiando casos de seguridad comunes en redes de computadores.

En este documento se ha querido ahondar en el estudio de esta nueva arquitectura y ahondar en la seguridad que la nueva arquitectura SDN ofrece.

## 2.3. Estructura del documento

El presente documento se divide en los siguientes capítulos y de la siguiente forma:

- Capítulo 1: Introducción. Contextualizar el proyecto en el escenario actual. Explicación del problema actual y la solución que plantea *SDN* como método de mejora para la *Infraestructura como Servicio* y la seguridad extra que ofrece.
- Capítulo 2: Factores del proyecto. Objetivos del proyecto con la motivación que ha llevado a ser realizado.
- Capítulo 3: Estado del Arte. Introducción y repaso de los conceptos más importantes y necesarios para entender este proyecto, los protocolos usados en la configuración y gestión de las redes como *OpenFlow*, *ARP* o *ARP Spoofing*.
- Capítulo 4: Amenazas de seguridad en redes *SDN*. Visión detallada de esta nueva arquitectura en relación con los ataques mencionados. Se revisarán los métodos para llevarlos a cabo y las posibles soluciones.
- Conclusión. Presentación de los objetivos finales obtenidos. Se añade una breve explicación de la situación actual del modelo y futuros pasos.

# Capítulo 3

## ESTADO DEL ARTE

El reto al que las empresas se han enfrentado en la última década ha sido la computación en la nube. Si hablamos desde la seguridad y según Gartner [8], las empresas que han movido su infraestructura a entornos virtuales han mejorado este apartado en al menos un 33 % que en modelos tradicionales.

Este mismo documento augura que para 2022, el 95 % de los fallos de seguridad en la nube serán fallos del cliente.

Es por tanto lógico pensar que las redes también migren a un modelo similar en donde obtengamos similares beneficios a los mencionados anteriormente. Aunque esto suponga que nos expongamos a nuevos retos de seguridad.

### 3.1. Internet

La historia de *Internet* [1] [2] empieza en 1950 durante el desarrollo de componentes electrónicos, protocolos y conceptos para la comunicación entre ordenadores.

Pero no es hasta la década de 1960 cuando se puso en marcha. El objetivo fue el de conectar varios centros de investigación que necesitaban compartir información entre ellas sobre un sistema de comunicación fiable. De ahí que el Gobierno de los Estados Unidos de América y su Ministerio de Defensa creara *ARPANET* (del inglés: “*Advanced Research Projects Agency Network*” con su versión castellana: “Red de la Agencia para los Proyectos de Investigación Avanzada”). Esta agencia se creó sobre la década de los 60. La organización detrás de *ARPANET* fue *ARPA* cuyas siglas en inglés significan “*Advanced Research Projects Agency*”.

En una primera instancia *ARPANET* estaba desplegada entre diferentes universidades o centros de investigación para poder conectarlas entre ellas. Debido al éxito de su funcionamiento se fueron incorporando más y más instituciones. Uno de sus principales atractivos es la descentralización de recursos. Los nodos podían comunicarse entre ellos y compartir datos incluso si uno de ellos fallaba. La arquitectura usada en Internet es la de cliente-servidor.

Durante las décadas siguientes se crearon multitud de protocolos y estándares para dar cabida a un mayor número de funcionalidades y dispositivos. *ARPANET* se estaba expandiendo y necesitaba de estándares para ello. Pero fue sobre el 1981 y con el desarrollo del concepto de *paquete de red* cuando la red *ARPANET* se empezó a expandir a gran escala. Sobre esta fecha, *ARPA*, la organización detrás de su creación, había sido absorbida por la *NSF* (del inglés: “*National Science Foundation*”). Un año más tarde en 1982, se dio a conocer el nuevo estándar por el que se dirigiría lo que hoy conocemos como *Internet*, dicho protocolo se bautizó como *TCP/IP* (del inglés: “*Transport Layer Control / Internet Protocol*”).

Tal y como se comentó anteriormente, *Internet* se basa en un modelo cliente-servidor, donde el servidor proporciona servicios a uno o varios clientes y el cliente es el demandante de estos. El cliente es el que establece entonces la comunicación en primera instancia y a través de la red, ya sea local o a través de la *WAN* (del inglés: “*Wide Area Network*”). Una vez la comunicación establecida, el servidor satisfecerá la petición del cliente, si procede, y cuando acabe dicha operación se dará por terminada la comunicación.

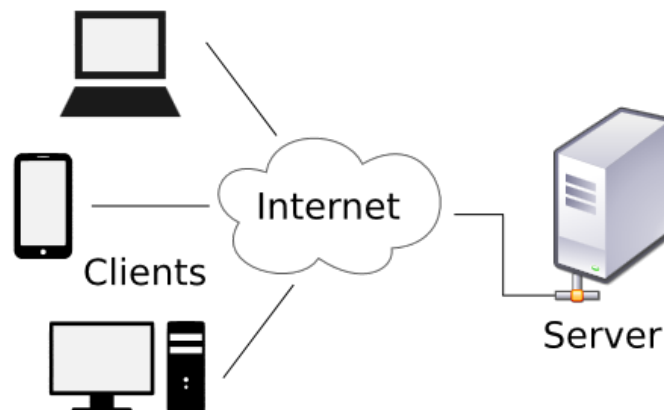


Figura 3.1: Modelo Cliente-Servidor

Pasaron los años y aunque *Internet* fue creciendo no fue hasta 1990 donde consiguió realmente revolucionar el mercado y vida social global. Expandiéndose de manera mayoritaria a todos los lugares del mundo.

Cualquier dispositivos que soportara el protocolo *TCP/IP* de *Internet* podía conectarse y comunicarse con otros dentro de esta nueva red, sin fronteras entre estados ni costes por saltar entre países.

Para ver este crecimiento podemos referir a la *Organización Internacional de las Telecomunicaciones* - ITU [9] (en inglés: “*International Telecommunications Union*”). Esta organización nos muestra el porcentaje de población conectada a *Internet* con un histórico:

	2005	2010	2020
Población mundial	6.5 mil millones	6.9 mil millones	7.8 mil millones
Sin usar Internet	84 %	70 %	40 %
Usan Internet	16 %	30 %	60 %
Internet en países en desarrollo	8 %	21 %	47 %
Internet en países desarrollados	51 %	67 %	87 %

Cuadro 3.1: Tabla de usuarios usando Internet

### 3.1.1. Pila de protocolo de Internet

Tal y como se comentó anteriormente, *Internet* funciona con la pila de protocolo conocida como *TCP/IP* - Protocolo de Control de Transmisión / Protocolo de Internet (o en inglés: “*Transmission Control Protocol / Internet Protocol*”).

Su primera implementación oficial fue en el año 1989 y está recogida en el RFC 1122 [10]. Dicho protocolo es el estándar de facto y basado en el conocido modelo *OSI* (en inglés: “*Open Systems Interconnection*”) que fue creado en primera instancia y como modelo teórico para modelar Internet. Este se desarrolló en los años 1982, en donde se implementó para ARPANET.

La pila de Internet (TCP/IP) se basa en dos ideas:

- Robustez: Capaz de enviar paquetes bien formados. Igualmente inteligente para recibir paquetes con ciertos errores.
- Extremo a extremo: Los únicos elementos que mantienen estado de la comunicación son los extremos, es decir, se entiende y se debe asumir que Internet no guarda el estado de la comunicación llevada a cabo entre dos clientes.

Este protocolo, al igual que su homólogo *OSI*, se basa en la segmentación de la comunicación en niveles y la encapsulación de la información según pasen a través de estos.

A continuación podemos ver una pequeña descripción de cada uno de los niveles de estas dos pilas de protocolos:

- Modelo OSI. Creada por *International Organization for Standardization* (ISO), y recogida en el siguiente estándar ISO/IEC 7498-1 [10]. Tiene 7 niveles diferentes: Físico, datos, red, transporte, sesión, presentación y aplicación.
- Modelo TCP/IP. Recogido en el estándar RFC 1122 [10] al que luego se añadieron otros RFC para el resto de las capas. Tenemos un total de cinco capas: Físico, datos, red, transporte y aplicación.

La imagen siguiente muestra la comparación de cada modelo y la división en niveles de la pila de protocolo de cada uno de ellos:

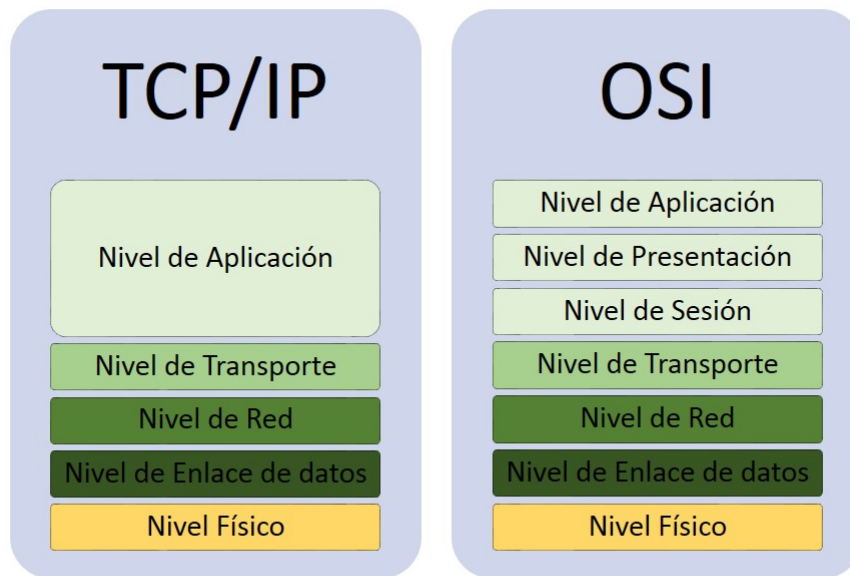


Figura 3.2: TCP/IP - OSI

La siguiente imagen describe un paquete de datos con las diferentes cabeceras por niveles según TCP/IP:

Cabecera Físico	Cabecera Red	Cabecera IP	Cabecera Transporte	Datos de Usuario
-----------------	--------------	-------------	---------------------	------------------

## 3.2. Computación en la Nube

Aunque existen diferentes definiciones sobre este concepto que pueden diferir unas de otras, *Computación en la Nube* [1] [2] [11] [12] [13] (del inglés: “*Cloud Computing*”) se puede definir básicamente como la transformación de servidores, almacenamiento y dispositivos de red en un producto escalable, consumible y auto abastecido.

La *Computación en la Nube* ofrece recursos tecnológicos de la siguiente manera:

- Sobre la red
- Como un servicio consumible

Además y normalmente tiene las siguientes características:

- Escalable. Añadiendo diferentes nodos o elementos según se necesiten a conjunto de recursos.
- Múltiples inquilinos. Varias empresas o clientes diferentes pueden disfrutar al mismo tiempo del recurso.

- Auto abastecido. Clientes pueden provisionar mayor o menor cantidad de recursos según lo necesiten.
- Pago por uso. Clientes pagan por los recursos usados y normalmente en relación con el uso en términos de computación del mismo.

Este nuevo modelo de servicio se puede desplegar en un centro de datos donde switches, routers, almacenamiento y servidores están todos localizados en el mismo sitio o geográficamente distribuidos en diferentes sitios donde cada sitio se compone de uno de estos centros de datos.

Las ventajas que los clientes finales y empresas pueden encontrar en este nuevo modelo de servicio son:

- Flexibilidad. Los clientes pueden provisionar la cantidad de recursos necesarios en cada momento cubriendo los posibles picos de carga que puedan tener en su negocio y desligarse de ellos una vez haya transcurrido dicha necesidad.
- Reducción de costes operacionales. *Computación en la Nube* permite a los clientes comenzar su negocio con un coste de entrada ínfimo si lo comparamos con lo que supondría el despliegue de recursos y mantenimiento por su cuenta. Recordando que el cliente paga por el uso que haya hecho del mismo.
- Accesibilidad. Debido a que los recursos se ofrecen a través de la red, los clientes pueden usarlos desde cualquier parte. Deslocalización geográfica es un beneficio para la empresa.
- Disponibilidad. Debido a que se puede provisionar gran cantidad de recursos, *Computación en la Nube* nos permite tener varios elementos funcionando de manera activa-activa/activa-pasiva para que en caso de fallo el servicio no se vea afectado. Esto se puede entender tanto a nivel de servicios como a nivel físico.
- Eficiencia. Puesto que Computación en la nube nos provee de servicios, los clientes y empleados pueden centrarse más en dichas tareas relegando a un segundo plano los problemas operacionales y de mantenimiento.
- Seguridad. Puesto que los servicios son ofrecidos sobre una plataforma, *Computación en Nube* puede enfocarse y automatizar procesos para mejorar y asegurar la integridad y privacidad de los datos.

### 3.2.1. Modelo de servicio

El modelo de servicio o estructura que lleva a cabo la *Computación en la Nube* determina el tipo de recursos que se proveen a cada tipo de clientes.

La computación en la nube se podría dividir según los tres tipos de clientes y servicios posibles [13]:

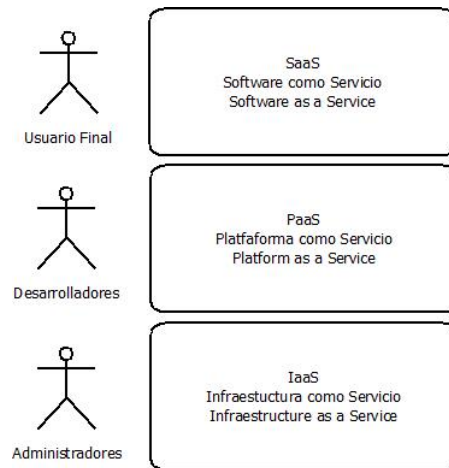


Figura 3.3: Computación en la nube

La definición de cada sección sería:

- *SaaS*: Aplicaciones que ofrecen acceso a un servicio sobre Internet. En este nivel y tipo de modelo de servicio nos encontramos a los consumidores de servicios finales. A ellos se les ofrece servicios como almacenamiento o computación online que pueden usar a su voluntad. El operador de la nube manejará el funcionamiento de ésta en una capa inferior.

En este modelo encontramos navegadores web o aplicaciones que puede ser ejecutadas en local o en la nube como *Gmail*, *Dropbox* o incluso sistemas operativos (a través de clientes ligeros) de los que el usuario maneja la capa superior pero que realmente reside y se está ejecutando externamente en la nube.

- *PaaS*: En este nivel encontramos las herramientas necesarias para que los desarrolladores creen sus aplicaciones. Este usuario recibirá por tanto una plataforma sobre la cual podrá desarrollar aplicaciones y que está provisionada sobre la nube misma. El usuario es por tanto capaz de manejar el almacenamiento, red, sistemas operativos u otros recursos que sean necesarios para crear sus aplicaciones. En este modelo nos encontramos con bases de datos como *MySQL*, servidores web como *Apache*. Algunas de las empresas que ofrecen este servicio son *Google* vía su *App Engine*.
- *IaaS*: Ofrece la infraestructura necesaria sobre la que los otros dos niveles se basan. Proveedores como *Amazon* vía su *Elastic Compute Cloud* son los encargados de ofrecernos servicios de máquinas virtuales, redes y almacenamiento en donde el servicio se cobra por uso computacional.

Una especial mención a la diferencia que hay entre *SaaS* y *PaaS* es que con *SaaS* los clientes o consumidores no crean aplicaciones sino que simplemente hacen uso de ellas.

En la imagen siguiente podemos ver que elementos y productos que podemos encontrar en cada una de estas divisiones:





Figura 3.4: Arquitectura Computación en la nube

De las tres tipos de divisiones, *IaaS* puede llegar a ser una de las más complicadas de entender, ya que por sus características es la más escondida y desconocida al conjunto de los usuarios. Aunque ya se mencionaron servicios que se sitúan en esta categoría no podemos olvidarnos de *Helion* [14] u *OpenStack* [15] como ejemplos importantes de *IaaS*.

### 3.2.2. Tipos de despliegues

Los despliegues posibles se basan en quién administrará los recursos que se van a entregar o servir.

En este sentido tenemos tres tipos de nubes o despliegues posibles:

- Nube privada - Dicha nube ofrece servicios sólo a unos usuarios específicos como miembros de una empresa o departamento. Los servicios, a través de la red, se mantienen siempre en la empresa y detrás de sus cortafuegos, todos ellos en sus servidores de datos. Normalmente no se transmiten a través de Internet los datos aquí contenidos.

Esta solución requiere el despliegue de hardware en local junto con su correspondiente mantenimiento, lo que hace que el coste se eleve. Aunque el mantenimiento puede ser llevado por una empresa externa. Ciertamente es que esta idea soluciona posibles problemas de seguridad, gobernabilidad sobre los datos, disponibilidad y control.

- Nube pública - También conocida como nube externa, la infraestructura es mantenida por el proveedor de servicios que es el dueño final de la infraestructura. Las compañías compran por tanto servicio y los usuarios acceden a dichos servicios sobre Internet u otro tipo de conexión ofrecida por el proveedor.

Nubes públicas ofrecen un uso compartido de servicios para todos los usuarios de las mismas o diferentes compañías. Las compañías normalmente pagan por uso computacional del mismo. En un sentido económico, el ahorro inicial es considerable, puesto que las compañías no tienen que hacer un gran desembolso inicial y además se desentienden del mantenimiento de la infraestructura, delegando todo ello en dicho proveedor de servicios.

Sin embargo, tener los recursos compartidos entre diferentes usuario con diferentes empresas de por medio, dilapida la seguridad y el control que las empresas tienen sobre sus datos, ya que no son los dueños finales de la infraestructura, simplemente la alquilan por uso. No saben qué ocurre de verdad con sus datos. Es la razón por la que la nube pública no es un modelo empresarial muy consolidado en aquellas empresas que tienen información altamente confidencial que no puede ser delegada o de la que carece de un control total.

- Nube comunitaria - Este tipo de nube esta compartida por diferentes organizaciones y empresas que comparten un mismo interés. Puede ser administrada tanto por una de estas organizaciones como por una externa (delegando en ella las tareas de mantenimiento)
- Nube híbrida - En dicho modelo el proveedor de servicios ofrece una nube pública al mismo tiempo que ofrece ciertos beneficios de las nubes privadas mencionadas anteriormente. Provee de un panel de administración donde servidores, red, almacenamiento y seguridad se pueden manejar individualmente por el consumidor o empresa. Normalmente también se ofrece un mayor rendimiento, disponibilidad y capacidad al igual que un mayor número de medidas de seguridad para garantizar aún más la privacidad que en las nubes públicas.

Es por tanto que en la actualidad muchas empresas buscan este modelo híbrido donde pueden tener un mayor control sobre la infraestructura que almacena sus datos pero que les permita delegar ciertas tareas de administración en terceras compañías.



Figura 3.5: Tipos de Cloud

Las nubes mencionadas anteriormente tienen todas las características siguientes en común:

- Alta accesibilidad.
- Recursos compartidos.
- Flexibilidad.
- Alta escalabilidad.
- Usan el servicio como medio.

### 3.3. Protocolos de red

Los *protocolos de red* se definen como la concreción de reglas y convenciones que permiten la comunicación entre uno o varios dispositivos de red. En ellos se usan técnicas de intercambio de paquetes para enviar y recibir información.

La razón de su existencia se debe a la necesidad de establecer un convenio común de normas que permitan la interacción de dispositivos. Pueden ser usados a una escala local o a nivel global (Internet) dependiendo de la complejidad y las necesidades.

Esta comunicación entre diferentes dispositivos de red requiere que los protocolos incluyan mecanismos de identificación para poder llevarse a cabo, normalmente recogidos en los estándares creados previos a su utilización. Estos documentos recogen los diversos tipos de mensajes que se pueden llevar a cabo para que ambas partes puedan comunicarse correctamente amén de la explicación de su funcionamiento, los más conocidos serían los RFCs (*Request for Comments*) [16]

Los siguientes protocolos de red son sólo una pequeña muestra de lo que se usa hoy en día. Han sido escogidos puesto que han sido usados en este proyecto:

- Spanning tree (o en inglés: “*Spanning Tree Protocol - STP*”)
- Protocolo de administración de red (o en inglés: “*Simple Network Management Protocol - SNMP*”).
- Seguridad de la capa de transporte (o en inglés: “*Transport Layer Security - TLS*”).
- Protocolo de resolución de direcciones (o en inglés: “*Address resolution protocol - ARP*”).
- Sistema de nombre de dominio (o en inglés: “*Domain Name System - DNS*”).

### 3.3.1. Spanning Tree Protocol

El objetivo de este protocolo es asegurar una red física libre de bucles lógicos. Al mismo tiempo debe evitar las conocidas tormentas por multidifusión de paquetes de red.

Dicho protocolo está recogido en el estándar IEEE 802.1D-2004 [1] [2] [10] [17]. Esta revisión es la última que existe en este momento y la actualmente usada en la mayoría de los dispositivos de red. La última revisión es retro compatible con las versiones anteriores [1] [2] [10] [17].

Tal y como se comentó anteriormente, este protocolo de red elimina bucles lógicos en una red física. Si partimos de una red con forma de malla, el algoritmo elimina bucles lógicos de dicha red física, cortando caminos hacia el destino que sean fútiles o no nos acerquen al destino, convirtiéndolo en una topología tipo árbol.

Ejecutándose cada vez que se modifica una conexión y realizándose a nivel de puerto. Los nodos en el sistema tienen un identificador. El sistema tiene un nodo principal o nodo raíz el cual tendrá el menor identificador de nodo. Este valor se calcula sumando la dirección física del switch conocida como dirección *MAC*, junto con la prioridad del mismo (siendo ésta configurable por el administrador). También se asigna a cada segmento un coste que depende de su distancia hasta el nodo raíz. En ciertos escenarios es posible que exista un empate entre dos caminos, en dicho caso, el que tenga mayor prioridad será el designado como candidato y camino disponible para la comunicación. El resto de puertos y segmentos de red se bloquearán y podarán para así evitar los bucles lógicos.

El algoritmo usado en *STP* consume un elevado número de recursos que aumenta cuanto más complicada es la red dada. De ahí que se haya depurado dicho algoritmo teniendo otros como “*Rapid STP Rapid Spanning Tree Protocol - RSTP*” - IEEE 802.1w [1] [2] [10]. *RSTP* es compatible con *STP*.

Otros protocolos de red ven el problema de manera diferente y solucionan el problema de los bucles de manera más eficiente. Este es el caso de *Multiple Spanning Tree Protocol - MSTP* - IEEE 802.1s [1] [2] [10] o *Per-VLAN Spanning Tree PVST* [1] [2] [10].

### 3.3.2. Protocolo Simple de Administración de Red

Dicho protocolo se encuentra en el nivel 5 o nivel de aplicación de la pila TCP/IP (véase sección: 3.1.1). Su última versión es la *SNMPv3*, recogida en el RFC 3410 [1] [2] [10]. Dicho protocolo nos permite intercambiar información de control y configuración entre dispositivos de red.

El objetivo que conseguimos es poder centralizar el mantenimiento de estos dispositivos (switches, routers, impresoras, cámaras, teléfonos IP o demás dispositivos) en un único punto donde los administradores estén al corriente de su estado y reciban periódicamente información de los mismos al igual que también puedan realizar ciertas configuraciones

remotas.

Este protocolo permite, tanto la recolección de datos desde el dispositivo, como la modificación de los mismos desde la aplicación de gestión central. Estos valores están almacenados en los dispositivos administrados y residen en variables organizadas de manera jerárquica.

La forma de acceder a ellas desde la aplicación central es conociendo previamente sus bases de datos, éstas son llamadas “Bases de información de gestión” (o en inglés: “*Management information base (MIBs)*” [1] [2]).

El escenario de este sistema de control sería:

- Servidor de gestión central donde reside la aplicación de manejo y control.
- Dispositivos de red administrados.
- Cliente software instalado en el dispositivo de red administrado que comunica con la aplicación de gestión central.

Por tanto el servidor central podrá tanto hacer llamadas para volcar o preguntar por información a los objetos de las bases de datos de los dispositivos conectados como recibir valores o modificaciones en forma de alarma o simple información, todo ello a través del agente que estos dispositivos tienen funcionando en sus sistemas.

Esta comunicación se realiza normalmente sobre UDP. La idea de usar UDP es la de no alterar y sobrecargar el funcionamiento global de la red con posibles pérdidas de datos y control de paquetería.

### 3.3.3. Seguridad en la capa de Transporte

Este protocolo de seguridad conocido por sus siglas en inglés como *TLS* (del inglés: “*Transport Layer Secure*”) [1] [2] se encuentra en la capa de transporte y justo debajo del nivel de aplicación (véase sección: 3.1.1). El objetivo es asegurar que la comunicación entre aplicaciones y usuarios sea segura (normalmente se usa a través de Internet) evitando por ejemplo que terceros puedan leer el mensaje en caso de interceptarlo.

Sucesor del bien conocido protocolo *SSL* [1] [2] (en inglés: “*Secure socket Layer*”), *TLS* en su última versión 1.2 y descrito en los RFC 5246 y RFC 6176 [10] asegura la comunicación en los siguientes pasos:

- Negociación del protocolo y algoritmo a usar.
- Intercambio de las llaves públicas y privadas (usando por ejemplo: RSA o Diffie-Hellman).
- Cifrado de la comunicación simétricamente usando RC4 o IDEA.

La pila de protocolo de dicho sistema de seguridad se puede ver en la siguiente imagen:

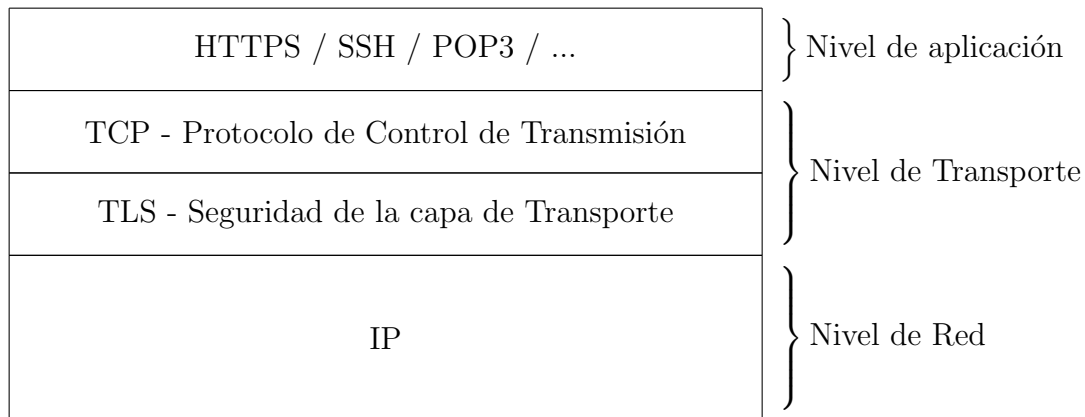


Figura 3.6: Pila de protocolo TLS

En la imagen superior se puede ver como *TLS* funciona sobre *TCP* pero dentro de dicho nivel, ofreciendo un nivel y capa de seguridad a las aplicaciones. Más información en la sección: 3.1.1.

### 3.3.4. Protocolo de resolución de direcciones

ARP [1] [18] [19] es un protocolo utilizado en el protocolo de Internet (IP) que asigna direcciones IP a las direcciones físicas (MAC). Este protocolo opera en la interfaz de red de la pila IP (sección: 3.1.1).

El término resolución de dirección apunta al proceso que se realiza para encontrar la dirección de un ordenador en red.

La resolución ocurre cuando el ordenador cliente envía una petición al servidor remoto para que este le devuelva un paquete de información con la dirección final del servidor requerida.

Una red Ethernet utiliza dos dirección físicas, una que identifica al emisor y otra el receptor. La dirección física del receptor puede variar y definir redes de difusión.

Para reducir el número de peticiones, cada elemento de red contiene una pequeña memoria que almacena la información por un corto periodo de tiempo, una vez terminado se volverá a actualizar dicha entrada en la tabla con otra petición.

La ejecución de la resolución física de una computadora sería algo como esto:

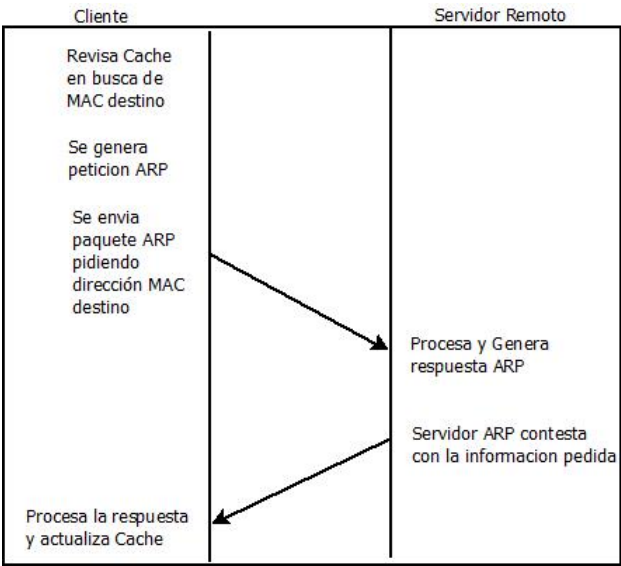


Figura 3.7: Mensaje ARP

En el siguiente esquema vemos cómo es un paquete ARP:

	Bits 0–7	Bits 8–15	Bits 16–23	Bits 24–31		
0	Tipo de dirección de hardware (HTYPE)		Tipo de protocolo de red (PTYPE)			
32	Longitud de la dirección de hardware (HLEN)	Longitud de la dirección de protocolo (PLEN)	Operación			
64	Dirección MAC del remitente					
96						
112	Dirección IP del remitente					
144	Dirección MAC del destinatario					
176						
192	Dirección IP del destinatario					

Figura 3.8: Paquete ARP

3.3.4.1. ARP Spoofing

Un ataque de tipo ARP Spoofing [1] [20] ocurre cuando un elemento de red envía paquetes ARP a un cliente o víctima intentando suplantar a un tercero, para ello sustituirá la MAC del servidor que quiere suplantar con la suya.

Después de un ataque exitoso, todo el tráfico del cliente que vaya a este supuesto host, pasará a través del atacante, este tipo de ataques se conocen como MITM ó *Man-in-the-middle* [1] [20].

Pensemos en este escenario:

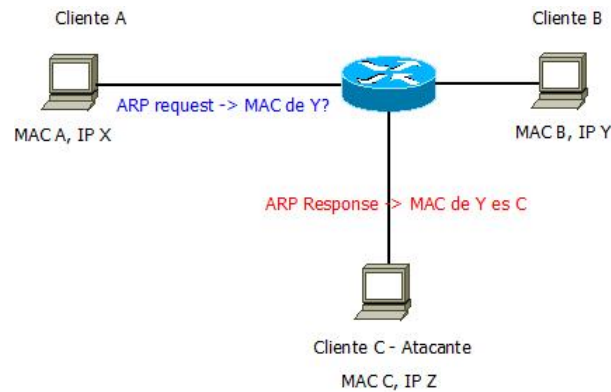


Figura 3.9: Escenario mensaje ARP

En dicho escenario el Cliente A, genera un paquete ARP de petición con esta forma:

Paquete ARP	
Protocolo	IPv4
MAC Emisor	MAC A - Cliente A
IP Emisor	IP X - Cliente A
MAC Receptor	Dirección MAC broadcast
IP Receptor	IP Y - Cliente B

Puesto que desconocemos los elementos de red y quién nos puede ayudar, el paquete se envía por difusión a todos los elementos de la red local (por eso usamos una dirección Broadcast para la MAC)

Por lo dicho anteriormente y sabiendo que todos los elementos recibirán esta petición, nuestro atacante el Cliente C, generará la respuesta ARP que envía al Cliente A con el objetivo de suplantar la identidad de B:

Paquete ARP	
Protocolo	IPv4
MAC Emisor	MAC C - Cliente C
IP Emisor	IP B - Cliente B
MAC Receptor	MAC A - Cliente A
IP Receptor	IP A - Cliente A

Con este paquete vemos como el Cliente C hace que el Cliente A asuma que la IP B tiene por dirección MAC C.



Por la definición y uso de ARP, este tipo de ataques afecta tanto a servidores como a otros elementos de red (switches, routers...) Enviar información falsa para infectar y alterar el contenido de la tabla ARP y por tanto alterar el tráfico de la red es el objetivo de este ataque.

Ataques como ARP Spoofing se dan en elementos de red que estén en la misma red local o capa 2. Este tipo de ataques pueden ser la entrada para otro tipo de ataques como los DoS (Denial of Service) [1] [2].

Una posible solución para este tipo de ataques es el uso de entradas ARP que sean estáticas, haciendo que la tabla ARP tenga valores ya predefinidos y que no los actualice. En redes empresariales esto puede ser inviable y podemos pensar en técnicas como DHCP snooping [1] [21].

### 3.4. Software-Defined Networking

*Software-defined network* (SDN) [1] [2] [11] [22] [23] [24] [25] es un nuevo modelo de arquitectura de red donde el panel de control y el panel de datos no se desarrollan en el mismo dispositivo de red.

Este nuevo diseño arquitectónico de red permite por tanto la centralización de las decisiones de control de flujo de datos en un punto externo al elemento de red. El sistema central enviará esta información de control a los dispositivos conectados para que ellos se encarguen exclusivamente de enviar el tráfico con respecto a estas órdenes.

La imagen siguiente compara la manera de controlar y manejar un dispositivo de red con el modelo actual y el modelo de *SDN*:



Figura 3.10: Modelo SDN y actual

Tal y como se puede ver en la imagen superior, *SDN* busca ofrecer a los operadores

de red acceder y controlar los elementos de la misma usando aplicaciones y técnicas de automatización vía diversos programas, sin tener que preocuparse por la capa física o la infraestructura actual de red.

Como resultado de esta nueva manera de entender la red, las futuras aplicaciones que se basen en ella podrán abstraerse de la infraestructura real y física que haya debajo, tratándola como un ente lógico y virtual. *SDN* consigue abstraerse de la capa física usando interfaces de acceso a la misma. Es así también como esta tecnología se centra en la mejora de nuestra Infraestructura como servicio (IaaS) mencionada anteriormente (véase sección: 3.2).

Alguna de las primeras aproximaciones de esta arquitectura podrían ser *WebSprocket/softswitch* [26] o *Geoplex* [27], años 1998.

### 3.4.1. Protocolos

*SDN* se puede llevar a cabo usando diversos protocolos. Existen dos partes en las que necesitamos protocolos para comunicarnos: Infraestructura-Controladora y Controladora-aplicaciones (véase imagen 3.10).

Normalmente el protocolo usado para comunicar la infraestructura física y la controladora es *Openflow*. Dicho protocolo es uno de los más conocidos y representativos pero no el único. Podemos decir que *SDN* es más que *Openflow* y *Openflow* no es sólo *SDN*. *SDN* no es un protocolo en concreto sino una nueva arquitectura de red como se comentó en el apartado anterior, debemos verlos y entenderlos como una revolución en este aspecto.

Otros protocolos que son también usados entre la infraestructura y la controladora serían: *l2RS*, *PCE-P*, *BGP-LS*, *FORCES*, *OMI*, and *NetConf/Yang*, *SNMP* [1] [2]. Todos ellos son estándares oficiales.

Por otra parte, la comunicación entre la controladora y las aplicaciones carecen de esta estandarización. En dicho campo no existe todavía un estándar oficial o referente y las compañías están creando sus propias soluciones. En estos momentos lo más habitual es encontrar desarrollos donde se usan *APIs* (sección: 3.4.4).

#### 3.4.1.1. Openflow

*Openflow* [1] [2] [28] [29] es el primero de los estándares de comunicación existentes en el mercado que definen una interfaz de comunicación entre el panel de control y el envío de datos en la arquitectura *SDN*.

Su desarrollo empezó en la *Stanford University* alrededor del 2007; ya en el 2009 se publicó la primera versión estable de este protocolo OF 1.0. Este estándar dio paso a la necesidad de crear una comunidad dedicada a este menester y en 2011 se creó la fundación

*ONF* (del inglés: “*Open Networking Foundation*”) [28].

En estos momentos muchas de las mayores empresas tecnológicas interesadas en *SDN* se han unido a esta organización y están contribuyendo en la medida de sus posibilidades.

En estos momentos, *Openflow* puede ser usada en su última versión estable: 1.5 [30]. Sin embargo, OF 1.0 y 1.3 siguen siendo las más extendidas y usadas en ámbitos empresariales.

Este nuevo estándar, permite acceso directo y manipulación de la capa de envío de datos de diferentes routers y switches, ya sean virtuales o físicos. A día de hoy no hay otro protocolo con la misma madurez que éste y que admita todas las características que éste realiza, siendo capaz de mover la capa de control fuera del dispositivo de red final a una unidad lógica central basada en software.

Una rápida comparativa de las diferentes versiones de *Openflow*, la podemos encontrar en la siguiente tabla:

Características	1.1	1.2	1.3	1.4	1.5
Tablas múltiples	X				
Grupos	X				
Etiquetas: MPLS y VLAN	X				
Puertos virtuales	X				
Fallo de conexión de la controladora	X				
Servicio extensible		X			
Compatible con IPv6 - Básico		X			
Cambio de rol de controladora		X			
Métricas por flujo			X		
Etiquetado red troncal			X		
Metadatos ID-túnel			X		
Compatible con IPv6 - extendido			X		
Bundles				X	
Puertos ópticos				X	
Sincronización de tablas				X	
Tablas de salida					X
Pipeline de paquetes					X
Openflow Extensible Statistics					X
Meter Actions					X

Cuadro 3.2: Tabla comparativa - Evolución de *Openflow*

La información detallada acerca de qué nuevas tecnologías trae la última versión de *Openflow* 1.5.1 puede verse en el siguiente documento [30].

Este protocolo especifica las diferentes operaciones y primitivas que un programa externo es capaz de usar para programar los dispositivos de red. *Openflow* tiene que ser

implementado en ambos lados de las comunicaciones para que sea posible la comunicación. Para ello se hacen uso de flujos (en inglés conocidos como “*flows*”), estos flujos de datos se usan para identificar el tráfico de red basado en las diferentes reglas ya predefinidas que pueden ser programados estática o dinámicamente por *SDN*.

La imagen siguiente escenifica dónde se encuentra dicho protocolo dentro del switch y su relación con la controladora:

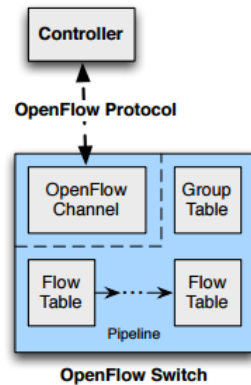


Figura 3.11: Esquema Switch Controladora en Openflow

Los flujos de datos contienen tres campos: paquete de entrada, campo para estadísticas y acciones a llevar a cabo en caso de que encontremos coincidencias en los campos señalados anteriormente.

Por otra parte cada dispositivo de red mantendrá una tabla (o varias) de flujo local (véase imagen: 3.11) que contiene una serie de campos que son en los que basará sus decisiones. Los campos más importantes serían:

- Coincidencia. Elementos a comparar con el paquete de entrada, en caso de encontrar se realizara lo que diga el campo acción.
- No coincidencia. En caso de no coincidir se enviará a la controladora.
- Acciones. En caso de que un paquete coincida con los elementos de coincidencia se realizarán las acciones listadas en este campo como por ejemplo: enviar, eliminar, difundir por todos los puertos...

Esta tabla de coincidencias puede ser completada por la controladora junto con las aplicaciones desplegadas sobre ella ya sea de manera reactiva o proactiva. Es decir, la controladora puede volcar datos nada más reconocer a un switch o esperar a que éste le pida información acerca de qué hacer con un paquete de red y entonces enviarle la información.

El funcionamiento y toma de decisiones de un switch con *Openflow* está en la siguiente imagen:

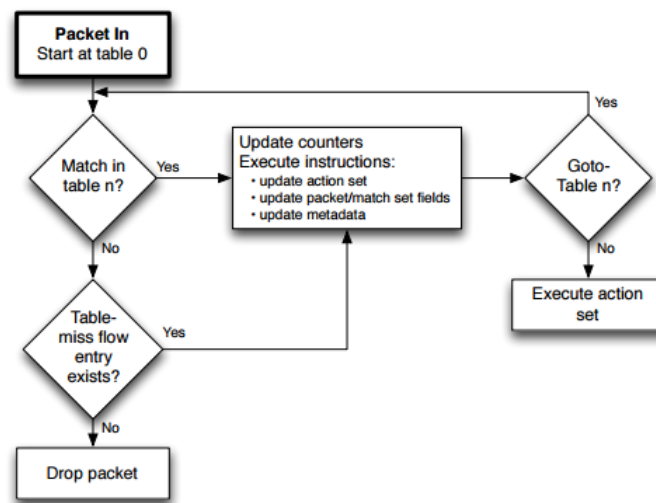


Figura 3.12: Flowchart [31]

Puesto que SDN permite ser configurable por flujos de datos, la red adquiere un grado de tecnificación y granularidad elevado.

### 3.4.2. Controladora SDN

Tal y como comentamos anteriormente *Software-defined network* (SDN) se basa en separar el panel de control y datos, donde el panel de control de datos está centralizado en una controladora.

Existen en el mercado varias controladoras, cada una de ellas tiene un abanico de posibilidades y funcionalidades diferentes. Algunas de las más importantes sería: OpenDayLight , HPE VAN SDN [32], *NOX* [33], Ryu y *POX*.



Figura 3.13: POX

Esta última controladora *POX* [1] [2] [34] [35], está escrita en Python [37] y básicamente está orientada a un uso académico y no comercial. La controladora sirve de entorno de trabajo donde los desarrolladores pueden crear sus aplicaciones y empezar a adentrarse en esta nueva tecnología. Esta controladora soporta las siguientes características:

- Soporta una interfaz “*Pythonic*” [1] [2] para poder comunicarse con las aplicaciones.

- Capacidad de usar diversos componentes de red útiles para descubrir una topología, selección de ruta, etc.
- Puesto que está basado en *Python* es multiplataforma (Windows, Linux, Mac OS).
- Soporta el mismo interfaz de usuario y herramientas de virtualización que usa NOX [33].

Actualmente puede comunicarse usando Openflow 1.0 solamente. Aunque la versión OF 1.1 podría llegar a funcionar pero sólo parcialmente.

Para el desarrollo de este trabajo se ha escogido esta controladora debido a los siguientes factores:

- Es una controladora desarrollada en Python cuyo lenguaje es cómodo de aprender y trabajar.
- El código es de libre acceso.
- Puesto que su enfoque es académico tenemos cantidad de material disponible en Internet [34] [35].

#### 3.4.2.1. Alternativas

Una de las posibles alternativas podría ser *NOX* [33]. Este proyecto nació a partir de *POX* y está recabando cada vez más adeptos. También con licencia *GNU* pero basada en este caso en *C++* en vez de *Python*.

Otra posibilidad habría sido elegir una controladora de uso comercial y/o privativa como puede ser la usada por la empresa HPE y llamada *VAN SDN Controller* [32]. Esta controladora basada en *Java* se caracteriza por tener mucho mayor rendimiento, permite despliegues con alta redundancia.

Otra posibilidad proveniente del ámbito empresarial sería *OpenDayLight* [38]. Esta controladora es también de código abierto y está desarrollada en Java. Ofrece alto rendimiento, modularidad y la posibilidad de desplegarla de manera distribuida, sin duda un candidato a *POX*.

En la siguiente tabla tenemos un estudio comparativo de las diferentes controladoras que entendemos son relevantes:

	<b>NOX</b>	<b>POX</b>	<b>Floodlight</b>	<b>OpenDayLight</b>	<b>HPE</b>
<b>Lenguaje</b>	C++	Python	Java	Java	Java
<b>Rendimiento</b>	Alto	Lento	Alto	Alto	Alto
<b>Distribuida</b>	No	No	Sí	Sí	Sí
<b>OpenFlow</b>	1.0	1.0	1.0, 1.3, 1.4	1.0, 1.3	1.0, 1.3
<b>Multi-tenant Clouds</b>	No	No	Sí	Sí	Sí
<b>Curva aprendizaje</b>	Moderado	Fácil	Difícil	Difícil	Difícil
<b>Código abierto/libre</b>	Sí	Sí	Sí	Sí	No

Cuadro 3.3: Tabla comparativa de controladoras SDN

### 3.4.3. Mininet

Emulador de red es la definición de *Mininet* [1] [2] [39] [40]. Con dicha aplicación podemos emular switches, servidores y routers. También podemos generar conexiones entre ellas creando así nuestra topología de red deseada.

*Mininet* se comporta como una máquina GNU/Linux de propósito general a la que nos podremos conectar vía *SSH* (del inglés: “*Secure Shell*”) por ejemplo. Una vez dentro de dicha máquina podremos lanzar nuestras aplicaciones como si de un entorno GNU/Linux normal se tratara.

El objetivo es poder ejecutar y poner en funcionamiento switches, router y servidores, a los que también nos podremos conectar como si de elementos independientes se trataran. Podremos por tanto crear una red virtual, configurarlos y manejar tráfico real entre ellos.

Dentro de las posibilidades que nos ofrece *Mininet* y GNU/Linux podremos configurar, generar y monitorizar el tráfico de red que nuestra red virtual podrá manejar usando herramientas externas como *WireShark* [41] u *Ostinato* [42], herramientas para la monitorización y generación de tráfico en la red.

Con *Mininet* crearemos la red virtual y su configuración y con herramientas adicionales (provistas con GNU/Linux) somos capaces de generar un tráfico de red que nuestra red virtual manejaría según nuestra configuración como si se tratase de una red real.

*Mininet* hace uso de un modelo de virtualización ligero conocido en inglés como “*light-weight virtualization*”. Con dicha técnica podemos usar un único sistema para conseguir recrear una red entera, funcionando sobre un único núcleo de Linux, sistema y usando el mismo código de usuario.

Es por tanto que *Mininet* es una de las herramientas más conocidas para crear redes virtuales a día de hoy. Tal y como se comentó antes y debido al tipo de virtualización que usa, requiere pocos recursos desde el lado del servidor donde funcionara. Aunque ello depende del tamaño y complejidad de la red.

Un ejemplo de red virtual que podemos crear en *Mininet* con herramientas como *MiniEdit*:

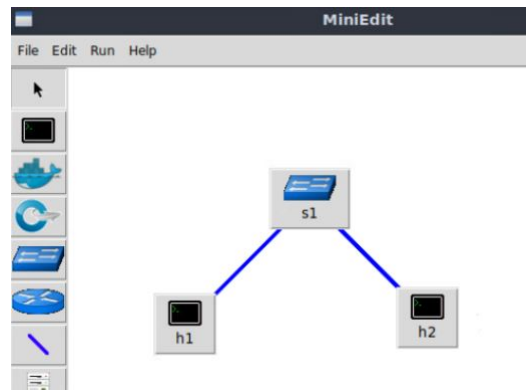


Figura 3.14: MiniEdit

### 3.4.4. RestAPI

*Transferencia de Estado Representacional* ó *Rest* (del inglés: “*Representational State Transfer - REST*”) *interfaz de programación de aplicaciones* ó *API* (del inglés: “*Application Programming Interface - API*”) [1] [2] [28] es una nuevo modo para crear, leer, actualizar, o borrar información en un servidor. Esta nueva arquitectura de software para sistemas distribuidos es la usada para manejar la conocida Internet 3.1: WWW o en inglés: “*Workd wide web*”, como modelo conocidos que usan *Restful API*.

*REST*, desarrollado por Roy Fileding, tiene por objetivo la interacción entre sistemas independientes. Permite la interacción cliente-servidor con una mínima sobrecarga en ambos extremos. En contraposición con otros protocolos como SOAP (del inglés: “*Simple Object Access Protocol*”).

Las características de este sistema de comunicaciones residen en:

- Modelos cliente servidor sin estados. La petición realizada con cada mensaje HTTP debe autocontener toda la información necesaria que haga que el receptor no necesite mantener un estado de la misma. Es decir, debe ser suficiente como para entender lo enviado.
- Un conjunto de operaciones definidas. Las operaciones que se pueden realizar deben ser especificadas con anterioridad y conocidas por ambas partes.
- Sintaxis universal. Rest hace uso de URI para poder identificas sus recursos de manera universal.
- Uso de hipermedios. para la transición de estados como para la información. Los más conocidos son XML o HTML.

Algunas de las mayores empresas que podemos encontramos hoy en día ofrecen este tipo de bibliotecas de funciones. De entre ellas podemos destacar a Facebook, Amazon y Yahoo!



Es ahí donde los desarrolladores pueden por tanto crear sus propias aplicaciones y realizar las peticiones necesarias a los servidores de dichas empresas para obtener o volcar información de o en ellos y abstraerse por medio de una capa de software de los procesos inferiores.

## 3.5. Python

*Python* [1] [2] [37] es un lenguaje de programación interpretado, con paradigma de objetos, de alto nivel y semántica dinámica.

Construido para admitir estructura de datos y con un tipado y vinculación de datos dinámicos hacen de él un lenguaje de alto nivel que además es multiplataforma. Dicho lenguaje fue creado por Guido van Rossum en 1991.

Las características que sobresalen sería la facilidad en la sintaxis, esto nos lleva a la facilidad en el aprendizaje del lenguaje, subrayando siempre la obsesión del mismo por la legibilidad del código en todo momento que hace que el futuro mantenimiento del código hecho en *Python* se reduzca.

Para sintetizar sería algo como: más fácil de leer es más fácil de entender y encontrar un posible fallo o mejora.

El intérprete de *Python* junto con la biblioteca están disponibles gratuitamente y se pueden consultar ya que pertenece al código abierto. Su última versión estable *Python* 3.9.5

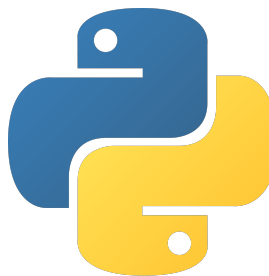


Figura 3.15: Logotipo Python

Con respecto a la emisión de errores, el intérprete de *Python* podrá emitir un error y lanzar una excepción en caso de fallo en la compilación o en caso de no detectarla en la compilación podrá seguir dicho error siguiendo la pila, con la ayuda del depurador en este caso para ayudar al desarrollador a poder encontrar el error en el código (creando punto de ruptura o ejecución paso a paso por ejemplo).

Estas razones son las que hacen de este lenguaje uno de los más atractivos para los desarrolladores de aplicaciones: aumento de la productividad en poco tiempo.

## 3.6. Diseño de dispositivos de red

En el mundo empresarial existen varias compañías que están a la vanguardia de la creación de elementos y protocolos de red. Algunas de estas empresas privadas más conocidas serían Cisco, HPE, Juniper [46].

Las empresas arriba mencionadas están a la vanguardia en la creación de hardware y software de uso empresarial y enfocado a las redes. Dispositivos de alto rendimiento y con variedad de tecnologías que ofrecen desde técnicas de virtualización hasta redundancia en las condiciones más adversas.

Todas ellas disponen de sistemas operativos creados específicamente para su hardware de la tecnología que el mercado y empresas necesitan.

### 3.6.1. Sistemas operativo red Aruba OS

La compañía *Aruba* [1] [2] fue adquirida por HPE en 2015 [1] [2]. Aruba tiene un sistema operativo el cual esta ramificado en varias distribuciones dependiendo del hardware en el que se instale. En el caso que nos interesa, tenemos ArubaOS [47], que está centrado en switches.

Aruba OS es un sistema operativo monolítico donde todos los servicios, procesos, sistema de ficheros y drivers están dentro del Kernel. [48].

Dicho sistema operativo, *ArubaOS*, tiene las siguiente características [49]:

- Protección contra ARP Spoofing, 802.1X, ACLs.
- Soporte para Openflow y posibilidad de usar Rest APIs.
- Protocolos para virtualización como: VLAN o VxLAN.
- Monitorización y control de errores en tiempo real sin interrupción del tráfico de red.

Un esquema visual de su arquitectura sería el siguiente:

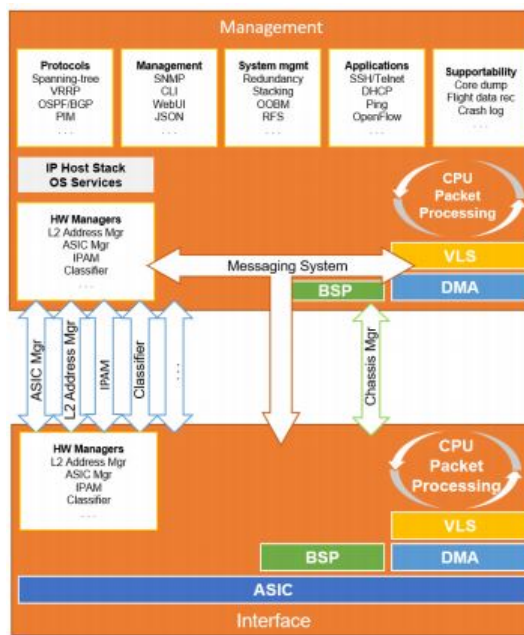


Figura 3.16: Centro de datos

Elementos como Openflow, STP ó PIM son llevados a cabo por el módulo Virtual LAN Subsystem ó VLS.

### 3.7. Centro de procesamiento de datos

Un *Centro de Procesamiento de Datos* (CPD) [1] [2] se define como el conjunto de diferentes elementos de computación como servidores, elementos de almacenamiento y elementos para su interconexión como conmutadores en un mismo espacio físico necesarios para una organización o proveedor de servicios.

La idea es concentrar en una sala todos los diferentes elementos necesarios para el procesamiento de datos para una empresa. Estos recursos y datos serán luego usados por clientes y empleados, realizándose en el mismo espacio todas las operaciones necesarias para llevar a cabo el negocio.

La intención es mejorar el rendimiento del negocio acercando datos y procesadores. Reducir coste de mantenimientos concentrando todos los elementos en un unico punto, sin olvidar que en ciertos casos se deben crear soluciones redundantes y copias de seguridad fuera del centro de datos.

Nuevas técnicas de alta disponibilidad se han desarrollado donde podemos ver por ejemplo como máquinas virtuales pueden moverse entre centro de datos dependiendo de las necesidades del cliente.

La imagen siguiente sería un ejemplo de un centro de datos:



Figura 3.17: Centro de datos

### 3.7.1. Network Functions Virtualization

Antes de la revolución que SDN trajo a las redes ya existían ejemplos que buscan virtualizar elementos dentro de los *CPDs* 3.7.

Este sentido podemos entender que *NFV Network Functions Virtualization* [1] [2] se utiliza para virtualizar los servicios de red, como los routers, cortafuegos... que tradicionalmente se ejecutaban en hardware propietario.

Estos servicios se empaquetan en máquinas virtuales que puede ejecutar en hardware estándar, lo que permite que proveedores de servicios ejecuten sus redes en servidores estándar y puedan deshacerse de sistemas propietarios.

El Instituto Europeo de Normas de Telecomunicaciones (ETSI) ha definido las normas para su implementación [43]. Cada elemento de la arquitectura en buscar estabilidad e interoperabilidad entre ellas. Tenemos Funciones de red virtualizadas, Infraestructura de virtualización de las funciones de red (NFVi) y Gestión, automatización y organización de la red (MANO).

SDN y NFV aunque son diferentes tienen ciertas similitudes. Por ejemplo, ambas usan la virtualización para poder abstraerse de la capa física. Sin embargo, SDN separan las funciones de envío y control, y delegan en una gestión central esta tarea 3.4. Por otra parte tenemos que NFV busca aislar las funciones de red del hardware donde se ejecuta. NFV proporciona la infraestructura en la cual puede ejecutarse el software de SDN.

Estas arquitecturas y conceptos pueden por tanto utilizarse a la vez para conseguir una arquitectura más flexible y programable que utilice los recursos de manera eficiente [44] [45]

# Capítulo 4

## Amenazas de seguridad en redes SDN

En este apartado trataremos de explicar algunas de las amenazas a las que las redes SDN tienen que enfrentarse hoy en día. La selección de estas amenazas se ha basado en varias listas que podemos encontrar en diferentes artículos [50] [51].

En cada una de las siguientes secciones explicaremos en detalle como funciona la amenaza y sus particularidades en el campo de SDN. También veremos y explicaremos como responder a dicha amenaza para mitigarlas.

### 4.1. ARP Spoofing en SDN

La primera de las amenazas que queremos estudiar está relacionada con el protocolo ARP. En secciones anteriores explicamos qué era el protocolo ARP y cómo funcionaba, sección 3.3.4. De igual manera también explicamos un ataque relacionado con este protocolo y llamado *ARP Spoofing*, sección 3.3.4.1, y es en el que nos centramos en esta sección.

Recordemos brevemente el funcionamiento de ARP y planteemos un escenario para una amenaza del tipo ARP Spoofing sin olvidarnos de la singularidad que una red SDN plantea.

Si hablamos de ataques ARP Spoofing podemos pensar en dos variantes en el caso de redes SDN:

- El mecanismo de difusión empleado por ARP para vincular direcciones IP a direcciones MAC hace que un elemento local que pertenezca al mismo dominio de difusión pueda suplantar la identidad de uno de ellos y redirigir el tráfico. Este tipo de ataques lo podemos encontrar en las redes actuales y es el que tratamos en la siguiente sección 4.1.1
- *Proxy ARP* [1] [2] es la manera más habitual que una red SDN controla las peticiones ARPs. La controladora central contiene y maneja la tabla MAC - IP para que el resto

de servidores no tenga que hacerlo. Esto hace que ataques dirigidos a la controladora para *envenenar* dicha tabla de traducción hagan un objetivo claro. La idea de tener un servidor Proxy es la de reducir las peticiones y el tráfico en la red. Este escenario se explicará en la sección 4.1.3.

#### 4.1.1. Escenario ARP Spoofing para SDN

En este primer escenario SDN, tendremos los elementos necesarios para desarrollar un ataque ARP Spoofing. Este es el escenario propuesto para este tipo de ataque:

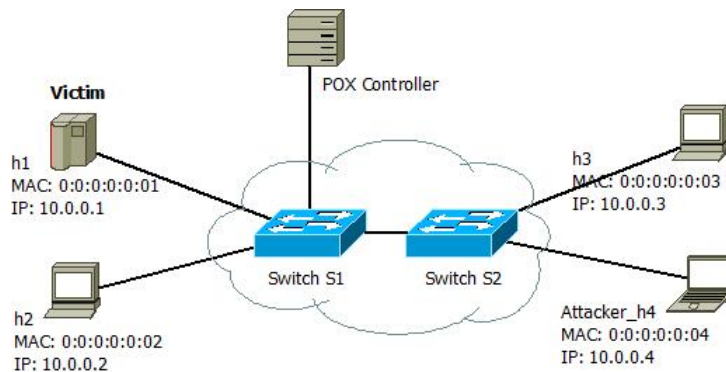


Figura 4.1: Escenario ataque ARP Spoofing

Nuestra topología para este escenario está recogida en el fichero *mininetSlice.py* que se puede encontrar en nuestro repositorio de GitHub [52].

Nuestro entorno de ejecución está explicado en el apartado *Entorno de Ejecución*, sección 4.3.

En este escenario, nuestra intención es desde la máquina atacante *h4* crear un ataque de intermediario ó *Man-in-the-middle MiTM* [1] [20]. El tráfico que el cliente ó víctima *h3* envíe al servidor *h1* pasará a través de *h4*.

Lanzamos nuestra topología en un terminal:

```
$ python2 mininetSlice.py
```

Lanzamos nuestra controladora POX en otra terminal:

```
$ pox.py forwarding.l2_learning
```

Tal y como se puede ver en la manera en que hemos lanzado la controladora POX, el control de las peticiones ARP no es centralizado sino individualmente y por cada host. Proxy ARP será tratado en la siguiente sección 4.1.3.

Tras lanzar los *scripts*, los *hosts* tienen su tabla ARP vacía ya que no se ha efectuado ninguna interacción entre ellos. Recordamos que esta información se puede obtener a través del comando *arp* dentro del *Xterm* de cada uno de los *hosts*.

Dentro de *Mininet* ejecutamos el siguiente comando:

```
mininet> xterm h1 h3 h4
```

Dentro del *Xterm* de nuestro *h4* ejecutamos la aplicación *WireShark* y *Ettercap* (4.3)

```
$ wireshark &
$ ettercap -G &
```

En *Wireshark* debemos elegir la interfaz *h4-eth1* para poder revisar el tráfico. En *Ettercap* debermos elegir la misma interfaz y añadir el *h1* y *h3* como *target* 1 y 2 respectivamente.

Dentro del *Mininet* ejecutamos le servidor web para nuestro *h1*:

```
mininet> h1 python -m SimpleHTTPServer 80 &
```

Ahora tenemos nuestro escenario preparado para el ataque. Si revisamos la tabla de ARP de nuestro *h1* justo antes de lanzar el mismo, veremos algo como lo siguiente:

Address	HWtype	HWAddresses	Flags	Mask	Iface
10.0.0.4	ether	00:00:00:00:00:04	C		h1-eth1
10.0.0.3	ether	00:00:00:00:00:03	C		h1-eth1
10.0.0.2	ether	00:00:00:00:00:02	C		h1-eth1

Cuadro 4.1: Tabla ARP - Antes del ataque ARP Spoofing

Si ahora ejecutamos *Ettercap* y actualizamos la tabla ARP de *h1* veremos estos cambios:

Address	HWtype	HWAddresses	Flags	Mask	Iface
10.0.0.4	ether	00:00:00:00:00:04	C		h1-eth1
<b>10.0.0.3</b>	ether	<b>00:00:00:00:00:04</b>	C		h1-eth1
10.0.0.2	ether	00:00:00:00:00:02	C		h1-eth1

Cuadro 4.2: Tabla ARP - Después del ataque ARP Spoofing

La situación actual de nuestra topología sería la siguiente:

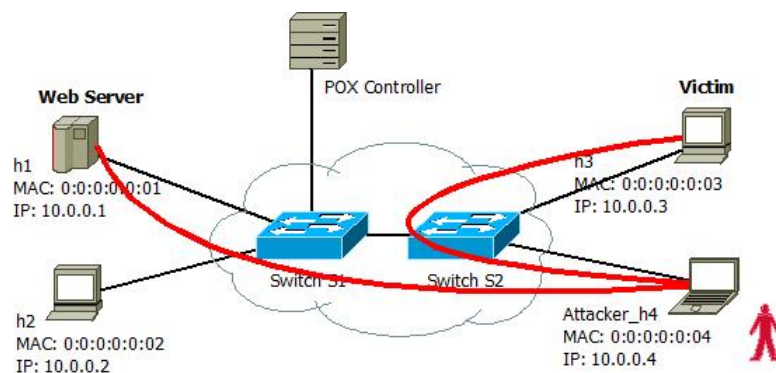


Figura 4.2: Ataque MiTM - ARP Spoofing



Podemos comprobar como todo el tráfico sigue funcionando entre *h3* y *h1* y que éste pasa a través de *h4* gracias a *Wireshark*. Por ejemplo el tráfico hacia y desde el servidor Web pasará por *h4*. Para comprobarlo y desde el terminal *Mininet* ejecutamos el siguiente comando con el hacemos una petición al servidor web:

```
mininet> h3 wget -O h1
```

Esto es lo que vemos en *Wireshark* de *h4*:

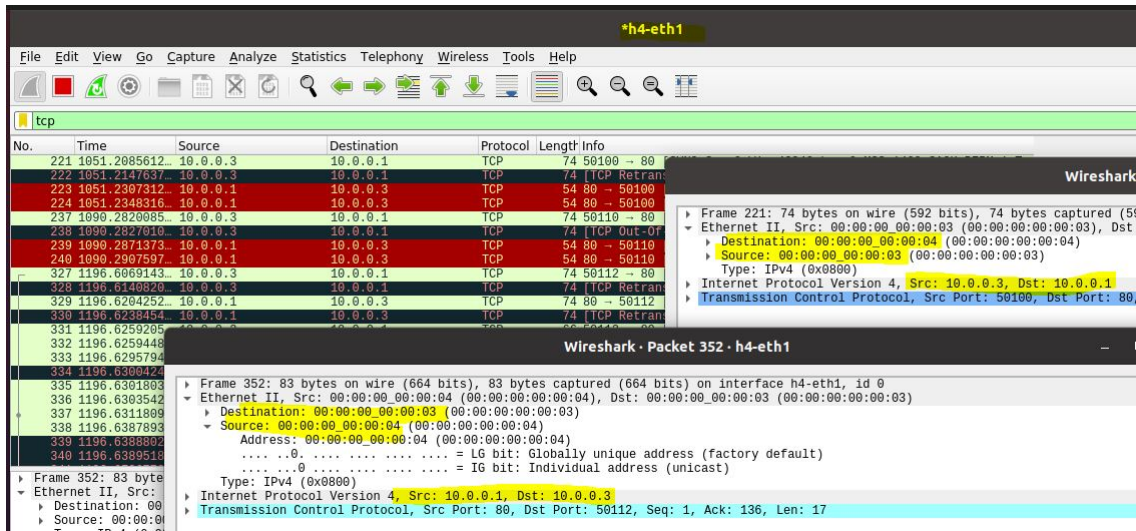


Figura 4.3: Ataque MiTM - WireShark

Podemos por tanto dar por confirmado que nuestro ataque de tipo *MiTM* [1] [2] usando ARP Spoofing es posible en redes SDN.

La misma confirmación la podemos ver en otros artículos escritos con anterioridad, alguno de los más relevantes serían:

- *A Technique for a Software-Defined and Network-based ARP Spoof Detection and Mitigation* [53].
- *Identifier Binding Attacks and Defenses in Software-Defined Networks* [54]

#### 4.1.2. Mitigar ARP Spoofing para SDN

En esta sección plantearemos algunas soluciones para mitigar *ARP Spoofing* y evitar que ataques tan fáciles de reproducir como el arriba mencionado se reproduzcan.

Para ello mencionaremos algunos de los métodos que actualmente se usan y cuál es la versión que usan las redes SDN.

Las redes tradicionales usan varios métodos que suelen estar recogidos bajo algún nombre que aglutina varios de ellos. Un ejemplo sería el de *Dynamic ARP Protection* [55], sección 3.6.1.



De entre los métodos posibles, nos centraremos en:

- Crear listas estáticas en la tabla ARP donde las direcciones IP-MAC sean fijas y no se puedan cambiar en cada uno de los hosts, sección 4.1.2.1.
- Crear puertos de confianza o *trusted* de los cuales aceptemos tráfico ARP y otros que sean revisados o bloqueados llegado el caso.

#### 4.1.2.1. Entradas estáticas IP-MAC para mitigar ARP Spoofing

Es una regla general que cada una de las entradas que componen la tabla ARP se actualiza cada cierto tiempo. Es por eso que cada dispositivo envíe paquetes ARP para actualizar dichos elementos, sección 3.3.4.

Por eso intentar envenenar esta tabla es el objetivo que persiguen los atacantes y que conseguimos en la sección anterior.

Una de las soluciones que proponemos es escribir de manera estática las entradas en dicha tabla y que los servidores relacionen inequívocamente una IP y su MAC para evitar así que terceros corrompan esta información.

Esto lo podemos conseguir con un pequeño re-diseño en la lógica de nuestra topología anterior para *Mininet* y que se encuentra en el fichero: *MitigateARPSpoofingStatic.py*, repositorio en [52]. En esta topología hemos *forzado* a los *hosts* a incluir entradas estáticas en dicha tabla que quedaría así para el *h1*:

Address	HWtype	HWAddresses	Flags Mask	Iface
10.0.0.4	ether	00:00:00:00:00:04	<b>CM</b>	h1-eth1
10.0.0.3	ether	00:00:00:00:00:03	<b>CM</b>	h1-eth1
10.0.0.2	ether	00:00:00:00:00:02	<b>CM</b>	h1-eth1

Cuadro 4.3: Tabla ARP - Entradas estáticas

Podemos confirmar como el *flag* confirma que dicha entrada ha sido manualmente introducida (sección 3.3.4).

Intentemos realizar el mismo ataque hecho en 4.1.1 desde *host h4* para comprobar si este método es efectivo.

Nuestro *hosts*, *h1* y *h3*, reciben paquetes ARP para que actualicen su tabla ARP. Sin embargo, los servidores comprueban que dichos valores ya están asignados y por tanto no lo actualizan.

En la siguiente captura de pantalla vemos que *h1* recibe paquetes ARP de *h4* con nuevos valores *envenenados* que no son actualizados en la tabla ARP:

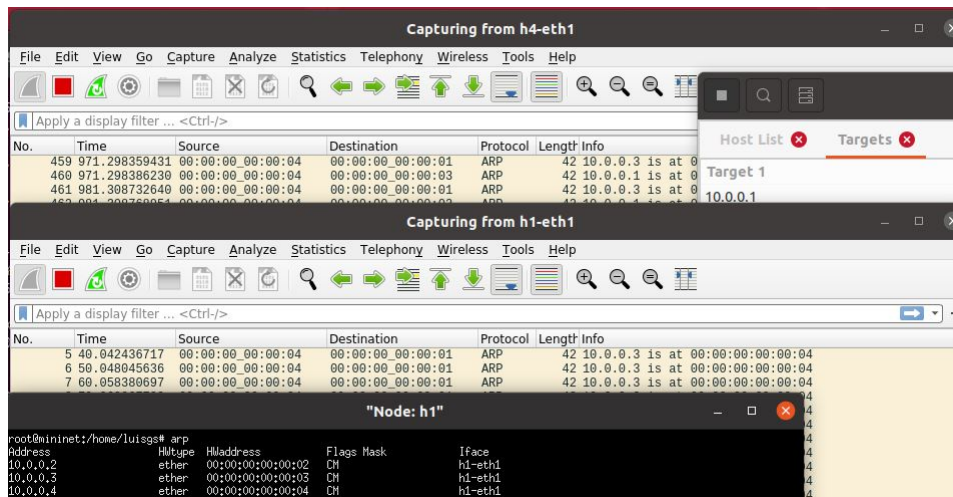


Figura 4.4: Tabla ARP - Entradas estáticas en WireShark

Por tanto podemos concluir que este pequeño método es válido para mitigar un ataque ARP Spoofing como el mencionado en 4.1.1. Sin embargo tiene ciertas limitaciones y desventajas que debemos plantearnos y tener en cuenta. Algunas de ellas serían:

- Deben ser introducidas manualmente lo que requiere de trabajo extra por parte de los administradores
- Se debe mantener una lista actualizada con estos elementos.

Si tenemos en cuenta estas desventajas, podemos comprender porque en redes con cientos de dispositivos es inviable. Más información en [56] [57].

#### 4.1.2.2. Puertos confiables para mitigar ARP Spoofing

Paquetes ARP recibidos en los puertos confiables no son revisados mientras que los paquetes ARP de los puertos no confiables son revisados contra la tabla IP-MAC.

Un primer intento de bloquear tráfico ARP proveniente de puertos no confiables podría ser:

```
mininet > sh ovs-ofctl add-flow s2 in_port=3,\
dl_type=0x806 , dl_dst=00:00:00:00:01 , actions=drop
```

Este *flow* se lanza y se inserta en el switch 2 de nuestra topología, figura 4.2. Con ello *tiramos* todo el tráfico de tipo ARP, *dl type=0x806*, con destino nuestra máquina *h1* y que entre en el puerto donde nuestro atacante está conectado. Hemos roto la redirección y el ataque *MiTM*.

Una propuesta más refinada sería enviar a nuestra controladora todos los paquetes ARP que provienen de estos puertos no confiables para que ésta los revise y responda adecuadamente.

Para ello tenemos que redirigir el tráfico ARP del puerto no confiable a nuestra controladora:

```
mininet > sh ovs-ofctl add-flow s2 in_port=3,
dl_type=0x806,actions=controller
```

### 4.1.3. Escenario Proxy ARP Spoofing para SDN

En este escenario usaremos la controladora *POX* como *Proxy ARP* [1] [2].

Nuestra controladora como *Proxy ARP* responderá a las peticiones de ARP de manera centralizada y pedirá la información en caso de carecer de ella. La controladora será por tanto la garante que los *hosts* tengan su tabla ARP correcta.

Usando nuestra controladora como *Proxy ARP* ofrece sencillez en el mantenimiento de nuestra red y evita tráfico innecesario. Sin embargo, si la tabla ARP de la controladora se ve alterada, toda la red que ésta controla está comprometida.

Este es nuestro nuevo escenario:

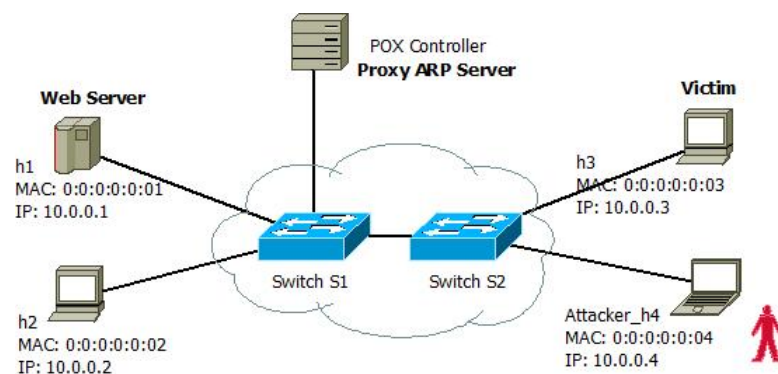


Figura 4.5: Escenario ataque Proxy ARP Spoofing

Lanzamos nuestra topología en un terminal:

```
$ python2 mininetSlice.py
```

Lanzamos nuestra controladora POX en otra terminal pero haciendo que trabaje como Proxy ARP (véase sección 3.4.3):

```
$ pox.py proto.arp_responder forwarding.l2_learning
```

Dentro de *Mininet* ejecutamos el siguiente comando:

```
mininet> xterm h1 h3 h4 s1
```

Tenemos que revisar el tráfico que viaja a nuestra controladora a través del switch *s1*.

Recordemos que ahora todos los paquetes de tipo ARP son enviados por nuestros switches a la controladora para que ésta confirme qué responder. Lo podemos confirmar

revisando los *flows* de Openflow que están instalados en cada uno de los switches de la siguiente manera:

```
mininet> dpctl dump-flows
*** s1 ***
cookie=0x0, duration=77.716s, table=0, n_packets=9, n_bytes=378, priority=28672,arp actions=CONTROLLER:65535
*** s2 ***
cookie=0x0, duration=77.719s, table=0, n_packets=12, n_bytes=504, priority=28672,arp actions=CONTROLLER:65535
```

Figura 4.6: ARP Responder y OpenFlows en cada switch

Dentro del *Xterm* de nuestro *h4* ejecutamos la aplicación *WireShark* y *Ettercap* (4.3)

```
$ wireshark &
$ ettercap -G &
```

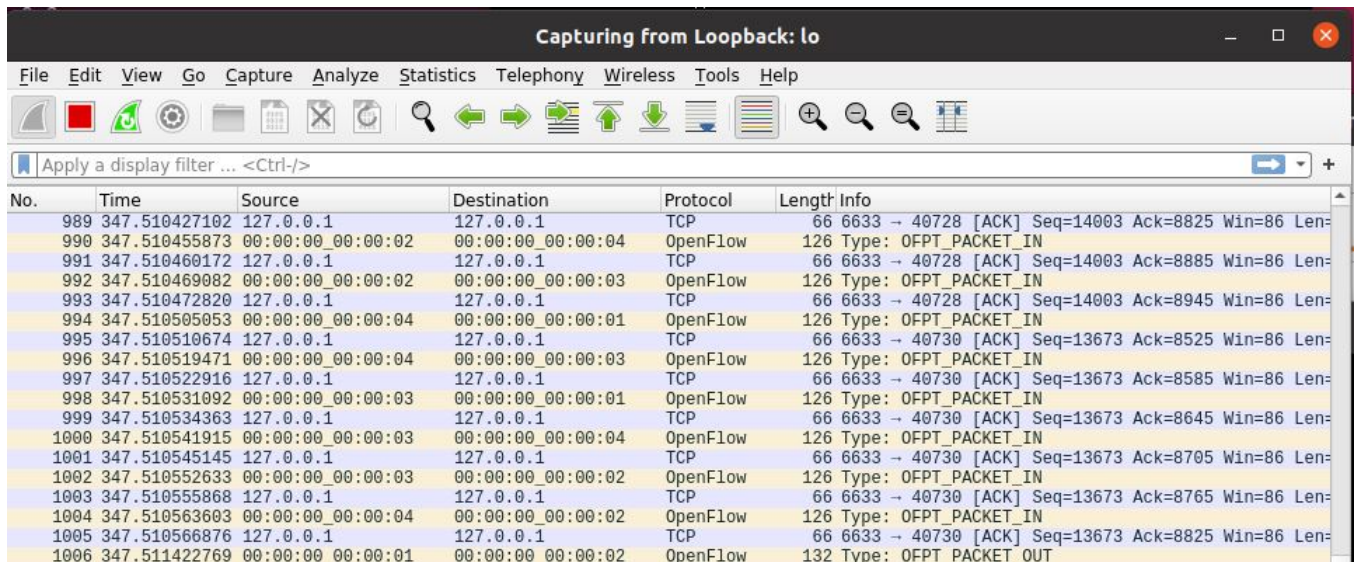
En *Wireshark* debemos elegir la interfaz *h4-eth1* para poder revisar el tráfico. En *Ettercap* debemos elegir la misma interfaz y añadir el *h1* y *h3* como *target* 1 y 2 respectivamente.

Dentro del terminal para *s1* ejecutamos la aplicación *WireShark* y seleccionamos la interfaz *loopback* que es la interfaz donde la controladora se comunica con los switches y podremos encontrar tráfico del tipo OpenFlow (4.3):

```
$ wireshark &
```

Esta sería una pequeña captura de pantalla del tráfico capturado tras lanzar este comando en *Mininet*:

```
mininet> pingall
```



No.	Time	Source	Destination	Protocol	Length	Info
989	347.510427102	127.0.0.1	127.0.0.1	TCP	66	6633 → 40728 [ACK] Seq=14003 Ack=8825 Win=86 Len=
990	347.510455873	00:00:00_00:00:02	00:00:00_00:00:04	OpenFlow	126	Type: OFPT_PACKET_IN
991	347.510460172	127.0.0.1	127.0.0.1	TCP	66	6633 → 40728 [ACK] Seq=14003 Ack=8885 Win=86 Len=
992	347.510469082	00:00:00_00:00:02	00:00:00_00:00:03	OpenFlow	126	Type: OFPT_PACKET_IN
993	347.510472820	127.0.0.1	127.0.0.1	TCP	66	6633 → 40728 [ACK] Seq=14003 Ack=8945 Win=86 Len=
994	347.510505053	00:00:00_00:00:04	00:00:00_00:00:01	OpenFlow	126	Type: OFPT_PACKET_IN
995	347.510510674	127.0.0.1	127.0.0.1	TCP	66	6633 → 40730 [ACK] Seq=13673 Ack=8525 Win=86 Len=
996	347.510519471	00:00:00_00:00:04	00:00:00_00:00:03	OpenFlow	126	Type: OFPT_PACKET_IN
997	347.510522916	127.0.0.1	127.0.0.1	TCP	66	6633 → 40730 [ACK] Seq=13673 Ack=8585 Win=86 Len=
998	347.510531092	00:00:00_00:00:03	00:00:00_00:00:01	OpenFlow	126	Type: OFPT_PACKET_IN
999	347.510534363	127.0.0.1	127.0.0.1	TCP	66	6633 → 40730 [ACK] Seq=13673 Ack=8645 Win=86 Len=
1000	347.510541915	00:00:00_00:00:03	00:00:00_00:00:04	OpenFlow	126	Type: OFPT_PACKET_IN
1001	347.510545145	127.0.0.1	127.0.0.1	TCP	66	6633 → 40730 [ACK] Seq=13673 Ack=8705 Win=86 Len=
1002	347.510552633	00:00:00_00:00:03	00:00:00_00:00:02	OpenFlow	126	Type: OFPT_PACKET_IN
1003	347.510555868	127.0.0.1	127.0.0.1	TCP	66	6633 → 40730 [ACK] Seq=13673 Ack=8765 Win=86 Len=
1004	347.510563603	00:00:00_00:00:04	00:00:00_00:00:02	OpenFlow	126	Type: OFPT_PACKET_IN
1005	347.510566876	127.0.0.1	127.0.0.1	TCP	66	6633 → 40730 [ACK] Seq=13673 Ack=8825 Win=86 Len=
1006	347.511422769	00:00:00_00:00:01	00:00:00_00:00:02	OpenFlow	132	Type: OFPT_PACKET_OUT

Figura 4.7: WireShark - Openflow

También podemos confirmar que los paquetes se envían a la controladora gracias a las instrucciones, *flows*, instaladas en los switches:

```
mininet> dpctl dump-flows
```

Dentro del *Mininet* ejecutamos el servidor web para nuestro *h1*:

```
mininet> h1 python -m SimpleHTTPServer 80 &
```

Ahora tenemos nuestro escenario preparado para el ataque. Si revisamos la tabla de ARP de nuestro *h1* justo antes de lanzar el mismo, veremos lo siguiente:

Address	HWtype	HWAddresses	Flags	Mask	Iface
10.0.0.4	ether	00:00:00:00:00:04	C		h1-eth1
10.0.0.3	ether	00:00:00:00:00:03	C		h1-eth1
10.0.0.2	ether	00:00:00:00:00:02	C		h1-eth1

Cuadro 4.4: Tabla ARP - Antes del ataque Proxy ARP Spoofing

Con nuestro escenario desplegado y las herramientas preparados podemos lanzar el ataque con *Ettercap*. Actualizamos la tabla ARP de *h1* veremos estos cambios:

Address	HWtype	HWAddresses	Flags	Mask	Iface
10.0.0.4	ether	00:00:00:00:00:04	C		h1-eth1
<b>10.0.0.3</b>	ether	<b>00:00:00:00:00:04</b>	C		h1-eth1
10.0.0.2	ether	00:00:00:00:00:02	C		h1-eth1

Cuadro 4.5: Tabla ARP - Después del ataque Proxy ARP Spoofing

La situación actual de nuestra topología sería la siguiente:

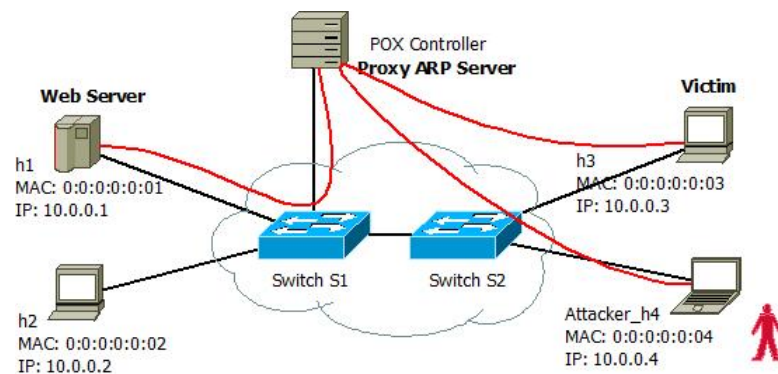


Figura 4.8: Ataque MiTM - Proxy ARP Spoofing

Podemos comprobar como todo el tráfico sigue funcionando entre *h3* y *h1* y que éste pasa a través de *h4* gracias a *WireShark*. Por ejemplo el tráfico hacia y desde el servidor Web pasará por *h4*. Para comprobarlo y desde el terminal *Mininet* ejecutamos el siguiente comando con el hacemos una petición al servidor web:

```
mininet> h3 wget -O h1
```

Esto es lo que vemos en *WireShark* de *h4*:



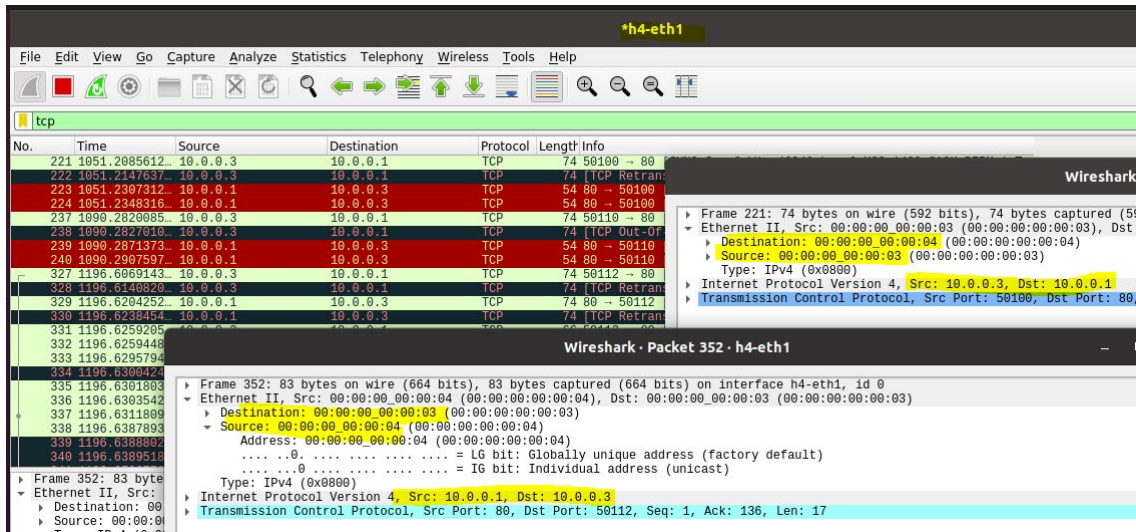


Figura 4.9: Ataque MiTM - WireShark

Al igual que hicimos en el caso anterior, podemos confirmar que dicho tipo de ataques es satisfactorio y que ha sido también explorado y confirmado en otros documentos:

- *A Technique for a Software-Defined and Network-based ARP Spoof Detection and Mitigation* [53].
- *Identifier Binding Attacks and Defenses in Software-Defined Networks* [54]

#### 4.1.4. Mitigar Proxy ARP Spoofing para SDN

Como sabemos el protocolo ARP no proporciona métodos para autenticar las respuestas ARP y esto hace que dichos paquetes ARP puedan venir de sistemas distintos o maliciosos, sección 3.3.4.1.

Un proxy ARP [1] [2] es un sistema que responde a una solicitud ARP en nombre de otro sistema para el cual reenviará el tráfico, normalmente como parte del diseño de la red. Sin embargo, un mensaje de respuesta ARP se puede falsificar fácilmente como vimos en el apartado anterior.

Para resolver este tipo de ataques debemos usar técnicas que también usamos en la sección anterior 4.1.2 y que se basa en crear una tabla donde tengamos IP-MAC.

Este tipo de técnica también la podemos ver usada en otros documentos como la siguiente tesis *Security in Software Defined Networks* [58]. En este documento vemos como se basan en las técnicas que las redes tradicionales tienen donde usamos *Dynamic ARP Inspection (DAI)*.

Nuestra solución para mitigar este ataque se centra en la misma idea: comprobar cada petición ARP contra una tabla confiable IP-MAC en nuestra controladora (que ahora funciona como Proxy ARP).

Para simplificar nuestra solución entenderemos que nuestro Proxy ARP creará una tabla inicial con todas las entradas y que luego evitará actualizarla, es decir, entenderá que son estáticas. Para ello tenemos que modificar nuestro componente *arp responder* que se encuentra en *pox/pox/proto/arp responder.py* con los siguientes cambios (visibles en nuestro repositorio, 4.3):

- Línea 177. Añadir la siguiente línea: *conflict=False*.
- Línea 198-200 debe ser comentadas. A continuación escribimos la siguientes dos líneas: *"log.warn("s is being poisoned! IT WILL NOT HAPPEN!", dpid to str(dpid))"*, a continuación escribimos: *"conflict=True"*.
- Línea 258. Modificar la línea para obtener el siguiente resultado: *"if self. check for flood(dpid, a) and not conflict:"*.

Con este método conseguimos que nuestro Proxy ARP aprenda en la primera iteración y tras hacer un *pingall* en Mininet, todas las combinaciones existentes IP-MAC pero evitaremos que actualice estos valores y que aprenda o actualice estos valores con información envenenada.

Este es el resultado en la controladora POX que nuestro nuevo y mejorado componente *arp responder* arroja tras intentar con Ettercap envenenar nuestra controladora:

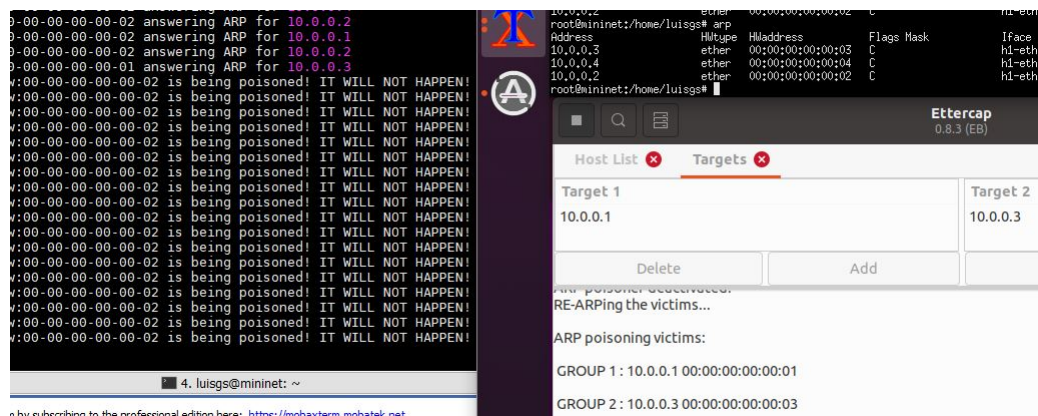


Figura 4.10: Proxy ARP siendo mitigado con ARP estática

Recordemos que la controladora está recibiendo todos los paquetes tipo ARP que nuestros switches están recibiendo, figura 4.6, y que éstos están envenenados. Nuestra controladora no actualiza la información al corroborar que el paquete contiene información fraudulenta.

Tal y como hemos mencionado antes, podríamos haber insertado los valores IP-MAC en la tabla ARP de la controladora y ahorrarnos estos cambios pero entendemos que este ejemplo añade algo de flexibilidad para esta demostración ya que simplemente tenemos que asegurarnos que nuestra controladora haga un primer estudio de los elementos de los elementos de red y confiar en que estas entradas sean correctas.

Esta solución se asemeja a la vista en los documentos siguientes *Security in Software Defined Networks* [58], en este caso crean un nuevo componente donde sí rellenan la tabla IP-MAC con información confiable. El resultado es igual.

## 4.2. Denial of Service - DoS

Un ataque DoS [1] [2] y sección 3, es un intento de que los elementos de una red no estén disponibles para su uso.

En el caso de una arquitectura SDN donde la controladora es la central de control, es obvio que los atacantes estarán enfocados en inutilizarla y de allí que los ataques DoS sean uno de los más comunes en estas redes como podemos ver en los ya mencionados documentos [50] [51] [58].

Existen dos métodos generales para un ataque DoS [59] [60] que serían por *flooding* o buscando tumbar algún servicio en particular.

### 4.2.1. Escenario DoS para SDN

Nos centraremos en un ataque DoS donde por desbordamiento hagamos que el tráfico entre dos elementos de la red sea inviable.

Nuestro escenario será el siguiente mostrado en la imagen y accesible desde nuestro repositorio [52]. En este caso particular y con el objetivo de facilitar el estudio de este tipo de ataques también exponemos el ancho de banda de cada uno de las conexiones relevantes.

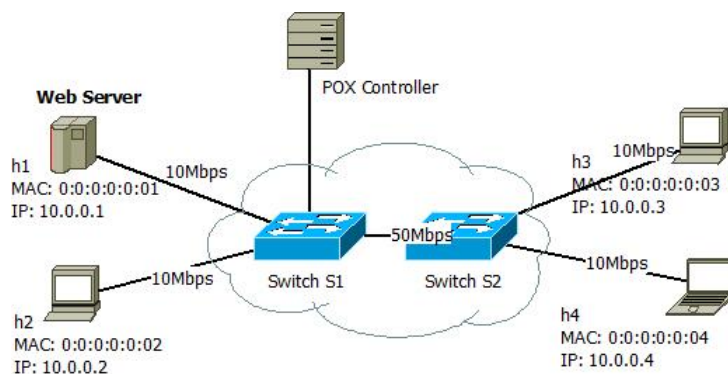


Figura 4.11: Escenario DoS para SDN

Para realizar un ataque DoS podemos pensar en varias herramientas como la ya mencionada Ettercap, sección 4.3, o el uso de *hping3* [1] [2].

Usaremos la herramienta *hping3* en este caso para atacar nuestra controladora. Queremos enviar una gran cantidad de paquetes a otra máquina para que sean reenviados a la controladora para revisión. Para ello haremos que cada paquete sea reenviado a la controladora puesto que tendrá información nueva que el switch desconozca cómo manejar:



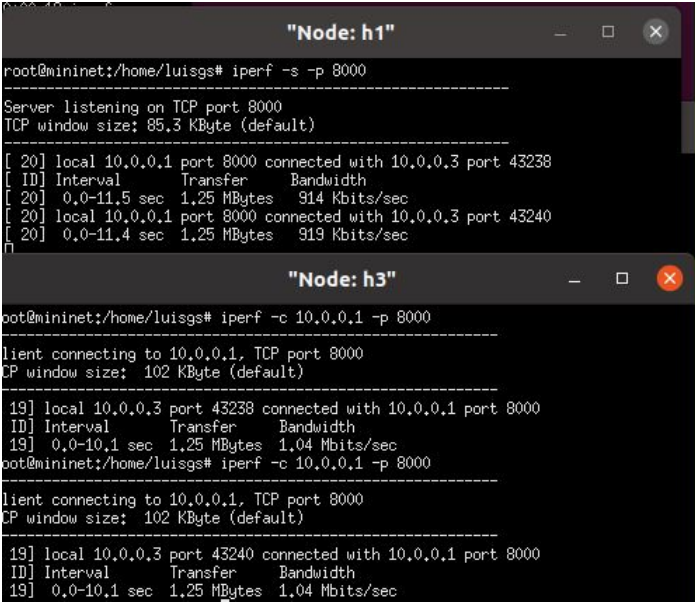
dirección IP aleatorias.

Este será el comando usado en nuestro host *h4*.

```
mininet> xterm h4
h4> hping3 --flood --rand-source 10.0.0.2
```

Antes de ejecutarlo podemos comprobar con *iperf* [1] [2] el ancho de banda actual entre dos elementos de nuestra red como son el host *h1* y el *h3*. Tendríamos el siguiente resultado tras ejecutar estos comandos en Mininet:

```
mininet> xterm h1 h3
h1> iperf -s -p 8000
h3> iperf -c 10.0.0.1 -p 8000
```



The image shows two terminal windows side-by-side. The top window is titled '"Node: h1"' and shows the output of 'iperf -s -p 8000'. It indicates the server is listening on TCP port 8000 and shows two successful connections from 10.0.0.3 with bandwidths of 914 Kbits/sec and 919 Kbits/sec. The bottom window is titled '"Node: h3"' and shows the output of 'iperf -c 10.0.0.1 -p 8000'. It indicates the client is connecting to 10.0.0.1 on TCP port 8000 and shows two successful connections to 10.0.0.3 with bandwidths of 1.04 Mbits/sec and 1.04 Mbits/sec.

```
"Node: h1"
root@mininet:/home/luigs# iperf -s -p 8000
-----
Server listening on TCP port 8000
TCP window size: 85.3 KByte (default)
-----
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43238
[ ID] Interval      Transfer    Bandwidth
[ 20] 0.0-11.5 sec  1.25 MBytes  914 Kbits/sec
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43240
[ 20] 0.0-11.4 sec  1.25 MBytes  919 Kbits/sec

"Node: h3"
root@mininet:/home/luigs# iperf -c 10.0.0.1 -p 8000
-----
Client connecting to 10.0.0.1, TCP port 8000
TCP window size: 102 KByte (default)
-----
[ 19] local 10.0.0.3 port 43238 connected with 10.0.0.1 port 8000
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-10.1 sec  1.25 MBytes  1.04 Mbits/sec
root@mininet:/home/luigs# iperf -c 10.0.0.1 -p 8000
-----
Client connecting to 10.0.0.1, TCP port 8000
TCP window size: 102 KByte (default)
-----
[ 19] local 10.0.0.3 port 43240 connected with 10.0.0.1 port 8000
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-10.1 sec  1.25 MBytes  1.04 Mbits/sec
```

Figura 4.12: IPerf entre dos host en estado normal

El resultado con *iperf* concuerda con el valor máximo dado al link entre los elementos, el código con la definición de los mismo se encuentra en nuestro repositorio [52].

Una vez que hemos explicado el escenario y comprobado el estado del mismo podemos proceder a inundar nuestra controladora con paquetes Openflow.

Tras la ejecución del comando *hping3* en nuestro host atacante *h4* podemos ver el resultado que arroja *iperf*:

```

"Node: h1"
root@mininet:/home/luisgs# iperf -s -p 8000
-----
Server listening on TCP port 8000
TCP window size: 85.3 KByte (default)
-----
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43242
[ ID] Interval      Transfer    Bandwidth
[ 20] 0.0-13.0 sec   735 KBytes  462 Kbits/sec

"Node: h3"
root@mininet:/home/luisgs# iperf -c 10.0.0.1 -p 8000
-----
Client connecting to 10.0.0.1, TCP port 8000
TCP window size: 102 KByte (default)
-----
[ 19] local 10.0.0.3 port 43242 connected with 10.0.0.1 port 8000
[ ID] Interval      Transfer    Bandwidth
[ 19] 0.0-10.5 sec   735 KBytes  573 Kbits/sec

"Node: h4"
root@mininet:/home/luisgs# hping3 --flood --rand-source 10.0.0.1
HPING 10.0.0.1 (h4-eth1 10.0.0.1): NO FLAGS are set, 40 headers + 0 data bytes
ping in flood mode, no replies will be shown

```

Figura 4.13: Resultado IPerf con hping3 activado

Según la captura de imagen superior, se ha reducido la velocidad de transmisión entre nuestros hosts. Si además revisamos el estado de nuestra controladora durante este ataque, veremos como *Wireshark* nos muestra que está siendo inundada con cientos de paquetes enviados desde nuestro switch openflow.

```

root@mininet:/home/luisgs# hping3 --flood --rand-source 10.0.0.1
HPING 10.0.0.1 (h4-eth1 10.0.0.1): NO FLAGS are set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown

```

packet.c:761: failed assertion "saved\_la  
00:41:09.753 Warn Dissector bu

**\*Loopback: lo**

No.	Time	Source	Destination	Protocol	Length	Info
18698	79.426578175	45.80.55.106	10.0.0.1	OpenFlow	138	Type: OFPT_PACKET_IN
18699	79.426583998	208.210.250.212	10.0.0.1	OpenFlow	138	Type: OFPT_PACKET_IN
18700	79.426590488	8.137.152.166	10.0.0.1	OpenFlow	138	Type: OFPT_PACKET_IN
18701	79.426596303	122.104.53.247	10.0.0.1	OpenFlow	138	Type: OFPT_PACKET_IN
18702	79.426602077	195.119.239.3	10.0.0.1	OpenFlow	138	Type: OFPT_PACKET_IN
18703	79.426607855	254.3.161.9	10.0.0.1	OpenFlow	138	Type: OFPT_PACKET_IN
18704	79.426613533	240.26.40.47	10.0.0.1	OpenFlow	138	Type: OFPT_PACKET_IN

Figura 4.14: Resultado Wireshark y hping3

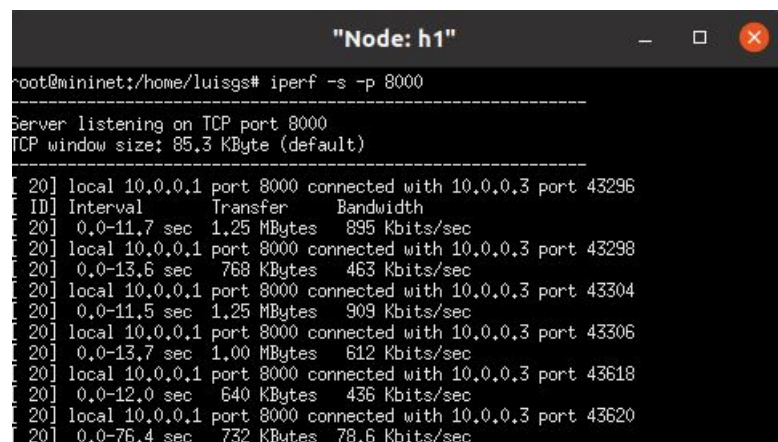
Estamos incrementando el consumo de red entre el switch y la controladora haciendo que la controladora tenga el switch envíe paquetes a la controladora y que ésta tenga que enviar *flows* al switch flows con la información acerca de cómo tratar dicho paquetes.

La falta de memoria de switches hace que entradas legítimas sean eliminadas para hacer hueco a las nuevas y tengan que ser actualizadas más adelante. Esto acompañado de la congestión de la controladora debido a esta inundación crea un efecto negativo en el rendimiento esperado.

Si ahora incluimos otro host como atacante para que también cree tráfico e inunde nuestra controladora podremos comprobar cuánto más afecta esto a nuestro escenario. Escogemos el host h2:

```
mininet> xterm h2
h2> hping3 --flood --rand-source 10.0.0.4
```

Si volvemos a estudiar el estado de conexión entre los dos mismos hosts, h1 y h3, tenemos el siguiente resultado:



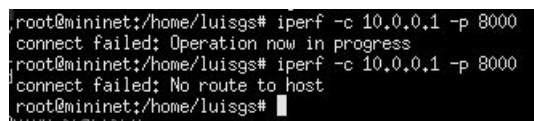
```

"Node: h1"
root@mininet:/home/luisgs# iperf -s -p 8000
-----
Server listening on TCP port 8000
TCP window size: 85.3 KByte (default)
-----
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43296
[ ID] Interval      Transfer    Bandwidth
[ 20] 0.0-11.7 sec  1.25 MBytes  895 Kbits/sec
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43298
[ 20] 0.0-13.6 sec   768 KBytes  463 Kbits/sec
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43304
[ 20] 0.0-11.5 sec  1.25 MBytes  909 Kbits/sec
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43306
[ 20] 0.0-13.7 sec   1.00 MBytes  612 Kbits/sec
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43618
[ 20] 0.0-12.0 sec   640 KBytes  436 Kbits/sec
[ 20] local 10.0.0.1 port 8000 connected with 10.0.0.3 port 43620
[ 20] 0.0-76.4 sec   732 KBytes  78.6 Kbits/sec

```

Figura 4.15: Resultado iperf con dos atacantes

Podemos ver como tras varias ejecuciones que nuestro rendimiento se ve afectado en cuanto más elementos participen en el ataque. Incluso llegar a ser inviable la conexión entre ellos debido a la congestión de nuestros switches.



```

root@mininet:/home/luisgs# iperf -c 10.0.0.1 -p 8000
connect failed: Operation now in progress
root@mininet:/home/luisgs# iperf -c 10.0.0.1 -p 8000
connect failed: No route to host
root@mininet:/home/luisgs#

```

Figura 4.16: Fallo de conexión

Este tipo de planteamientos están confirmados en varios documentos [61] [62]. En estos documentos se confirma que los ataques DoS en redes SDN son posibles.

#### 4.2.2. Mitigar DoS para SDN

Tras haber confirmado en nuestro escenario como un ataque DoS puede interferir e inutilizar llegado el caso la infraestructura exponemos algunos de los métodos que lo podrían mitigar.

La primera que podemos aplicar es limitar el número de paquetes que nuestra controladora admite [61] [63] y evitar que se sobrecargara. Debemos tener en cuenta que si un servidor es accedido por muchos clientes a la vez debemos tenerlo en cuenta para evitar estar impidiendo el uso normal de nuestra red.

Nuestra segunda propuesta sería ajustar el tiempo que un flow reside en el switch Openflow [63], podríamos alargar aquellos flows que se usen con mayor frecuencia.

Por último, podemos usar técnicas para agrupar flows bajo una misma entrada [63] [64], con lo que podemos tener más memoria para futuras entradas retrasando posibles desbordamientos y liberando recursos.

### 4.3. Entorno de ejecución

Este proyecto ha intentado usar herramientas de libre acceso y de código abierto allá donde ha sido posible. Evitar depender de terceras compañías para un futuro mantenimiento a la vez que tener una visibilidad completa de cada una de ellas son claves para este proyecto.

En este aspecto, el usar herramientas de libre acceso aumenta la facilidad a la hora de encontrar material didáctico. Esta es la lista de las herramientas utilizadas para la realización de esta parte del proyecto: Python, Vim, VirtualBox, Ubuntu/GNU Linux, Mininet, POX y Git. En las siguientes subsecciones comentaremos en detalle estos elementos junto con otros que entendemos que son relevantes.

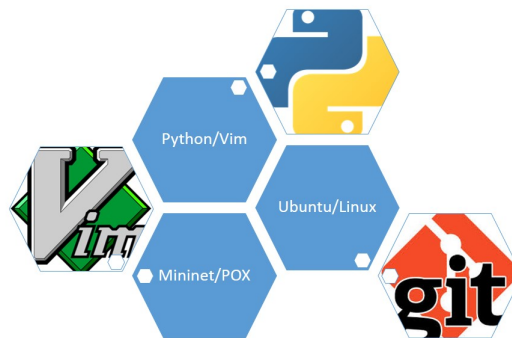


Figura 4.17: Herramientas usadas

#### 4.3.1. Controladora

La controladora elegida y sobre la cual se ha desarrollado esta aplicación es conocida como *POX* [34] (véase sección: 3.4.2).

Esta controladora con licencia *GNU* y basada en *Python* es la utilizada y recomendada por la comunidad en la introducción a esta tecnología. Su rendimiento no es remarcable pero está cerca de controladoras de mayor complejidad como puede ser *NOX* [33].

Otra de las ventajas principales de esta controladora es la posibilidad de usar componentes y módulos en nuestro código ya creados para la controladora como por ejemplo *STP* o *VLAN*, por lo que no tenemos que reescribir estas características en cada una de nuestras propia implementaciones.

La versión actual de *POX* admite sólo *Openflow* 1.0 y requiere *Python* 2.7 para su correcto funcionamiento. Esto lo podemos confirmar en su documentación [34].

#### 4.3.1.1. Alternativas

Una de las posibles controladoras sería *OpenDayLight* [38]. Una de las más usadas actualmente y bajo la Fundación Linux. Al igual que *POX*, *OpenDaylight Project* (ODL) es un proyecto de código abierto.

La controladora es multiprotocolo y modular [36]. Alguno de los protocolos que soporta serían *OpenFlow*, *Open VSwitch* (OVS) *Database* (OVSDB), *NETCONF* y *BGP*. La controladora está escrita en *Java* y ésta está contenida en su propia *Máquina Virtual de Java* que hace que sea portable a casi la totalidad de equipos y sistemas operativos.

Otras diferencias se pueden ver en la tabla antes realizada, tabla 3.3.

### 4.3.2. Lenguaje de programación

El lenguaje de programación elegido ha sido *Python* [37] (véase sección: 3.5). Multiplataforma y con paradigma de objetos, el cual encaja bien en la idea de *SDN* puesto que manejamos paquetes de datos encapsulados unos en otros y este paradigma nos ayudará con la limpieza del código y la ocultación de cierta información y librerías.

La versión usada para el desarrollo de los test está basado en *Python* 2.7, ya que es la requerida por la controladora *POX*. Existe la posibilidad de trabajar con *Python* 3, pero todavía no está mantenida oficialmente por la comunidad.

*Python* es un lenguaje multiplataforma y con una gran comunidad de desarrolladores que lo mantienen. *Python* también ayuda a poder desarrollar elementos muy potentes en poco tiempo debido a la facilidad y claridad con la que se puede programar.

#### 4.3.2.1. Alternativas

Uno de los lenguajes candidatos podría ser *Java*. Al igual que *Python*, *Java* es orientado a objetos, multiplataforma y con una gran comunidad y documentación accesibles.

El lenguaje con el que hemos trabajado está ligado a la controladora usada.

### 4.3.3. Construcción del entorno

*Mininet* [39] (véase sección: 3.4.3) es un emulador de red.

Capaz de ejecutar y lanzar servidores, switches, routers y crear conexiones entre ellos sobre una única máquina GNU-Linux. Para ello usa un tipo de virtualización ligera que consigue crear red completa sobre un mismo sistema, todos ellos funcionando con el mismo núcleo del sistema operativo y código de usuario.

Cada dispositivo creado en *Mininet* funciona como uno en la vida real. Podemos por tanto conectarnos a ellos vía *SSH* y lanzar programas dentro de ellos. Esto es particularmente útil para hacer test de conexión o pruebas de rendimiento de nuestra red, es decir, podemos usar herramientas tan conocidas como *Ettercap* o *WireShark* en cada uno de los dispositivos.

Con ello hemos podido crear diferentes redes para poner en práctica nuestros test de intrusión.

Una de las topologías con las que hemos trabajado se puede ver en la imagen 4.5. Esta topología está explicada en el *OriginalTopology.py* fichero publicado en nuestro repositorio [52].

#### 4.3.3.1. Alternativas

La alternativa sería el uso de dispositivos físicos con soporte *Openflow*. Esta solución es más fiable en cuanto a los resultados que se puedan obtener y futuros problemas en la implantación real de nuestra aplicación

Sin embargo, resulta poco asequible, ya que uno de estos dispositivos con soporte para *Openflow* puede llegar a ser muy costoso. Amén de que necesitaríamos varios de ellos para poder hacer un despliegue igual al realizado de manera virtual con *Mininet*.

### 4.3.4. Entorno de desarrollo

El editor *VIM* [65] o *VI mejorado*, ha sido suficiente para cubrir las necesidades de este proyecto. La versión usada ha sido la 7.3 a la que se han añadido el módulo de desarrollo para *Python*. Con ello hemos conseguido automatizar el sangrado del texto y acelerar la corrección de errores por reconocimiento de texto y coloreado.

*VIM* es un magnífico editor de texto vía terminal que cubre nuestras necesidades dado el tamaño de la aplicación. El lenguaje usado tampoco necesitaba de funcionalidades extras o entornos de programación más complejos.

#### 4.3.4.1. Alternativas

Otras herramientas de desarrollo podrían haber sido las herramientas de desarrollo integrado como *Visual Studio* o *Eclipse*. Estas aplicaciones gozan de multitud de herra-

mientas para *Python* amén de una interfaz de usuario más agradable.

### 4.3.5. Entorno de ejecución

*Ubuntu* es el sistema operativo con kernel de Linux escogido para hacer funcionar nuestra controladora POX y desplegar nuestra red virtual vía Mininet. Sobre dicho sistema operativo ejecutaremos tanto la controladora como crearemos la red virtual vía Mininet.

La versión de Ubuntu 20.04 LTS es la requerida para hacer funcionar Mininet y por tanto el entorno de ejecución.

Por otra parte, Ubuntu nos ofrece soporte nativo para Python, VIM y GIT con lo que la configuración es extremadamente sencilla.

#### 4.3.5.1. Alternativas

Cualquier otro sistema operativo Linux con soporte para Python 2.7 es válido. Podríamos haber escogido *Open Suse* o *Debian* ya que ambos están bajo licencia *GNU* y soportan los programas requeridos anteriormente.

### 4.3.6. Despliegue de la aplicación

El sistema al completo (controladora y red virtual) se han desplegado en máquinas virtuales *Ubuntu* funcionando sobre *Virtual Box*. *Virtual Box* es el programa para virtualizar sistemas operativos. Con ello aumentamos la portabilidad y la accesibilidad a nuestro sistema. También eliminamos los posibles errores durante la instalación y configuración del sistema operativo, ya que, está todo autocontenido en el fichero proveído.

*Virtual Box* funciona y es compatible en varios sistemas operativos y tienen licencia *GNU*.

#### 4.3.6.1. Alternativas

*VMware Workstation* es una opción muy válida. Sin embargo, carece de licencia GNU. Por otra parte, *VMware workstation* no está tan extendida en el mundo académico como nuestra primera opción.

### 4.3.7. Control de versiones

*Git* ha sido el sistema de control de versiones escogido para este proyecto. *Git* v2.33, concretamente, ha sido la versión escogida ya que es la última soportada en el sistema operativo escogido para nuestro entorno de ejecución.

*Git* es un sistema que se caracteriza por ser distribuido, rápido y compatible con muchos de los protocolos existentes en el mercado.

*Git* También se caracteriza por su presencia en el manejo de proyectos a gran escala y la increíble aceptación que ha tenido en la comunidad aun siendo un sistema de control de versiones relativamente moderno. Estas razones junto con la rapidez han hecho que sean nuestro *CVS* elegido.

#### 4.3.7.1. Alternativas

*Subversion* es una herramienta que hay que tener en cuenta cuando se busca un sistema de control de versiones centralizado. Otros sistemas como *Mercurial* también son distribuidos y habrían encajado bien en las necesidades de nuestro proyecto.

#### 4.3.8. Código fuente

El código fuente para esta parte práctica del proyecto junto con una copia de este documento puede encontrarse a través de la siguiente dirección de internet:

<https://github.com/luisgs/ThreadMySDN/>



# Capítulo 5

## CONCLUSIONES

La *Computación en la Nube* ha cambiado la manera de pensar y administrar nuestro elementos de negocio. Gracias a esta arquitectura hemos podido desarrollar ideas como la infraestructura como servicio ó *IaaS*, sección 3.2.1.

Dicha técnica nos permite ofrecer mayor flexibilidad a clientes, usuarios y empresas puesto que conseguimos una mayor automatización, seguridad, escalabilidad, productividad y seguridad [66]. Además podemos ofrecerles una reducción en los costes de capital (CAPEX) y operacionales (OPEX) [1] [2] siempre y cuando el dimensionamiento haya sido el correcto.

Una de las empresas de investigación y asesoría tecnológica más importantes e influyentes del sector como *Gartner*, Inc. (NYSE: IT) [67] ya mencionó *Cloud computing* y *Software Defined Networking* o *SDN* entre las diez tecnologías más influyentes en el año 2014 [68].

### 5.1. Planes futuros para SDN

Sin embargo, el impacto de esta nueva arquitectura parece no haber sido lo suficiente y el mundo tecnológico lo ha relegado a un segundo plano. Al menos en lo que a Openflow se refiere. *Gartner* se hace eco de esta realidad en el siguiente artículo [69].

Según Gartner las tecnologías tienen un ciclo de vida [70] que podemos ver reflejado en la imagen 5.1. Para SDN y según Gartner su momento *pico* paso hace años al igual que la fase de adopción masiva o *Plateau of Productivity* [70], dejando esta tecnología como obsoleta. En el siguiente artículo podemos leer más al respecto [71].

Esto también lo podemos confirmar por la escasez de productos en el mercado empresarial con las últimas versiones de Openflow 1.5.1, ejemplos los tenemos con empresas como Arista y Cisco. En estos dos documentos podemos confirmar que la versión de Openflow que ambas compañías trabajan es la 1.3.1 [72] [73].

Como toda nueva tecnología que intenta alcanzar la cima, sea satisfactoria o no, deja cambios y mejoras que en este caso viene en forma de *API* 3.4.4 [74].

*API* no es una revolución y podemos verlo como una evolución ya que este tipo de tecnología lleva con nosotros bastante tiempo [1] [2] pero se ha empezado a aplicar recientemente en redes empresariales, podemos ver que HPE empezó a usarlas en 2017 [75]. La tecnología se incorporó en sistema operativo mencionado anteriormente ArubaOS 3.6.1 donde vemos *Openflow* esta presente junto con *RestAPI*.

¿Será el uso de APIs la mezcla perfecta en el despliegue de arquitecturas que persigan una filosofía SDN? Artículos como los siguientes parecen apuntar a esto [74] [76]

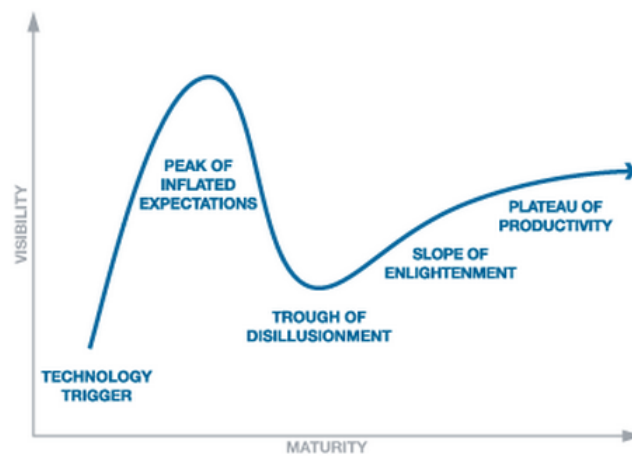


Figura 5.1: Ciclo de vida tecnológico

## 5.2. SDN y seguridad

Si nos centramos en el campo de la seguridad y su relación con SDN, podemos encontrar diversas listas con problemas de seguridad [50] [51]. El aumento de ataques año tras año no para de aumentar [7] y hará que esta lista tenga que actualizarse.

Si recordamos qué elementos componen SDN, podemos distinguir Infraestructura, Controladora y Aplicaciones 3.4. Entre estas tres capas tenemos dos interfaces que podemos resumir hablando de Openflow y RestAPI, sección 3.

SDN ha reforzado cada uno de estos elementos [78]. Tenemos el ejemplo de Openflow y la adopción de TLS en sus comunicaciones controladora-switch, sección 3.4.1.1.

Sin embargo, podemos resumir los objetivos a atacar en SDN en los siguientes elementos y es donde el futuro de SDN se decide en términos de seguridad:

- Cliente o host. Un ejemplo de ataques serían los propuestos en este proyecto: ARP 4.1.1 y Proxy ARP Spoofing, sección 4.1.3.

- Red. El objetivo de los ataques sería evitar el uso o alterar la misma red. Un ejemplo sería el estudiado en este proyecto: DoS, sección 3.
- Tráfico. Entendemos este tipo de ataques como aquellos que buscan acceder a los mensajes que transitan en la red.
- Componentes SDN. Ataques dirigidos a los pilares de esta arquitectura como son las aplicaciones, RestAPI, controladora, Openflow y switches.

## 5.3. Conclusiones

SDN puede aportar muchas ventajas para mejorar la seguridad [58] [77]. Sin embargo, SDN tiene también que enfrentarse a retos de seguridad particulares.

Este proyecto presenta una evaluación de ataques de seguridad contra clientes y de red haciendo uso de técnicas usadas en redes actuales como son ARP Spoofing y DoS. Para ello hemos utilizado las mismas técnicas y herramientas usadas en redes actuales.

Durante el estudio de ARP Spoofing, sección 4.1.1, pudimos confirmar que creando una tabla IP-MAC confiable era posible y solucionaba el problema. Solución que podemos ver hoy en día con *Dynamic ARP Inspection*. Instalar elementos fijos en la tabla ARP es una solución poco efectiva debido a la sobrecarga que eso tiene en mantener esa información actualizada. Por eso que nuestra segunda opción sería revisar este tipo de paquetes en puertos no confiables. Para por ultimo, proponer un Proxy ARP, sección 4.1.2.

La segunda parte del estudio habla de los ataques Proxy ARP Spoofing, sección 4.1.3. Pudimos demostrar que diferentes tipos de ataque en este escenario son posibles [58].

Igualmente hemos investigado un caso de Denegación De Servicio (DoS) en SDN, sección 4.2. En dicho estudio pudimos demostrar dicho ataque y analizar el impacto en nuestra red. Pudimos también proponer varias soluciones aplicadas a las redes SDN como son agrupar flows o configurar el tiempo de vida de los mismos, sección 4.2.2.

Existen variedad de artículos que también han alcanzado resultados similares, un ejemplo de ellos serían los siguientes [53] [54]. Si buscamos documentos más extensos tenemos un libro donde también podemos encontrar confirmación que respaldan nuestros hallazgos, *Software-Defined Networking and Security: From Theory to Practice* ISBN: 9781351210744 [77].

Esperamos que este trabajo proporcione la motivación necesaria para continuar investigaciones futuras. SDN puede ofrecer una red más segura si está debidamente protegida ya que tiene el potencial de revolucionar la industria de las redes.



# Bibliografía

- [1] *Wikipedia ES*, <http://es.wikipedia.org/wiki/Wikipedia:Portada>
- [2] *Wikipedia EN*, [http://en.wikipedia.org/wiki/Main\\_Page](http://en.wikipedia.org/wiki/Main_Page)
- [3] *Redes activas*, <http://arxiv.org/pdf/cs/0203014.pdf>
- [4] *Proyecto ANTS*, [http://www.cse.wustl.edu/~jain/cis788-97/ftp/active\\_nets/](http://www.cse.wustl.edu/~jain/cis788-97/ftp/active_nets/)
- [5] *Proyecto SwitchWare*, <http://www.cis.upenn.edu/~switchware/papers/switchware.ps>
- [6] *Crecimiento de Internet*, <https://purplesec.us/resources/cyber-security-statistics/>
- [7] *Crecimiento de ataques en Internet*, <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>
- [8] *Gartner*, <https://www.gartner.com/smarterwithgartner/is-the-cloud-secure/>
- [9] *ITU releases 2014 ICT figures*, [http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2014/ITU\\_Key\\_2005-2014\\_ICT\\_data.xls](http://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2014/ITU_Key_2005-2014_ICT_data.xls)
- [10] *Estándares IEEE*, <https://tools.ietf.org/>
- [11] *SDN: Software Defined Networks*. Thomas D. Nadeau & Ken Gray. Primera Edición. O'Really, 2013. 384 páginas. ISBN: 978-1-449-34230-2
- [12] *Cloud Computing en Inglés*, <http://www8.hp.com/hpnext/tags/cloud-computing#.VQgHvY7F9P0>
- [13] *Apache CloudStack Cloud Computing*. Navin Sabharwal Ravi Shankar. SBN 978-1-78216-010-6
- [14] *Helion en inglés*, <http://www8.hp.com/us/en/cloud/helion-overview.html>
- [15] *OpenStack en inglés*, <https://www.openstack.org/>

- [16] *rfcorg*. <https://www.ietf.org/standards/rfcs/>
- [17] *802.1D-2004 Spanning Tree Protocol*, <http://standards.ieee.org/getieee802/download/802.1D-2004.pdf>
- [18] *Definición de ARP*. [https://support.hpe.com/hpesc/public/docDisplay?docId=emr\\_na-c00686597](https://support.hpe.com/hpesc/public/docDisplay?docId=emr_na-c00686597)
- [19] *IETF de ARP*. <https://datatracker.ietf.org/doc/html/rfc826>
- [20] *ARP Spoofing*. [https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst\\_pon/software/configuration\\_guide/sec/b-gpon-config-security/preventing\\_arf\\_spoofing\\_and\\_flood\\_attack.html](https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst_pon/software/configuration_guide/sec/b-gpon-config-security/preventing_arf_spoofing_and_flood_attack.html)
- [21] *DHCP Snooping*. <https://community.arubanetworks.com/blogs/esupport1/2020/04/30/how-to-configure-dhcp-snooping>
- [22] *Software Defined Networks A Comprehensive Approach*. Paul Göransson y Chuck Black. Kaitlin Herbert. 333 páginas. ISBN: 978-0-12-416675-2
- [23] *Preguntas frecuentes acerca de OF por HP en inglés*, [http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA4-4714ENW.pdf?jumpid=em\\_r1165\\_ww/en/large/eg/RelatedLink/Virtual\\_Application\\_Networks\\_Overview\\_FAQs/resourcefinder/Jan\\_2013](http://h20195.www2.hp.com/V2/GetPDF.aspx/4AA4-4714ENW.pdf?jumpid=em_r1165_ww/en/large/eg/RelatedLink/Virtual_Application_Networks_Overview_FAQs/resourcefinder/Jan_2013)
- [24] *Documentación de Standorfs University acerca de Openflow y SDN*, [http://archive.openflow.org/wk/index.php/OpenFlow\\_Releases](http://archive.openflow.org/wk/index.php/OpenFlow_Releases)
- [25] *Visión de HP para SDI*, <http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA5-5770ENW.pdf>
- [26] *WebSprocket - Soft Switch*, [https://s3-us-west-2.amazonaws.com/sdnarchive/vmf\\_technicalreference12.pdf](https://s3-us-west-2.amazonaws.com/sdnarchive/vmf_technicalreference12.pdf)
- [27] *Aplicación Geoplex*, <http://www.cerias.purdue.edu/>
- [28] *Definición de OpenFlow* <https://www.opennetworking.org/sdn-resources/openflow/57-sdn-resources/onf-specifications/openflow?layout=blog>
- [29] *Portal oficial de Openflow* <http://archive.openflow.org/documents/openflow-wp-latest.pdf>
- [30] *Información sobre OpenFlow 1.5* <https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>
- [31] *Documentación de OpenFlow 1.4* <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf>

- [32] *HP VAN SDN Controller*, <http://h17007.www1.hp.com/docs/networking/solutions/sdn/4AA4-8807ENW.PDF>
- [33] *Controladora NOX*, <http://www.noxrepo.org/>
- [34] *Controladora POX*, <https://openflow.stanford.edu/display/ONL/POX+Wiki>
- [35] *Controladora POX*, <http://www.noxrepo.org/pox/about-pox/>
- [36] *Explicación de ODL* <https://www.sdxcentral.com/networking/sdn/definitions/.opendaylight-controller/>
- [37] *Página principal de Python*, <https://www.python.org/>
- [38] *OpenDayLight*. <https://www.opendaylight.org/getting-started>
- [39] *Página principal de Mininet*, <http://mininet.org/overview/>
- [40] *Introducción a Mininet*, <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet#what>
- [41] *Aplicación WireShark*, <https://www.wireshark.org/>
- [42] *Aplicación Ostinato*, <https://code.google.com/p/ostinato/>
- [43] *ETSI y NFV*, <https://www.etsi.org/technologies/nfv>
- [44] *RHEL y NFV*, <https://www.cisco.com/c/en/us/solutions/software-defined-networking/sdn-vs-nfv.html>
- [45] *Cisco y NFV*, <https://www.redhat.com/en/topics/virtualization/what-is-nfv>
- [46] *Lista Gartner empresas de redes*. <https://www.gartner.com/reviews/market/wired-wireless-lan-access-infrastructure>
- [47] *Sistema Operativo ArubaOS*. [https://www.arubanetworks.com/assets/ds/DS\\_ArubaOS.pdf](https://www.arubanetworks.com/assets/ds/DS_ArubaOS.pdf)
- [48] *Sistema monolítico vs microkernel*. <https://www.8bitavenue.com/2012/11/monolithic-vs-microkernel-os-architectures>
- [49] *Lista de tecnologías ArubaOS*. [https://support.hpe.com/hpesc/public/docDisplay?docId=a00038733en\\_us](https://support.hpe.com/hpesc/public/docDisplay?docId=a00038733en_us)
- [50] *Lista amenazas SDN* <https://www.routerfreak.com/9-types-software-defined-network-attacks-protect/>
- [51] *Lista amenazas SDN2* <https://www.blackhat.com/docs/us-16/materials/us-16-Yoon-Attacking-SDN-Infrastructure-Are-We-Ready-For-T.pdf>

- [52] *Repositorio GitHub* <https://github.com/luisgs/ThreadMySDN>
- [53] *A Technique for a Software-Defined and Network-based ARP Spoof Detection and Mitigation* [https://www.ripublication.com/ijaer18/ijaerv13n20\\_50.pdf](https://www.ripublication.com/ijaer18/ijaerv13n20_50.pdf)
- [54] *Identifier Binding Attacks and Defenses in Software-Defined Networks* <https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-jero.pdf>
- [55] *Dynamic ARP Protection en ArubaOS* [https://techhub.hpe.com/eginfolib/networking/docs/switches/WB/15-18/5998-8152\\_wb\\_2920\\_asg/content/ch11s04.html](https://techhub.hpe.com/eginfolib/networking/docs/switches/WB/15-18/5998-8152_wb_2920_asg/content/ch11s04.html)
- [56] *Huawei ARP Guide* <https://support.huawei.com/enterprise/en/doc/EDOC1100055041/ec75616e/static-arp>
- [57] *TCP Guide* [http://www.tcpipguide.com/free/t\\_ARPCaching.htm](http://www.tcpipguide.com/free/t_ARPCaching.htm)
- [58] *Security in Software Defined Networks by Talal Alharbi* <https://www.uq.edu.au/>
- [59] *Palo Alto y DoS attacks para SDN* <https://www.paloaltonetworks.com/cyberpedia/what-is-a-denial-of-service-attack-dos>
- [60] *Imperva y DoS attacks para SDN* <https://www.imperva.com/learn/ddos/ddos-attacks/>
- [61] *The effects of DoS attacks on ODL and POX SDN controllers* [https://www.researchgate.net/publication/316456006\\_The\\_effects\\_of\\_DoS\\_attacks\\_on\\_ODL\\_and\\_POX\\_SDN\\_controllers](https://www.researchgate.net/publication/316456006_The_effects_of_DoS_attacks_on_ODL_and_POX_SDN_controllers)
- [62] *Simulation Protect the Pox Controller from DDoS Attacks* <https://www.ijser.org/researchpaper/Simulation-Protect-the-Pox-Controller-from-DDoS-Attacks.pdf>
- [63] *Network Security and Management in SDN* <https://www.hindawi.com/journals/scn/2018/7545079/>
- [64] *Reducing the effects of DoS attacks* <https://link.springer.com/content/pdf/10.1186/s13673-019-0176-7.pdf>
- [65] *Aplicación VIM*, <http://www.vim.org/>
- [66] *CompTIA - IaaS*, <https://www.comptia.org/content/articles/what-is-iaas>
- [67] *Gartner, Inc. (NYSE: IT)*, <http://www.gartner.com/technology/home.jsp>



- [68] *Predicciones tecnológicas Gartner para el 2014* <http://www.networkworld.com/article/2170665/data-center/gartner--the-top-10-it-altering-predictions-for-2014.html>
- [69] *Gartner y el futuro de SDN* <https://www.networkcomputing.com/networking/sdn-time-move-gartner-says>
- [70] *Ciclo de vida de una tecnología según Gartner* <https://www.gartner.com/en/research/methodologies/gartner-hype-cycle>
- [71] *Gartner marca a SDN como obsoleta* <https://www.linkedin.com/pulse/gartner-says-sdn-has-left-building-say-hello-network-juha-holkkola/>
- [72] *Arista y Openflow 1.3* [https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1611/b\\_1611\\_programmability\\_cg/OpenFlow.html](https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1611/b_1611_programmability_cg/OpenFlow.html)
- [73] *Cisco y Openflow 1.3* [https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1611/b\\_1611\\_programmability\\_cg/OpenFlow.html](https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/1611/b_1611_programmability_cg/OpenFlow.html)
- [74] *Artículo SDN y API* <https://redneckorthodox.blogspot.com/2021/05/sdn-and-openflow-is-protocol-dead.html>
- [75] *HPE y Rest API* [https://support.hpe.com/hpesc/public/docDisplay?docLocale=en\\_US&docId=emr\\_na-c05373669](https://support.hpe.com/hpesc/public/docDisplay?docLocale=en_US&docId=emr_na-c05373669)
- [76] *Después de SDN viene API* <https://sdn-lab.com/2019/05/01/whats-left-for-software-defined-networking-after-the-hype/>
- [77] *Libro "Software-Defined Networking and Security: From Theory to Practice"* [https://www.researchgate.net/publication/329611761\\_Software-Defined\\_Networking\\_and\\_Security\\_From\\_Theory\\_to\\_Practice](https://www.researchgate.net/publication/329611761_Software-Defined_Networking_and_Security_From_Theory_to_Practice)
- [78] *Seguridad en SDN.* <https://www.acadiatech.com/blog/how-sdn-architecture-hardens-network-security/>



# Índice de figuras

3.1. Modelo Cliente-Servidor . . . . .	6
3.2. Modelo TCP/IP y OSI . . . . .	8
3.3. Modelo computacion en la nube . . . . .	10
3.4. Arquitectura Computación en la nube . . . . .	11
3.5. Tipos de Cloud . . . . .	12
3.6. Pila de protocolo TLS . . . . .	16
3.7. Mensaje ARP . . . . .	17
3.8. Paquete ARP . . . . .	17
3.9. Escenario mensaje ARP . . . . .	18
3.10. Modelo SDN y actual . . . . .	19
3.11. Esquema Switch Controladora en Openflow . . . . .	22
3.12. Flowchart . . . . .	23
3.13. Logotipo POX . . . . .	23
3.14. Topología MiniEdit . . . . .	26
3.15. Logotipo Python . . . . .	27
3.16. Diagrama de Aruba OS . . . . .	29
3.17. Centro de datos . . . . .	30
4.1. Escenario ataque ARP Spoofing . . . . .	32
4.2. Ataque MiTM - ARP Spoofing . . . . .	33
4.3. Ataque MiTM - WireShark . . . . .	34
4.4. Tabla ARP - Entradas estáticas en WireShark . . . . .	36
4.5. Escenario ataque Proxy ARP Spoofing . . . . .	37
4.6. ARP Responder y OpenFlows en cada switch . . . . .	38
4.7. WireShark - Openflow . . . . .	38
4.8. Ataque MiTM - Proxy ARP Spoofing . . . . .	39
4.9. Ataque MiTM - WireShark . . . . .	40
4.10. Proxy ARP siendo mitigado con ARP estáticas . . . . .	41
4.11. Escenario DoS para SDN . . . . .	42
4.12. IPerf entre dos host en estado normal . . . . .	43
4.13. Resultado IPerf con hping3 activado . . . . .	44
4.14. Resultado Wireshark y hping3 . . . . .	44
4.15. Resultado iperf con dos atacantes . . . . .	45
4.16. Fallo de conexión . . . . .	45
4.17. Herramientas usadas . . . . .	46

5.1. Ciclo de vida tecnológico . . . . . 52

# Índice de cuadros

3.1.	Tabla de usuarios usando Internet . . . . .	7
3.2.	Tabla comparativa - Evolución de <i>Openflow</i> . . . . .	21
3.3.	Tabla comparativa de controladoras SDN . . . . .	25
4.1.	Tabla ARP - Antes del ataque ARP Spoofing . . . . .	33
4.2.	Tabla ARP - Después del ataque ARP Spoofing . . . . .	33
4.3.	Tabla ARP - Entradas estáticas . . . . .	35
4.4.	Tabla ARP - Antes del ataque Proxy ARP Spoofing . . . . .	39
4.5.	Tabla ARP - Después del ataque Proxy ARP Spoofing . . . . .	39